# 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 Under UNIX and Apple OS X

2.6 Building FLTK Under Microsoft Windows

2.6.1 GNU toolsets (Cygwin or MinGW) hosted on Windows

2.6.2 Using the Visual C++ DLL Library

2.7 Internet Resources

2.8 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 Compiling Programs with Standard Compilers

3.3 Compiling Programs with Makefiles

3.4 Compiling Programs with Microsoft Visual C++

3.5 Naming

3.6 Header Files

---

## 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

4.7 Box Types

4.7.1 Making Your Own Boxtypes

4.8 Labels and Label Types

4.9 Callbacks

---

Generated by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.10</td>
<td>Shortcuts</td>
<td>23</td>
</tr>
<tr>
<td>1.5</td>
<td>Designing a Simple Text Editor</td>
<td>23</td>
</tr>
<tr>
<td>1.5.1</td>
<td>Determining the Goals of the Text Editor</td>
<td>24</td>
</tr>
<tr>
<td>1.5.2</td>
<td>Designing the Main Window</td>
<td>24</td>
</tr>
<tr>
<td>1.5.3</td>
<td>Variables</td>
<td>24</td>
</tr>
<tr>
<td>1.5.4</td>
<td>Menubars and Menus</td>
<td>24</td>
</tr>
<tr>
<td>1.5.5</td>
<td>Editing the Text</td>
<td>25</td>
</tr>
<tr>
<td>1.5.6</td>
<td>The Replace Dialog</td>
<td>25</td>
</tr>
<tr>
<td>1.5.7</td>
<td>Callbacks</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.1</td>
<td>changed_cb()</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.2</td>
<td>copy_cb()</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.3</td>
<td>cut_cb()</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.4</td>
<td>delete_cb()</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.5</td>
<td>find_cb()</td>
<td>26</td>
</tr>
<tr>
<td>1.5.7.6</td>
<td>find2_cb()</td>
<td>27</td>
</tr>
<tr>
<td>1.5.7.7</td>
<td>new_cb()</td>
<td>27</td>
</tr>
<tr>
<td>1.5.7.8</td>
<td>open_cb()</td>
<td>27</td>
</tr>
<tr>
<td>1.5.7.9</td>
<td>paste_cb()</td>
<td>27</td>
</tr>
<tr>
<td>1.5.7.10</td>
<td>quit_cb()</td>
<td>28</td>
</tr>
<tr>
<td>1.5.7.11</td>
<td>replace_cb()</td>
<td>28</td>
</tr>
<tr>
<td>1.5.7.12</td>
<td>replace2_cb()</td>
<td>28</td>
</tr>
<tr>
<td>1.5.7.13</td>
<td>replall_cb()</td>
<td>28</td>
</tr>
<tr>
<td>1.5.7.14</td>
<td>replcan_cb()</td>
<td>29</td>
</tr>
<tr>
<td>1.5.7.15</td>
<td>save_cb()</td>
<td>29</td>
</tr>
<tr>
<td>1.5.7.16</td>
<td>saveas_cb()</td>
<td>29</td>
</tr>
<tr>
<td>1.5.8</td>
<td>Other Functions</td>
<td>29</td>
</tr>
<tr>
<td>1.5.8.1</td>
<td>check_save()</td>
<td>30</td>
</tr>
<tr>
<td>1.5.8.2</td>
<td>load_file()</td>
<td>30</td>
</tr>
<tr>
<td>1.5.8.3</td>
<td>save_file()</td>
<td>30</td>
</tr>
<tr>
<td>1.5.8.4</td>
<td>set_title()</td>
<td>30</td>
</tr>
<tr>
<td>1.5.9</td>
<td>The main() Function</td>
<td>31</td>
</tr>
<tr>
<td>1.5.10</td>
<td>Compiling the Editor</td>
<td>31</td>
</tr>
<tr>
<td>1.5.11</td>
<td>The Final Product</td>
<td>31</td>
</tr>
<tr>
<td>1.5.12</td>
<td>Advanced Features</td>
<td>32</td>
</tr>
<tr>
<td>1.5.12.1</td>
<td>Syntax Highlighting</td>
<td>32</td>
</tr>
<tr>
<td>1.6</td>
<td>Drawing Things in FLTK</td>
<td>35</td>
</tr>
<tr>
<td>1.6.1</td>
<td>When Can You Draw Things in FLTK?</td>
<td>35</td>
</tr>
<tr>
<td>1.6.1.1</td>
<td>What Drawing Unit do FLTK drawing functions use?</td>
<td>35</td>
</tr>
<tr>
<td>1.6.2</td>
<td>Drawing Functions</td>
<td>36</td>
</tr>
<tr>
<td>1.6.2.1</td>
<td>Boxes</td>
<td>36</td>
</tr>
<tr>
<td>1.6.2.2</td>
<td>Clipping</td>
<td>37</td>
</tr>
<tr>
<td>1.6.3</td>
<td>Colors</td>
<td>38</td>
</tr>
</tbody>
</table>
1.6.3.1 Line Dashes and Thickness ................................................. 40
1.6.3.2 Drawing Fast Shapes ...................................................... 41
1.6.3.3 Drawing Complex Shapes .................................................... 43
1.6.3.4 Drawing Text ........................................................................ 45
1.6.3.5 Fonts ................................................................................. 47
1.6.3.6 Character Encoding ............................................................ 48
1.6.3.7 Drawing Overlays ............................................................... 48
1.6.4 Drawing Images ....................................................................... 48
1.6.4.1 Direct Image Drawing .......................................................... 49
1.6.4.2 Direct Image Reading ............................................................. 50
1.6.4.3 Image Classes ................................................................. 51
1.6.4.4 Offscreen Drawing .............................................................. 52
1.7 Handling Events .......................................................................... 54
1.7.1 The FLTK Event Model ............................................................ 54
1.7.2 Mouse Events ........................................................................ 54
1.7.2.1 FL_PUSH .......................................................................... 54
1.7.2.2 FL_DRAG ........................................................................... 54
1.7.2.3 FL_RELEASE ................................................................. 54
1.7.2.4 FL_MOVE .......................................................................... 54
1.7.2.5 FL_MOUSEWHEEL ........................................................... 55
1.7.3 Focus Events ........................................................................... 55
1.7.3.1 FL_ENTER .......................................................................... 55
1.7.3.2 FL_LEAVE .......................................................................... 55
1.7.3.3 FL_FOCUS ................................................................. 55
1.7.3.4 FL_UNFOCUS ..................................................................... 55
1.7.4 Keyboard Events ..................................................................... 56
1.7.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP ....................... 56
1.7.4.2 FL_SHORTCUT ................................................................. 56
1.7.5 Widget Events .......................................................................... 56
1.7.5.1 FL_DEACTIVATE ............................................................ 56
1.7.5.2 FL_ACTIVATE ................................................................. 56
1.7.5.3 FL_HIDE .......................................................................... 57
1.7.5.4 FL_SHOW ........................................................................... 57
1.7.6 Clipboard Events ..................................................................... 57
1.7.6.1 FL_PASTE .......................................................................... 57
1.7.6.2 FL_SELECTIONCLEAR ..................................................... 57
1.7.7 Drag and Drop Events ............................................................... 57
1.7.7.1 FL_DND_ENTER .............................................................. 58
1.7.7.2 FL_DND_DRAG ............................................................... 58
1.7.7.3 FL_DND_LEAVE .............................................................. 58
1.7.7.4 FL_DND_RELEASE .......................................................... 58
1.7.8 Other events ........................................................................... 58
1.14.4.2 Include Files ................................................................. 128
1.14.4.3 Description ................................................................. 128
1.14.4.4 Members ................................................................. 128
1.14.4.5 Methods ................................................................. 129

1.15 Forms Compatibility ......................................................... 129
1.15.1 Importing Forms Layout Files ............................................. 129
1.15.2 Using the Compatibility Header File ...................................... 129
1.15.3 Problems You Will Encounter ............................................. 130
1.15.4 Additional Notes .......................................................... 132

1.16 Operating System Issues ..................................................... 133
1.16.1 Accessing the OS Interfaces ............................................... 133
1.16.2 The UNIX (X11) Interface ................................................ 134
  1.16.2.1 Handling Other X Events .......................................... 134
  1.16.2.2 Drawing using Xlib ................................................. 135
  1.16.2.3 Changing the Display, Screen, or X Visual ....................... 136
  1.16.2.4 Using a Subclass of Fl_Window for Special X Stuff .............. 137
  1.16.2.5 Setting the Icon of a Window .................................... 138
  1.16.2.6 X Resources ........................................................ 139
1.16.3 The Windows (WIN32) Interface ......................................... 140
  1.16.3.1 Using filenames with non-ASCII characters ...................... 140
  1.16.3.2 Responding to WM_QUIT ............................................. 140
  1.16.3.3 Handling Other WIN32 Messages .................................. 140
  1.16.3.4 Drawing Things Using the WIN32 GDI ............................ 141
  1.16.3.5 Setting the Icon of a Window .................................... 141
  1.16.3.6 How to Not Get a MSDOS Console Window ....................... 141
  1.16.3.7 Known WIN32 Bugs and Problems ................................ 142
1.16.4 The Apple OS X Interface ............................................... 142
  1.16.4.1 Setting the icon of an application ................................ 144
  1.16.4.2 Drawing Things Using Quartz .................................... 144
  1.16.4.3 Internationalization ................................................. 144
  1.16.4.4 OpenGL and ‘retina’ displays ................................... 145
  1.16.4.5 Fl_Double_Window .................................................. 145
  1.16.4.6 Mac File System Specifics ......................................... 145
1.17 Migrating Code from FLTK 1.0 to 1.1 .................................... 146
  1.17.1 Color Values .......................................................... 146
  1.17.2 Cut and Paste Support ............................................... 146
  1.17.3 File Chooser ........................................................... 146
  1.17.4 Function Names ........................................................ 146
  1.17.5 Image Support ........................................................ 147
  1.17.6 Keyboard Navigation .................................................. 147
1.18 Migrating Code from FLTK 1.1 to 1.3 .................................... 147
  1.18.1 Migrating From FLTK 1.0 .............................................. 147
1.21.1.31 help ................................................................. 162
1.21.1.32 iconize ............................................................... 162
1.21.1.33 image ............................................................... 162
1.21.1.34 inactive ............................................................. 162
1.21.1.35 input ............................................................... 162
1.21.1.36 input_choice ...................................................... 162
1.21.1.37 keyboard ........................................................... 162
1.21.1.38 label ............................................................... 162
1.21.1.39 line_style .......................................................... 162
1.21.1.40 list_visuals ......................................................... 162
1.21.1.41 mandelbrot ......................................................... 163
1.21.1.42 menubar ............................................................ 163
1.21.1.43 message ............................................................ 163
1.21.1.44 minimum .......................................................... 163
1.21.1.45 navigation ......................................................... 163
1.21.1.46 output ............................................................. 163
1.21.1.47 overlay ............................................................ 163
1.21.1.48 pack ............................................................... 163
1.21.1.49 pixmap_browser .................................................. 163
1.21.1.50 pixmap ............................................................ 163
1.21.1.51 preferences ......................................................... 164
1.21.1.52 radio ............................................................... 164
1.21.1.53 resizebox .......................................................... 164
1.21.1.54 resize ............................................................. 164
1.21.1.55 scroll ............................................................. 164
1.21.1.56 shape ............................................................. 164
1.21.1.57 subwindow ........................................................ 164
1.21.1.58 sudoku ............................................................ 164
1.21.1.59 symbols ........................................................... 164
1.21.1.60 tabs ............................................................... 164
1.21.1.61 threads ............................................................ 165
1.21.1.62 tile ............................................................... 165
1.21.1.63 tiled_image ........................................................ 165
1.21.1.64 unittests ......................................................... 165
1.21.1.65 utf8 ............................................................... 165
1.21.1.66 valuators .......................................................... 165
1.21.1.67 fluid .............................................................. 165

1.22 FAQ (Frequently Asked Questions) ........................................ 165
1.22.1 Where do I start learning FLTK? ........................................ 165
1.22.2 How do I make a box with text? ....................................... 166
1.22.3 Can I use FLTK to make closed-source commercial applications? 166
1.22.4 Hitting the 'Escape' key closes windows - how do I prevent this? 166
8.3.2.8 enable_im() ................................................................. 203
8.3.2.9 event() ........................................................................ 203
8.3.2.10 event_button() ............................................................ 203
8.3.2.11 event_button1() .......................................................... 203
8.3.2.12 event_button2() .......................................................... 203
8.3.2.13 event_button3() .......................................................... 203
8.3.2.14 event_buttons() ........................................................... 203
8.3.2.15 event_clicks() [1/2] ....................................................... 204
8.3.2.16 event_clicks() [2/2] ....................................................... 204
8.3.2.17 event_clipboard() ......................................................... 204
8.3.2.18 event_clipboard_type() ................................................. 204
8.3.2.19 event_dispatch() ......................................................... 204
8.3.2.20 event_dx() .................................................................. 205
8.3.2.21 event_dy() .................................................................. 205
8.3.2.22 event_inside() [1/2] ...................................................... 205
8.3.2.23 event_inside() [2/2] ...................................................... 206
8.3.2.24 event_is_click() [1/2] .................................................... 206
8.3.2.25 event_is_click() [2/2] .................................................... 206
8.3.2.26 event_key() [1/2] ........................................................ 206
8.3.2.27 event_key() [2/2] ........................................................ 207
8.3.2.28 event_length() ........................................................... 207
8.3.2.29 event_original_key() .................................................. 207
8.3.2.30 event_state() [1/2] ....................................................... 207
8.3.2.31 event_state() [2/2] ....................................................... 208
8.3.2.32 event_text() ............................................................... 208
8.3.2.33 event_x_root() ............................................................ 208
8.3.2.34 event_y_root() ............................................................ 208
8.3.2.35 focus() [1/2] .............................................................. 208
8.3.2.36 focus() [2/2] .............................................................. 209
8.3.2.37 get_key() ................................................................. 209
8.3.2.38 get_mouse() .............................................................. 209
8.3.2.39 handle() ................................................................. 209
8.3.2.40 handle_() ................................................................. 210
8.3.2.41 pushed() [1/2] .......................................................... 210
8.3.2.42 pushed() [2/2] .......................................................... 210
8.3.2.43 remove_handler() ..................................................... 210
8.3.2.44 remove_system_handler() ......................................... 210
8.3.2.45 testShortcut() .......................................................... 211

8.3.3 Variable Documentation .................................................. 211
8.3.3.1 fl_eventnames .............................................................. 211
8.3.3.2 fl_fontnames .............................................................. 211

8.4 Selection & Clipboard functions ............................................ 212
8.4.1 Detailed Description ................................................................. 212
8.4.2 Function Documentation .......................................................... 212
  8.4.2.1 add_clipboard_notify() ......................................................... 212
  8.4.2.2 clipboard_contains() ......................................................... 213
  8.4.2.3 copy() ........................................................................ 213
  8.4.2.4 dnd() ........................................................................ 213
  8.4.2.5 paste() ........................................................................ 213
  8.4.2.6 paste() ........................................................................ 213
  8.4.2.7 selection() ................................................................ 214
  8.4.2.8 selection_owner() ................................................................ 214
  8.4.2.9 selection_owner() ................................................................ 214
8.5 Screen functions ........................................................................... 215
  8.5.1 Detailed Description ................................................................. 215
  8.5.2 Function Documentation .......................................................... 215
    8.5.2.1 screen_dpi() .................................................................. 215
    8.5.2.2 screen_num() .................................................................. 216
    8.5.2.3 screen_num() .................................................................. 216
    8.5.2.4 screen_work_area() ......................................................... 216
    8.5.2.5 screen_work_area() ......................................................... 216
    8.5.2.6 screen_work_area() ......................................................... 216
    8.5.2.7 screen_xywh() ................................................................ 217
    8.5.2.8 screen_xywh() ................................................................ 217
    8.5.2.9 screen_xywh() ................................................................ 218
    8.5.2.10 screen_xywh() ................................................................. 218
8.6 Color & Font functions ................................................................... 218
  8.6.1 Detailed Description ................................................................. 220
  8.6.2 Function Documentation .......................................................... 220
    8.6.2.1 fl_color() ...................................................................... 220
    8.6.2.2 fl_color() ...................................................................... 220
    8.6.2.3 fl_color() ...................................................................... 220
    8.6.2.4 fl_color_average() ............................................................ 221
    8.6.2.5 fl_contrast() ................................................................... 221
    8.6.2.6 fl_font() ....................................................................... 221
    8.6.2.7 fl_font() ....................................................................... 221
    8.6.2.8 fl_height() ..................................................................... 221
    8.6.2.9 fl_height() ..................................................................... 222
    8.6.2.10 fl_latin1_to_local() .......................................................... 222
    8.6.2.11 fl_local_to_latin1() ........................................................... 222
    8.6.2.12 fl_local_to_mac_roman() .................................................. 222
    8.6.2.13 fl_mac_roman_to_local() .................................................... 223
    8.6.2.14 fl_show_colormap() ............................................................ 223
    8.6.2.15 fl_size() ....................................................................... 224
8.6.2.16 fl_text_extents() [1/2] ........................................ 224
8.6.2.17 fl_text_extents() [2/2] ........................................ 224
8.6.2.18 fl_width() ...................................................... 224
8.6.2.19 fl_xpixel() [1/2] .............................................. 225
8.6.2.20 fl_xpixel() [2/2] .............................................. 225
8.6.2.21 free_color() .................................................... 225
8.6.2.22 get_color() [1/2] .............................................. 225
8.6.2.23 get_color() [2/2] .............................................. 226
8.6.2.24 get_font() ...................................................... 226
8.6.2.25 get_font_name() ............................................... 226
8.6.2.26 get_font_sizes() ............................................... 226
8.6.2.27 set_color() [1/2] .............................................. 226
8.6.2.28 set_color() [2/2] .............................................. 227
8.6.2.29 set_font() ...................................................... 227
8.6.2.30 set_fonts() ...................................................... 227

8.7 Drawing functions ..................................................... 227

8.7.1 Detailed Description ............................................... 231
8.7.2 Macro Definition Documentation .................................. 232
  8.7.2.1 fl_clip .......................................................... 232
8.7.3 Enumeration Type Documentation .................................. 232
  8.7.3.1 anonymous enum .............................................. 232
8.7.4 Function Documentation ........................................... 232
  8.7.4.1 copy_offscreen() ............................................. 232
  8.7.4.2 fl_add_symbol() ............................................... 232
  8.7.4.3 fl_arc() [1/2] ................................................ 233
  8.7.4.4 fl_arc() [2/2] ................................................ 233
  8.7.4.5 fl_begin_complex_polygon() .................................. 234
  8.7.4.6 fl_begin_offscreen() .......................................... 234
  8.7.4.7 fl_begin_points() ............................................ 234
  8.7.4.8 fl_can_do_alpha_blending() .................................. 234
  8.7.4.9 fl_circle() .................................................... 234
  8.7.4.10 fl_clip_box() ............................................... 235
  8.7.4.11 fl_clip_region() ............................................ 235
  8.7.4.12 fl_copy_offscreen() ......................................... 235
  8.7.4.13 fl_create_offscreen() ...................................... 236
  8.7.4.14 fl_cursor() .................................................. 236
  8.7.4.15 fl_curve() ................................................... 236
  8.7.4.16 fl_delete_offscreen() ...................................... 236
  8.7.4.17 fl_draw() [1/4] .............................................. 237
  8.7.4.18 fl_draw() [2/4] .............................................. 237
  8.7.4.19 fl_draw() [3/4] .............................................. 237
  8.7.4.20 fl_draw() [4/4] .............................................. 237
8.7.4.21 fl_draw_box() ................................................. 237
8.7.4.22 fl_draw_image() [1/2] ........................................ 238
8.7.4.23 fl_draw_image() [2/2] ........................................ 238
8.7.4.24 fl_draw_image_mono() [1/2] ................................. 239
8.7.4.25 fl_draw_image_mono() [2/2] ................................. 239
8.7.4.26 fl_draw_pixmap() [1/2] ...................................... 240
8.7.4.27 fl_draw_pixmap() [2/2] ...................................... 240
8.7.4.28 fl_draw_symbol() ............................................. 240
8.7.4.29 fl_expand_text() ............................................. 241
8.7.4.30 fl_frame() ..................................................... 241
8.7.4.31 fl_frame2() ..................................................... 241
8.7.4.32 fl_gap() ....................................................... 242
8.7.4.33 fl_line_style() ............................................... 242
8.7.4.34 fl_measure() .................................................. 242
8.7.4.35 fl_measure_pixmap() [1/2] .................................. 243
8.7.4.36 fl_measure_pixmap() [2/2] .................................. 243
8.7.4.37 fl_mult_matrix() ............................................. 243
8.7.4.38 fl_not_clipped() ............................................. 243
8.7.4.39 fl_old_shortcut() ............................................ 244
8.7.4.40 fl_pie() ...................................................... 245
8.7.4.41 fl_polygon() [1/2] ........................................... 245
8.7.4.42 fl_polygon() [2/2] ........................................... 245
8.7.4.43 fl_pop_clip() .................................................. 245
8.7.4.44 fl_push_clip() ................................................ 246
8.7.4.45 fl_push_matrix() ............................................. 246
8.7.4.46 fl_read_image() .............................................. 246
8.7.4.47 fl_rect() ..................................................... 246
8.7.4.48 fl_rectf() .................................................... 247
8.7.4.49 fl_reset_spot() .............................................. 247
8.7.4.50 fl_rotate() .................................................. 247
8.7.4.51 fl_scale() [1/2] .............................................. 247
8.7.4.52 fl_scale() [2/2] .............................................. 247
8.7.4.53 fl_scroll() ................................................... 247
8.7.4.54 fl_set_spot() ................................................ 248
8.7.4.55 fl_set_status() .............................................. 248
8.7.4.56 fl_shortcut_label() [1/2] .................................. 248
8.7.4.57 fl_shortcut_label() [2/2] .................................. 249
8.7.4.58 fl_transform_dx() .......................................... 249
8.7.4.59 fl_transform_dy() .......................................... 250
8.7.4.60 fl_transform_x() ............................................ 250
8.7.4.61 fl_transform_y() ............................................ 250
8.7.4.62 fl_transformed_vertex() .................................. 250

Generated by Doxygen
9.2.1 Detailed Description ................................................. 299
9.2.2 Member Enumeration Documentation .............................. 299
  9.2.2.1 Fl_Option .................................................. 299
9.2.3 Member Function Documentation .................................. 300
  9.2.3.1 abi_check() ............................................... 300
  9.2.3.2 abi_version() ........................................... 300
  9.2.3.3 add_check() ............................................. 300
  9.2.3.4 add_fd() ................................................ 300
  9.2.3.5 add_idle() .............................................. 301
  9.2.3.6 add_timeout() .......................................... 301
  9.2.3.7 api_version() ........................................... 301
  9.2.3.8 arg() ..................................................... 301
  9.2.3.9 args()[1/2] ............................................. 302
  9.2.3.10 args()[2/2] ............................................ 302
  9.2.3.11 background() ......................................... 303
  9.2.3.12 background2() .................................... 303
  9.2.3.13 box_color() ........................................... 303
  9.2.3.14 box_dh() .............................................. 303
  9.2.3.15 box_dw() .............................................. 304
  9.2.3.16 box_dx() .............................................. 304
  9.2.3.17 box_dy() .............................................. 304
  9.2.3.18 check() ................................................ 304
  9.2.3.19 display() ................................................ 304
  9.2.3.20 dnd_text_ops()[1/2] .................................. 305
  9.2.3.21 dnd_text_ops()[2/2] .................................. 305
  9.2.3.22 draw_box_active() .................................... 305
  9.2.3.23 flush() ................................................ 305
  9.2.3.24 get_system_colors() ................................... 305
  9.2.3.25 gl_visual() .......................................... 305
  9.2.3.26 is_scheme() ......................................... 305
  9.2.3.27 option()[1/2] ......................................... 306
  9.2.3.28 option()[2/2] ......................................... 307
  9.2.3.29 own_colormap() ....................................... 307
  9.2.3.30 readqueue() ........................................... 307
  9.2.3.31 ready() ................................................ 307
  9.2.3.32 release() .............................................. 308
  9.2.3.33 reload_scheme() ...................................... 308
  9.2.3.34 remove_check() ....................................... 308
  9.2.3.35 remove_timeout() ..................................... 308
  9.2.3.36 repeat_timeout() ..................................... 308
  9.2.3.37 run() .................................................. 309
  9.2.3.38 scheme() .............................................. 309
9.2.3.39 scrollbar_size() .................................................. 309
9.2.3.40 scrollbar_size() .................................................. 309
9.2.3.41 set_box_color() .................................................. 309
9.2.3.42 set_idle() ......................................................... 310
9.2.3.43 use_high_res_GL() ................................................ 310
9.2.3.44 use_high_res_GL() ................................................ 310
9.2.3.45 version() ......................................................... 310
9.2.3.46 visible_focus() .................................................... 310
9.2.3.47 visible_focus() .................................................... 310
9.2.3.48 visual() ........................................................... 311
9.2.3.49 wait() ............................................................... 311
9.2.4 Member Data Documentation .......................................... 311
  9.2.4.1 help ............................................................... 311
  9.2.4.2 idle ............................................................... 311
9.3 Fl_Adjuster Class Reference ........................................... 312
  9.3.1 Detailed Description ................................................ 319
  9.3.2 Constructor & Destructor Documentation ................................ 320
    9.3.2.1 Fl_Adjuster() .................................................. 320
  9.3.3 Member Function Documentation ................................... 320
    9.3.3.1 draw() .......................................................... 320
    9.3.3.2 handle() ....................................................... 320
    9.3.3.3 soft() .......................................................... 321
    9.3.3.4 soft() .......................................................... 321
    9.3.3.5 value_damage() ................................................ 321
9.4 Fl_Bitmap Class Reference ............................................. 321
  9.4.1 Detailed Description ................................................ 323
  9.4.2 Member Function Documentation ................................... 323
    9.4.2.1 copy() .......................................................... 323
    9.4.2.2 draw() .......................................................... 324
    9.4.2.3 label() .......................................................... 324
    9.4.2.4 label() .......................................................... 324
    9.4.2.5 uncache() ....................................................... 324
9.5 Fl_BMP_Image Class Reference ......................................... 324
  9.5.1 Detailed Description ................................................ 327
  9.5.2 Constructor & Destructor Documentation ................................ 327
    9.5.2.1 Fl_BMP_Image() .................................................. 327
9.6 Fl_Box Class Reference .................................................. 327
  9.6.1 Detailed Description ................................................ 333
  9.6.2 Constructor & Destructor Documentation ................................ 333
    9.6.2.1 Fl_Box() ....................................................... 333
  9.6.3 Member Function Documentation ................................... 333
    9.6.3.1 draw() .......................................................... 333
9.7.3.37 lineposition() .................................................. 358
9.7.3.38 load() ................................................................. 359
9.7.3.39 make_visible() .................................................... 359
9.7.3.40 middleline() ....................................................... 359
9.7.3.41 move() ................................................................. 359
9.7.3.42 remove() .............................................................. 361
9.7.3.43 remove_icon() ..................................................... 361
9.7.3.44 select() ............................................................... 361
9.7.3.45 selected() ........................................................... 362
9.7.3.46 show() ................................................................. 362
9.7.3.47 show() ................................................................. 362
9.7.3.48 size() ................................................................. 362
9.7.3.49 swap() ................................................................. 362
9.7.3.50 swap() ................................................................. 362
9.7.3.51 text() ................................................................. 363
9.7.3.52 text() ................................................................. 363
9.7.3.53 textsize() ............................................................ 363
9.7.3.54 topline() ............................................................. 364
9.7.3.55 topline() ............................................................. 364
9.7.3.56 value() ............................................................... 364
9.7.3.57 value() ............................................................... 364
9.7.3.58 visible() ............................................................. 364

9.8 Fl_Browser_ Class Reference ........................................... 365
  9.8.1 Detailed Description .................................................. 375
    9.8.1.1 Keyboard navigation of browser items ..................... 375
  9.8.2 Member Enumeration Documentation ................................ 375
    9.8.2.1 anonymous enum ........................................... 375
  9.8.3 Constructor & Destructor Documentation .......................... 375
    9.8.3.1 Fl_Browser_() ................................................ 375
  9.8.4 Member Function Documentation .................................. 376
    9.8.4.1 bbox() ......................................................... 376
    9.8.4.2 deleting() .................................................... 376
    9.8.4.3 deselect() ................................................... 376
    9.8.4.4 display() ..................................................... 376
    9.8.4.5 displayed() .................................................. 377
    9.8.4.6 draw() ........................................................ 377
    9.8.4.7 find_item() .................................................. 377
    9.8.4.8 full_height() ............................................... 377
    9.8.4.9 full_width() ................................................ 378
    9.8.4.10 handle() ..................................................... 378
    9.8.4.11 has_scrollbar() ............................................ 378
    9.8.4.12 hposition() ............................................... 379
9.9.2.1 Fl_Button() ........................................... 394
9.9.3 Member Function Documentation ................................ 395
  9.9.3.1 clear() ............................................. 395
  9.9.3.2 down_box() [1/2] .................................... 395
  9.9.3.3 down_box() [2/2] .................................... 395
  9.9.3.4 draw() ............................................. 395
  9.9.3.5 handle() ........................................... 395
  9.9.3.6 set() .............................................. 396
  9.9.3.7 shortcut() [1/2] ..................................... 396
  9.9.3.8 shortcut() [2/2] ..................................... 396
  9.9.3.9 value() ............................................ 396
9.10 Fl_Cairo_State Class Reference .................................. 397
  9.10.1 Detailed Description ...................................... 397
  9.10.2 Member Function Documentation ............................. 398
    9.10.2.1 cc() .............................................. 398
9.11 Fl_Cairo_Window Class Reference ................................ 398
  9.11.1 Detailed Description ...................................... 409
  9.11.2 Member Function Documentation ............................. 409
    9.11.2.1 draw() ............................................. 409
    9.11.2.2 set_draw_cb() ...................................... 409
9.12 Fl_Chart Class Reference ......................................... 409
  9.12.1 Detailed Description ...................................... 416
  9.12.2 Constructor & Destructor Documentation ..................... 416
    9.12.2.1 Fl_Chart() ....................................... 416
  9.12.3 Member Function Documentation ................................ 417
    9.12.3.1 add() ............................................. 417
    9.12.3.2 autosize() [1/2] .................................... 417
    9.12.3.3 autosize() [2/2] .................................... 417
    9.12.3.4 bounds() [1/2] .................................... 417
    9.12.3.5 bounds() [2/2] .................................... 418
    9.12.3.6 draw() ............................................. 418
    9.12.3.7 insert() ........................................... 418
    9.12.3.8 maxsize() ......................................... 418
    9.12.3.9 replace() ......................................... 418
9.13 FL_CHART_ENTRY Struct Reference ................................ 419
  9.13.1 Detailed Description ...................................... 419
9.14 Fl_Check_Browser Class Reference ................................ 419
  9.14.1 Detailed Description ...................................... 429
  9.14.2 Member Function Documentation ................................ 430
    9.14.2.1 add() [1/2] ....................................... 430
    9.14.2.2 add() [2/2] ....................................... 430
    9.14.2.3 handle() .......................................... 430
9.23.2 Member Function Documentation ........................................... 491
  9.23.2.1 print() ............................................................... 491
  9.23.2.2 rectangle_capture() ................................................. 491

9.24 Fl_Dial Class Reference ....................................................... 492
  9.24.1 Detailed Description ..................................................... 499
  9.24.2 Constructor & Destructor Documentation ............................... 499
    9.24.2.1 Fl_Dial() .......................................................... 499

9.24.3 Member Function Documentation ......................................... 499
  9.24.3.1 angle1() .............................................................. 499
  9.24.3.2 draw() [1/2] ........................................................... 499
  9.24.3.3 draw() [2/2] ........................................................... 500
  9.24.3.4 handle() [1/2] ........................................................ 500
  9.24.3.5 handle() [2/2] ........................................................ 500

9.25 Fl_Display_Device Class Reference ....................................... 500
  9.25.1 Detailed Description .................................................... 501
  9.25.2 Member Function Documentation ...................................... 502
    9.25.2.1 class_name() ....................................................... 502

9.26 Fl_Double_Window Class Reference ...................................... 502
  9.26.1 Detailed Description .................................................... 512
  9.26.2 Constructor & Destructor Documentation .............................. 513
    9.26.2.1 ~Fl_Double_Window() .............................................. 513
  9.26.3 Member Function Documentation ...................................... 513
    9.26.3.1 flush() [1/2] ....................................................... 513
    9.26.3.2 flush() [2/2] ....................................................... 513
    9.26.3.3 hide() .................................................................. 513
    9.26.3.4 resize() ............................................................... 513
    9.26.3.5 show() ................................................................. 513

9.27 Fl_End Class Reference ....................................................... 514
  9.27.1 Detailed Description ..................................................... 514

9.28 Fl_File_Browser Class Reference ........................................ 514
  9.28.1 Detailed Description ..................................................... 526
  9.28.2 Constructor & Destructor Documentation .............................. 527
    9.28.2.1 Fl_File_Browser() .................................................. 527
  9.28.3 Member Function Documentation ...................................... 527
    9.28.3.1 filetype() [1/2] ...................................................... 527
    9.28.3.2 filetype() [2/2] ...................................................... 527
    9.28.3.3 filter() [1/2] ......................................................... 527
    9.28.3.4 filter() [2/2] ......................................................... 527
    9.28.3.5 iconsize() [1/2] ...................................................... 527
    9.28.3.6 iconsize() [2/2] ...................................................... 527
    9.28.3.7 load() ................................................................. 527

9.29 Fl_File_Chooser Class Reference ......................................... 528
9.42.2.5 descent()

9.42.2.6 draw() [1/5]

9.42.2.7 draw() [2/5]

9.42.2.8 draw() [3/5]

9.42.2.9 draw() [4/5]

9.42.2.10 draw() [5/5]

9.42.2.11 draw_image() [1/2]

9.42.2.12 draw_image() [2/2]

9.42.2.13 draw_image_mono() [1/2]

9.42.2.14 draw_image_mono() [2/2]

9.42.2.15 font()

9.42.2.16 height()

9.42.2.17 rtl_draw()

9.42.2.18 text_extents()

9.42.2.19 width() [1/2]

9.42.2.20 width() [2/2]

9.43 Fl_GDI_Printer_Graphics_Driver Class Reference

9.43.1 Detailed Description

9.43.2 Member Function Documentation

9.43.2.1 class_name()

9.43.2.2 draw() [1/2]

9.43.2.3 draw() [2/2]

9.43.2.4 draw_scaled()

9.44 Fl_GIF_Image Class Reference

9.44.1 Detailed Description

9.44.2 Constructor & Destructor Documentation

9.44.2.1 Fl_GIF_Image()

9.45 Fl_Gl_Choice Class Reference

9.46 Fl_Gl_Window Class Reference

9.46.1 Detailed Description

9.46.2 Constructor & Destructor Documentation

9.46.2.1 Fl_Gl_Window() [1/2]

9.46.2.2 Fl_Gl_Window() [2/2]

9.46.3 Member Function Documentation

9.46.3.1 as_gl_window()

9.46.3.2 can_do()

9.46.3.3 can_do_overlay()

9.46.3.4 context() [1/2]

9.46.3.5 context() [2/2]

9.46.3.6 context_valid()

9.46.3.7 draw()

9.46.3.8 flush()
9.53.2.54 width() [2/2] ................................................. 664
9.53.2.55 xyline() [1/3] ............................................. 664
9.53.2.56 xyline() [2/3] ............................................. 664
9.53.2.57 xyline() [3/3] ............................................. 664
9.53.2.58 yxline() [1/3] ............................................. 664
9.53.2.59 yxline() [2/3] ............................................. 664
9.53.2.60 yxline() [3/3] ............................................. 665

9.53.3 Friends And Related Symbol Documentation ................................................. 665
9.53.3.1 fl_arc [1/2] ................................................ 665
9.53.3.2 fl_arc [2/2] ................................................ 665
9.53.3.3 fl_begin_complex_polygon ........................................... 666
9.53.3.4 fl_begin_points ............................................. 666
9.53.3.5 fl_circle .................................................. 666
9.53.3.6 fl_clip_box ................................................. 666
9.53.3.7 fl_clip_region ............................................. 667
9.53.3.8 fl_color [1/2] ............................................. 667
9.53.3.9 fl_color [2/2] ............................................. 667
9.53.3.10 fl_copy_offscreen ........................................ 667
9.53.3.11 fl_curve .................................................. 668
9.53.3.12 fl_draw ................................................... 668
9.53.3.13 fl_draw_image [1/2] ...................................... 668
9.53.3.14 fl_draw_image [2/2] ...................................... 669
9.53.3.15 fl_draw_image_mono [1/2] .................................. 670
9.53.3.16 fl_draw_image_mono [2/2] .................................. 670
9.53.3.17 fl_font .................................................. 670
9.53.3.18 fl_gap ................................................... 670
9.53.3.19 fl_line_style ............................................. 671
9.53.3.20 fl_mult_matrix ........................................... 671
9.53.3.21 fl_not_clipped .......................................... 671
9.53.3.22 fl_pie .................................................. 672
9.53.3.23 fl_polygon [1/2] ........................................ 672
9.53.3.24 fl_polygon [2/2] ........................................ 672
9.53.3.25 fl_pop_clip .............................................. 672
9.53.3.26 fl_push_clip ............................................. 673
9.53.3.27 fl_push_matrix .......................................... 673
9.53.3.28 fl_rect .................................................. 673
9.53.3.29 fl_rotate ................................................ 673
9.53.3.30 fl_scale [1/2] ........................................... 673
9.53.3.31 fl_scale [2/2] ........................................... 673
9.53.3.32 fl_transform_dx .......................................... 674
9.53.3.33 fl_transform_dy .......................................... 674
9.53.3.34 fl_transform_x .......................................... 674
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.57.2.2 show()</td>
<td>692</td>
</tr>
<tr>
<td>9.57.2.3 textsize()</td>
<td>692</td>
</tr>
<tr>
<td>9.57.2.4 value()</td>
<td>692</td>
</tr>
<tr>
<td>9.57.2.5 value()</td>
<td>692</td>
</tr>
<tr>
<td>9.58 Fl_Help_Font_Sack Struct Reference</td>
<td>693</td>
</tr>
<tr>
<td>9.59 Fl_Help_Font_Style Struct Reference</td>
<td>693</td>
</tr>
<tr>
<td>9.59.1 Detailed Description</td>
<td>693</td>
</tr>
<tr>
<td>9.60 Fl_Help_Link Struct Reference</td>
<td>694</td>
</tr>
<tr>
<td>9.60.1 Detailed Description</td>
<td>694</td>
</tr>
<tr>
<td>9.61 Fl_Help_Target Struct Reference</td>
<td>694</td>
</tr>
<tr>
<td>9.61.1 Detailed Description</td>
<td>694</td>
</tr>
<tr>
<td>9.62 Fl_Help_View Class Reference</td>
<td>694</td>
</tr>
<tr>
<td>9.62.1 Detailed Description</td>
<td>703</td>
</tr>
<tr>
<td>9.62.2 Constructor &amp; Destructor Documentation</td>
<td>704</td>
</tr>
<tr>
<td>9.62.2.1 ~Fl_Help_View()</td>
<td>704</td>
</tr>
<tr>
<td>9.62.3 Member Function Documentation</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.1 draw()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.2 find()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.3 handle()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.4 leftline()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.5 link()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.6 load()</td>
<td>705</td>
</tr>
<tr>
<td>9.62.3.7 resize()</td>
<td>706</td>
</tr>
<tr>
<td>9.62.3.8 scrollbar_size()</td>
<td>706</td>
</tr>
<tr>
<td>9.62.3.9 scrollbar_size()</td>
<td>706</td>
</tr>
<tr>
<td>9.62.3.10 topline()</td>
<td>706</td>
</tr>
<tr>
<td>9.62.3.11 topline()</td>
<td>707</td>
</tr>
<tr>
<td>9.62.3.12 value()</td>
<td>707</td>
</tr>
<tr>
<td>9.63 Fl_Hold_Browser Class Reference</td>
<td>707</td>
</tr>
<tr>
<td>9.63.1 Detailed Description</td>
<td>719</td>
</tr>
<tr>
<td>9.63.2 Constructor &amp; Destructor Documentation</td>
<td>719</td>
</tr>
<tr>
<td>9.63.2.1 Fl_Hold_Browser()</td>
<td>719</td>
</tr>
<tr>
<td>9.64 Fl_Hor_Fill_Slider Class Reference</td>
<td>720</td>
</tr>
<tr>
<td>9.65 Fl_Hor_Nice_Slider Class Reference</td>
<td>727</td>
</tr>
<tr>
<td>9.66 Fl_Hor_Slider Class Reference</td>
<td>735</td>
</tr>
<tr>
<td>9.66.1 Detailed Description</td>
<td>742</td>
</tr>
<tr>
<td>9.67 Fl_Hor_Value_Slider Class Reference</td>
<td>742</td>
</tr>
<tr>
<td>9.68 Fl_Image Class Reference</td>
<td>750</td>
</tr>
<tr>
<td>9.68.1 Detailed Description</td>
<td>752</td>
</tr>
<tr>
<td>9.68.2 Constructor &amp; Destructor Documentation</td>
<td>752</td>
</tr>
<tr>
<td>9.68.2.1 Fl_Image()</td>
<td>752</td>
</tr>
<tr>
<td>9.68.3 Member Function Documentation</td>
<td>752</td>
</tr>
</tbody>
</table>
9.71.3.1 copy() ................................................................. 778
9.71.3.2 copy_cuts() .......................................................... 778
9.71.3.3 cursor_color() [1/2] ............................................. 778
9.71.3.4 cursor_color() [2/2] ............................................. 778
9.71.3.5 cut() [1/3] .............................................................. 779
9.71.3.6 cut() [2/3] .............................................................. 779
9.71.3.7 cut() [3/3] .............................................................. 779
9.71.3.8 drawtext() .............................................................. 779
9.71.3.9 handle_mouse() ....................................................... 780
9.71.3.10 handletext() ......................................................... 780
9.71.3.11 index() ................................................................. 780
9.71.3.12 input_type() [1/2] .................................................. 780
9.71.3.13 input_type() [2/2] .................................................. 781
9.71.3.14 insert() ............................................................... 781
9.71.3.15 line_end() ............................................................. 781
9.71.3.16 line_start() ........................................................... 781
9.71.3.17 mark() [1/2] .......................................................... 782
9.71.3.18 mark() [2/2] .......................................................... 782
9.71.3.19 maximum_size() [1/2] ............................................ 782
9.71.3.20 maximum_size() [2/2] ............................................ 782
9.71.3.21 position() [1/3] ..................................................... 782
9.71.3.22 position() [2/3] ..................................................... 783
9.71.3.23 position() [3/3] ..................................................... 783
9.71.3.24 readonly() [1/2] .................................................... 783
9.71.3.25 readonly() [2/2] .................................................... 784
9.71.3.26 replace() ............................................................. 784
9.71.3.27 resize() ............................................................... 785
9.71.3.28 shortcut() [1/2] .................................................... 785
9.71.3.29 shortcut() [2/2] .................................................... 785
9.71.3.30 size() [1/2] ........................................................... 785
9.71.3.31 size() [2/2] ........................................................... 786
9.71.3.32 static_value() [1/2] .............................................. 786
9.71.3.33 static_value() [2/2] .............................................. 786
9.71.3.34 tab_nav() [1/2] ..................................................... 787
9.71.3.35 tab_nav() [2/2] ..................................................... 787
9.71.3.36 textcolor() [1/2] ................................................... 787
9.71.3.37 textcolor() [2/2] ................................................... 787
9.71.3.38 textfont() [1/2] .................................................... 788
9.71.3.39 textfont() [2/2] .................................................... 788
9.71.3.40 textsize() [1/2] .................................................... 788
9.71.3.41 textsize() [2/2] .................................................... 788
9.71.3.42 undo() ................................................................. 788
9.82.19 labelcolor() [1/2] .......................................................... 877
9.82.20 labelcolor() [2/2] .......................................................... 877
9.82.21 labelfont() [1/2] .......................................................... 877
9.82.22 labelfont() [2/2] .......................................................... 877
9.82.23 labeltype() [1/2] .......................................................... 877
9.82.24 labeltype() [2/2] .......................................................... 877
9.82.25 measure() ................................................................. 878
9.82.26 next() [1/2] ................................................................. 878
9.82.27 next() [2/2] ................................................................. 878
9.82.28 popup() ................................................................. 878
9.82.29 pulldown() ............................................................... 878
9.82.30 radio() ................................................................. 879
9.82.31 set() ................................................................. 879
9.82.32 setonly() ............................................................... 879
9.82.33 shortcut() ............................................................. 879
9.82.34 size() ................................................................. 879
9.82.35 submenu() ............................................................. 879
9.82.36 test_shortcut() .......................................................... 880
9.82.37 uncheck() ............................................................. 880
9.82.38 value() ............................................................... 880

9.83 Fl_Menu_Window Class Reference ................................................. 880
9.83.1 Detailed Description ......................................................... 891
9.83.2 Member Function Documentation ........................................ 891
  9.83.2.1 clear_overlay() .................................................. 891
  9.83.2.2 flush() ............................................................. 891
  9.83.2.3 hide() .............................................................. 891
  9.83.2.4 set_overlay() .................................................... 891
  9.83.2.5 show() .............................................................. 891

9.84 Fl_Multi_Browser Class Reference ............................................ 892
9.84.1 Detailed Description ....................................................... 904
9.84.2 Constructor & Destructor Documentation ................................ 904
  9.84.2.1 Fl_Multi_Browser() ............................................... 904

9.85 Fl_Multi_Label Struct Reference ............................................. 904
9.85.1 Detailed Description ..................................................... 905
9.85.2 Member Data Documentation ............................................. 905
  9.85.2.1 labela ............................................................. 905
  9.85.2.2 labelb ............................................................. 905
  9.85.2.3 typea ............................................................. 905
  9.85.2.4 typeb ............................................................. 905

9.86 Fl_Multiline_Input Class Reference ........................................ 905
9.86.1 Detailed Description ..................................................... 914
9.86.2 Constructor & Destructor Documentation ................................ 914
9.115 Fl_Roller Class Reference ......................................................... 1100
  9.115.1 Detailed Description .................................................... 1107
  9.115.2 Constructor & Destructor Documentation ............................ 1107
    9.115.2.1 Fl_Roller() .......................................................... 1107
  9.115.3 Member Function Documentation ...................................... 1107
    9.115.3.1 draw() .................................................................. 1107
    9.115.3.2 handle() .................................................................. 1107
9.116 Fl_Round_Button Class Reference ......................................... 1108
  9.116.1 Detailed Description ..................................................... 1115
  9.116.2 Constructor & Destructor Documentation ........................... 1115
    9.116.2.1 Fl_Round_Button() .................................................... 1115
9.117 Fl_Round_Clock Class Reference ......................................... 1115
  9.117.1 Detailed Description ..................................................... 1122
9.118 Fl_Scroll Class Reference .................................................. 1122
  9.118.1 Detailed Description ..................................................... 1130
  9.118.2 Constructor & Destructor Documentation .......................... 1131
    9.118.2.1 Fl_Scroll() ............................................................. 1131
  9.118.3 Member Function Documentation .................................... 1131
    9.118.3.1 bbox() ................................................................ 1131
    9.118.3.2 draw() .................................................................. 1131
    9.118.3.3 handle() .................................................................. 1132
    9.118.3.4 recalc_scrobllbars() ................................................. 1132
    9.118.3.5 resize() ................................................................. 1132
    9.118.3.6 scroll_to() ............................................................. 1133
    9.118.3.7 scrollbar_size() [1/2] ............................................ 1133
    9.118.3.8 scrollbar_size() [2/2] ............................................ 1133
    9.118.3.9 xposition() ............................................................ 1134
    9.118.3.10 yposition() ........................................................... 1134
9.119 Fl_Scrollbar Class Reference ................................................. 1134
  9.119.1 Detailed Description ..................................................... 1142
  9.119.2 Constructor & Destructor Documentation .......................... 1142
    9.119.2.1 Fl_Scrollbar() .......................................................... 1142
  9.119.3 Member Function Documentation .................................... 1142
    9.119.3.1 draw() ................................................................. 1142
    9.119.3.2 handle() ............................................................... 1143
    9.119.3.3 linesize() ............................................................. 1143
    9.119.3.4 value() [1/3] ......................................................... 1143
    9.119.3.5 value() [2/3] ......................................................... 1143
    9.119.3.6 value() [3/3] ......................................................... 1144
9.120 Fl_Scroll::Fl_Scrollbar_Data Struct Reference ..................... 1144
  9.120.1 Detailed Description ..................................................... 1144
9.121 Fl_Secret_Input Class Reference ........................................... 1144
9.126.3 Member Function Documentation ............................................... 1201
  9.126.3.1 bounds() ................................................................. 1201
  9.126.3.2 draw() .................................................................. 1201
  9.126.3.3 handle() ................................................................. 1201
  9.126.3.4 scrollvalue() ......................................................... 1201
  9.126.3.5 slider_size() ......................................................... 1202
9.127 Fl_Spinner Class Reference ......................................................... 1202
  9.127.1 Detailed Description ...................................................... 1210
  9.127.2 Constructor & Destructor Documentation ................................ 1211
    9.127.2.1 Fl_Spinner() ....................................................... 1211
  9.127.3 Member Function Documentation ......................................... 1211
    9.127.3.1 handle() ............................................................. 1211
    9.127.3.2 maximum() ......................................................... 1211
    9.127.3.3 minimum() .......................................................... 1211
    9.127.3.4 resize() .............................................................. 1211
    9.127.3.5 step() ................................................................ 1212
    9.127.3.6 type() [1/2] ......................................................... 1212
    9.127.3.7 type() [2/2] ......................................................... 1212
    9.127.3.8 value() ............................................................... 1212
9.128 Fl_Surface_Device Class Reference .............................................. 1212
  9.128.1 Detailed Description ....................................................... 1213
  9.128.2 Member Function Documentation ......................................... 1214
    9.128.2.1 class_name() ....................................................... 1214
    9.128.2.2 set_current() ....................................................... 1214
    9.128.2.3 surface() ............................................................ 1214
9.129 Fl_Sys_Menu_Bar Class Reference ................................................. 1214
  9.129.1 Detailed Description ....................................................... 1223
  9.129.2 Constructor & Destructor Documentation ................................ 1224
    9.129.2.1 Fl_Sys_Menu_Bar() ................................................ 1224
  9.129.3 Member Function Documentation ......................................... 1224
    9.129.3.1 add() [1/3] ......................................................... 1224
    9.129.3.2 add() [2/3] ......................................................... 1224
    9.129.3.3 add() [3/3] ......................................................... 1224
    9.129.3.4 clear() ............................................................... 1225
    9.129.3.5 clear_submenu() .................................................. 1225
    9.129.3.6 draw() ............................................................... 1225
    9.129.3.7 insert() [1/2] ...................................................... 1225
    9.129.3.8 insert() [2/2] ...................................................... 1226
    9.129.3.9 menu() ................................................................ 1226
    9.129.3.10 mode() ............................................................. 1226
    9.129.3.11 remove() ........................................................... 1226
    9.129.3.12 replace() .......................................................... 1227
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.130 F1_System_Printer Class Reference</td>
<td>1227</td>
</tr>
<tr>
<td>9.130.1 Detailed Description</td>
<td>1229</td>
</tr>
<tr>
<td>9.130.2 Member Function Documentation</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.1 class_name()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.2 end_job()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.3 end_page()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.4 margins()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.5 origin()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.6 origin()</td>
<td>1230</td>
</tr>
<tr>
<td>9.130.2.7 printable_rect()</td>
<td>1231</td>
</tr>
<tr>
<td>9.130.2.8 rotate()</td>
<td>1231</td>
</tr>
<tr>
<td>9.130.2.9 scale()</td>
<td>1231</td>
</tr>
<tr>
<td>9.130.2.10 start_job()</td>
<td>1232</td>
</tr>
<tr>
<td>9.130.2.11 start_page()</td>
<td>1232</td>
</tr>
<tr>
<td>9.130.2.12 translate()</td>
<td>1232</td>
</tr>
<tr>
<td>9.130.2.13 untranslate()</td>
<td>1232</td>
</tr>
<tr>
<td>9.131 F1_Table Class Reference</td>
<td>1233</td>
</tr>
<tr>
<td>9.131.1 Detailed Description</td>
<td>1244</td>
</tr>
<tr>
<td>9.131.2 Member Enumeration Documentation</td>
<td>1246</td>
</tr>
<tr>
<td>9.131.2.1 TableContext</td>
<td>1246</td>
</tr>
<tr>
<td>9.131.3 Constructor &amp; Destructor Documentation</td>
<td>1246</td>
</tr>
<tr>
<td>9.131.3.1 F1_Table()</td>
<td>1246</td>
</tr>
<tr>
<td>9.131.3.2 ~F1_Table()</td>
<td>1247</td>
</tr>
<tr>
<td>9.131.4 Member Function Documentation</td>
<td>1247</td>
</tr>
<tr>
<td>9.131.4.1 callback()</td>
<td>1247</td>
</tr>
<tr>
<td>9.131.4.2 callback_col()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.3 callback_context()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.4 callback_row()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.5 child()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.6 children()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.7 clear()</td>
<td>1248</td>
</tr>
<tr>
<td>9.131.4.8 col_header()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.9 col_resize()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.10 col_resize_min()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.11 col_width()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.12 col_width_all()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.13 draw()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.14 draw_cell()</td>
<td>1249</td>
</tr>
<tr>
<td>9.131.4.15 get_selection()</td>
<td>1251</td>
</tr>
<tr>
<td>9.131.4.16 handle()</td>
<td>1251</td>
</tr>
<tr>
<td>9.131.4.17 is_interactive_resize()</td>
<td>1252</td>
</tr>
<tr>
<td>9.131.4.18 is_selected()</td>
<td>1252</td>
</tr>
</tbody>
</table>
9.131.4.19 resize() .................................................. 1252
9.131.4.20 row_header() ............................................ 1252
9.131.4.21 row_height() ............................................ 1252
9.131.4.22 row_height_all() ....................................... 1252
9.131.4.23 row_resize() ............................................ 1252
9.131.4.24 row_resize_min() ....................................... 1252
9.131.4.25 rows() .................................................... 1253
9.131.4.26 scrollbar_size() [1/2] ................................ 1253
9.131.4.27 scrollbar_size() [2/2] ................................ 1253
9.131.4.28 set_selection() ......................................... 1253
9.131.4.29 tab_cell_nav() [1/2] .................................. 1254
9.131.4.30 tab_cell_nav() [2/2] .................................. 1254
9.131.4.31 table_box() .............................................. 1254
9.131.4.32 top_row() [1/2] ........................................ 1254
9.131.4.33 top_row() [2/2] ........................................ 1254
9.131.4.34 visible_cells() .......................................... 1255
9.131.4.35 when() .................................................... 1255

9.132 Fl_Table_Row Class Reference .................................. 1255
9.132.1 Detailed Description ........................................ 1267
9.132.2 Constructor & Destructor Documentation ................ 1267
  9.132.2.1 Fl_Table_Row() ........................................ 1267
  9.132.2.2 ~Fl_Table_Row() ...................................... 1267
9.132.3 Member Function Documentation ........................... 1267
  9.132.3.1 clear() ................................................ 1267
  9.132.3.2 handle() .............................................. 1268
  9.132.3.3 row_selected() ....................................... 1268
  9.132.3.4 rows() ................................................. 1268
  9.132.3.5 select_all_rows() .................................... 1268
  9.132.3.6 select_row() ......................................... 1268
  9.132.3.7 type() ................................................ 1268

9.133 Fl_Tabs Class Reference ....................................... 1269
9.133.1 Detailed Description ....................................... 1276
9.133.2 Constructor & Destructor Documentation ................ 1279
  9.133.2.1 Fl_Tabs() ............................................. 1279
9.133.3 Member Function Documentation ........................... 1279
  9.133.3.1 client_area() ........................................ 1279
  9.133.3.2 draw() ............................................... 1280
  9.133.3.3 handle() ............................................. 1280
  9.133.3.4 push() [1/2] ......................................... 1280
  9.133.3.5 push() [2/2] ......................................... 1281
  9.133.3.6 value() [1/2] ......................................... 1281
  9.133.3.7 value() [2/2] ......................................... 1281
9.134.4 Member Data Documentation

9.134.4.1 file_encoding_warning_message
9.134.4.2 mTabDist
9.134.4.3 transcoding_warning_action

9.135 Fl_Text_Display Class Reference

9.135.1 Detailed Description

9.135.2 Member Enumeration Documentation

9.135.2.1 anonymous enum
9.135.2.2 anonymous enum

9.135.3 Constructor & Destructor Documentation

9.135.3.1 Fl_Text_Display()
9.135.3.2 ~Fl_Text_Display()

9.135.4 Member Function Documentation

9.135.4.1 absolute_top_line_number()
9.135.4.2 buffer() [1/3]
9.135.4.3 buffer() [2/3]
9.135.4.4 buffer() [3/3]
9.135.4.5 buffer_modified_cb()
9.135.4.6 buffer_predelete_cb()
9.135.4.7 calc_last_char()
9.135.4.8 calc_line_starts()
9.135.4.9 clear_rect()
9.135.4.10 col_to_x()
9.135.4.11 count_lines()
9.135.4.12 cursor_color() [1/2]
9.135.4.13 cursor_color() [2/2]
9.135.4.14 cursor_style()
9.135.4.15 display_insert()
9.135.4.16 draw() 
9.135.4.17 draw_cursor()
9.135.4.18 draw_line_numbers()
9.135.4.19 draw_range()
9.135.4.20 draw_string()
9.135.4.21 draw_text()
9.135.4.22 draw_vline()
9.135.4.23 empty_vlines()
9.135.4.24 extend_range_for_styles()
9.143 Fl_Tree Class Reference ........................................... 1394

9.143.1 Detailed Description ........................................... 1407

9.143.2 Member Function Documentation ............................... 1411

9.143.2.1 add() .................................................... 1411
9.143.2.2 add() .................................................... 1411
9.143.2.3 calc_dimensions() ....................................... 1413
9.143.2.4 calc_tree() .............................................. 1413
9.143.2.5 callback_item() .......................................... 1413
9.143.2.6 callback_item() .......................................... 1414
9.143.2.7 callback_reason() ...................................... 1414
9.143.2.8 callback_reason() ...................................... 1414
9.143.2.9 clear() .................................................. 1414
9.143.2.10 clear_children() ....................................... 1414
9.143.2.11 close() .................................................. 1414
9.143.2.12 close() .................................................. 1415
9.143.2.13 closeicon() ............................................. 1415
9.143.2.14 closeicon() ............................................. 1416
9.143.2.15 connectorstyle() ...................................... 1416
9.143.2.16 deselect() .............................................. 1416
9.143.2.17 deselect() .............................................. 1416
9.143.2.18 deselect_all() .......................................... 1417
9.143.2.19 display() ................................................ 1417
9.143.2.20 displayed() ............................................. 1417
9.143.2.21 draw() .................................................. 1418
9.143.2.22 extend_selection() ..................................... 1418
9.143.2.23 extend_selection_dir() ................................ 1418
9.143.2.24 find_clicked() ......................................... 1419
9.143.2.25 find_item() .............................................. 1419
9.143.2.26 first() ................................................... 1420
9.143.2.27 first_selected_item() .................................. 1420
9.143.2.28 first_visible() ......................................... 1420
9.143.2.29 first_visible_item() ................................... 1421
9.143.2.30 get_selected_items() .................................. 1421
9.143.2.31 handle() .................................................. 1421
9.143.2.32 hposition() [1/2] ........................................ 1421
9.143.2.33 hposition() [2/2] ........................................ 1422
9.143.2.34 insert() .................................................... 1422
9.143.2.35 insert_above() ............................................ 1423
9.143.2.36 is_close() [1/2] ........................................... 1423
9.143.2.37 is_close() [2/2] ........................................... 1423
9.143.2.38 is_hscroll_visible() ..................................... 1424
9.143.2.39 is_open() [1/2] ............................................ 1424
9.143.2.40 is_open() [2/2] ............................................ 1424
9.143.2.41 is_scrollbar() ............................................. 1425
9.143.2.42 is_selected() [1/2] ....................................... 1425
9.143.2.43 is_selected() [2/2] ....................................... 1425
9.143.2.44 is_vscroll_visible() ..................................... 1425
9.143.2.45 item_clicked() [1/2] ..................................... 1426
9.143.2.46 item_clicked() [2/2] ..................................... 1426
9.143.2.47 item_draw_mode() [1/3] ................................. 1426
9.143.2.48 item_draw_mode() [2/3] ................................. 1426
9.143.2.49 item_draw_mode() [3/3] ................................. 1426
9.143.2.50 item_labelbgcolor() [1/2] ............................ 1427
9.143.2.51 item_labelbgcolor() [2/2] ............................ 1427
9.143.2.52 item_labelfgcolor() .................................... 1427
9.143.2.53 item_labelfont() ........................................ 1427
9.143.2.54 item_labelsizer() ....................................... 1427
9.143.2.55 item_pathname() ........................................ 1427
9.143.2.56 item_reselect_mode() [1/2] ......................... 1428
9.143.2.57 item_reselect_mode() [2/2] ......................... 1428
9.143.2.58 last() .................................................... 1428
9.143.2.59 last_selected_item() ................................... 1428
9.143.2.60 last_visible() ........................................... 1429
9.143.2.61 last_visible_item() ..................................... 1429
9.143.2.62 load() .................................................... 1429
9.143.2.63 next() .................................................... 1429
9.143.2.64 next_item() ............................................... 1430
9.143.2.65 next_selected_item() .................................. 1430
9.143.2.66 next_visible_item() .................................... 1431
9.143.2.67 open() [1/2] ............................................. 1431
9.143.2.68 open() [2/2] ............................................. 1432
9.143.2.69 open_toggle() ........................................... 1433
9.143.2.70 openicon() [1/2] ........................................ 1433
9.143.2.71 openicon() [2/2] ........................................ 1433
9.143.2.72 prev() .................................................... 1433
9.143.2.73 recalc_tree() .................................................. 1434
9.143.2.74 remove() ....................................................... 1434
9.143.2.75 resize() .......................................................... 1434
9.143.2.76 root() ............................................................. 1434
9.143.2.77 root_label() ..................................................... 1435
9.143.2.78 scrollbar_size() [1/2] ........................................ 1435
9.143.2.79 scrollbar_size() [2/2] ........................................ 1435
9.143.2.80 select() [1/2] .................................................... 1435
9.143.2.81 select() [2/2] .................................................... 1436
9.143.2.82 select_all() ...................................................... 1436
9.143.2.83 select_only() .................................................... 1437
9.143.2.84 select_toggle() .................................................. 1437
9.143.2.85 selectbox() [1/2] ............................................... 1438
9.143.2.86 selectbox() [2/2] ............................................... 1438
9.143.2.87 selectmode() [1/2] ............................................. 1438
9.143.2.88 selectmode() [2/2] ............................................. 1438
9.143.2.89 set_item_focus() ................................................. 1438
9.143.2.90 show_item() [1/2] ............................................... 1438
9.143.2.91 show_item() [2/2] ............................................... 1439
9.143.2.92 show_item_bottom() ............................................ 1439
9.143.2.93 show_item_middle() ............................................ 1439
9.143.2.94 show_item_top() ............................................... 1439
9.143.2.95 show_self() ...................................................... 1440
9.143.2.96 showcollapse() [1/2] ......................................... 1440
9.143.2.97 showcollapse() [2/2] ......................................... 1440
9.143.2.98 showroot() ....................................................... 1440
9.143.2.99 sortorder() ........................................................ 1440
9.143.2.100 usericon() [1/2] ................................................. 1440
9.143.2.101 usericon() [2/2] ................................................. 1441
9.143.2.102 vposition() [1/2] .............................................. 1441
9.143.2.103 vposition() [2/2] .............................................. 1441

9.144 Fl_Tree_Item Class Reference ......................................... 1441

9.144.1 Detailed Description .............................................. 1446

9.144.2 Constructor & Destructor Documentation .......................... 1446
9.144.2.1 Fl_Tree_Item() [1/2] .......................................... 1446
9.144.2.2 Fl_Tree_Item() [2/2] .......................................... 1447

9.144.3 Member Function Documentation .................................... 1447
9.144.3.1 activate() ......................................................... 1447
9.144.3.2 add() [1/4] ....................................................... 1447
9.144.3.3 add() [2/4] ....................................................... 1447
9.144.3.4 add() [3/4] ....................................................... 1448
9.144.3.5 add() [4/4] ....................................................... 1448
9.144.3.48 prev_visible() ........................................................................ 1457
9.144.3.49 recalc_tree() ........................................................................ 1457
9.144.3.50 remove_child() [1/2] ................................................................. 1457
9.144.3.51 remove_child() [2/2] ................................................................. 1458
9.144.3.52 reparent() ............................................................................. 1458
9.144.3.53 replace() ............................................................................. 1458
9.144.3.54 replace_child() ................................................................... 1459
9.144.3.55 select() ............................................................................. 1459
9.144.3.56 select_all() ....................................................................... 1459
9.144.3.57 show_self() ....................................................................... 1459
9.144.3.58 show_widgets() ................................................................... 1459
9.144.3.59 swap_children() [1/2] ............................................................. 1460
9.144.3.60 swap_children() [2/2] ............................................................. 1460
9.144.3.61 tree() [1/2] ........................................................................ 1460
9.144.3.62 tree() [2/2] ........................................................................ 1460
9.144.3.63 update_prev_next() ................................................................. 1460
9.144.3.64 userdeicon() [1/2] ................................................................. 1461
9.144.3.65 userdeicon() [2/2] ................................................................. 1461
9.144.3.66 usericon() ........................................................................ 1461
9.144.3.67 visible_r() ........................................................................ 1461

9.145 Fl_Tree_Item_Array Class Reference .............................................. 1462
9.145.1 Detailed Description ..................................................................... 1462
9.145.2 Constructor & Destructor Documentation ..................................... 1463
  9.145.2.1 Fl_Tree_Item_Array() ............................................................... 1463
9.145.3 Member Function Documentation ................................................ 1463
  9.145.3.1 add() ............................................................................. 1463
  9.145.3.2 clear() ............................................................................. 1463
  9.145.3.3 deparent() ........................................................................ 1463
  9.145.3.4 insert() ........................................................................... 1463
  9.145.3.5 manage_item_destroy() ......................................................... 1463
  9.145.3.6 move() ........................................................................... 1464
  9.145.3.7 remove() [1/2] ................................................................ 1464
  9.145.3.8 remove() [2/2] ................................................................ 1464
  9.145.3.9 reparent() ................................................................... 1464
  9.145.3.10 replace() ................................................................... 1464

9.146 Fl_Tree_Prefs Class Reference ....................................................... 1465
9.146.1 Detailed Description ..................................................................... 1467
9.146.2 Member Function Documentation ................................................ 1467
  9.146.2.1 closedeicon() .................................................................. 1467
  9.146.2.2 closeicon() ..................................................................... 1468
  9.146.2.3 item_draw_mode() ............................................................... 1468
  9.146.2.4 item_labelbgcolor() [1/2] .................................................. 1468
9.146.2.5 item_labelbgcolor() ................................. 1468
9.146.2.6 marginbottom() ................................. 1468
9.146.2.7 opendeicon() ................................. 1468
9.146.2.8 openicon() ................................ 1468
9.146.2.9 openicon() ................................ 1468
9.146.2.10 selectmode() ................................. 1469
9.146.2.11 showcollapse() ................................. 1469
9.146.2.12 showroot() ................................. 1469
9.146.2.13 sortorder() ................................. 1469
9.146.2.14 userdeicon() ................................. 1469

9.147 Fl_Valuator Class Reference .......................... 1470
  9.147.1 Detailed Description .......................... 1477
  9.147.2 Constructor & Destructor Documentation ......... 1477
    9.147.2.1 Fl_Valuator() .......................... 1477
  9.147.3 Member Function Documentation ................. 1477
    9.147.3.1 format() ................................. 1477
    9.147.3.2 increment() ............................. 1478
    9.147.3.3 maximum() ................................ 1478
      9.147.3.4 maximum() ................................ 1478
    9.147.3.5 minimum() ................................ 1478
      9.147.3.6 minimum() ................................ 1478
    9.147.3.7 precision() ................................ 1478
    9.147.3.8 range() .................................... 1478
    9.147.3.9 round() ..................................... 1479
    9.147.3.10 step() ..................................... 1479
    9.147.3.11 value() .................................... 1479
      9.147.3.12 value() .................................... 1479
    9.147.3.13 value_damage() ............................ 1479

9.148 Fl_Value_Input Class Reference ....................... 1479
  9.148.1 Detailed Description .......................... 1487
  9.148.2 Constructor & Destructor Documentation .......... 1487
    9.148.2.1 Fl_Value_Input() .......................... 1487
  9.148.3 Member Function Documentation ................. 1487
    9.148.3.1 cursor_color() ................................ 1487
    9.148.3.2 cursor_color() ................................ 1487
    9.148.3.3 draw() ..................................... 1488
    9.148.3.4 handle() ..................................... 1488
    9.148.3.5 resize() ..................................... 1488
    9.148.3.6 shortcut() .................................... 1489
      9.148.3.7 shortcut() .................................... 1489
    9.148.3.8 soft() ...................................... 1489
    9.148.3.9 textcolor() .................................. 1489
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.151.4.46</td>
<td>draw_label() [2/3]</td>
</tr>
<tr>
<td>9.151.4.47</td>
<td>draw_label() [3/3]</td>
</tr>
<tr>
<td>9.151.4.48</td>
<td>h() [1/2]</td>
</tr>
<tr>
<td>9.151.4.49</td>
<td>h() [2/2]</td>
</tr>
<tr>
<td>9.151.4.50</td>
<td>handle()</td>
</tr>
<tr>
<td>9.151.4.51</td>
<td>hide()</td>
</tr>
<tr>
<td>9.151.4.52</td>
<td>image() [1/3]</td>
</tr>
<tr>
<td>9.151.4.53</td>
<td>image() [2/3]</td>
</tr>
<tr>
<td>9.151.4.54</td>
<td>image() [3/3]</td>
</tr>
<tr>
<td>9.151.4.55</td>
<td>inside()</td>
</tr>
<tr>
<td>9.151.4.56</td>
<td>is_label_copied()</td>
</tr>
<tr>
<td>9.151.4.57</td>
<td>label() [1/3]</td>
</tr>
<tr>
<td>9.151.4.58</td>
<td>label() [2/3]</td>
</tr>
<tr>
<td>9.151.4.59</td>
<td>label() [3/3]</td>
</tr>
<tr>
<td>9.151.4.60</td>
<td>label_shortcut()</td>
</tr>
<tr>
<td>9.151.4.61</td>
<td>labelcolor() [1/2]</td>
</tr>
<tr>
<td>9.151.4.62</td>
<td>labelcolor() [2/2]</td>
</tr>
<tr>
<td>9.151.4.63</td>
<td>labelfont() [1/2]</td>
</tr>
<tr>
<td>9.151.4.64</td>
<td>labelfont() [2/2]</td>
</tr>
<tr>
<td>9.151.4.65</td>
<td>labelsize() [1/2]</td>
</tr>
<tr>
<td>9.151.4.66</td>
<td>labelsize() [2/2]</td>
</tr>
<tr>
<td>9.151.4.67</td>
<td>labeltype() [1/2]</td>
</tr>
<tr>
<td>9.151.4.68</td>
<td>labeltype() [2/2]</td>
</tr>
<tr>
<td>9.151.4.69</td>
<td>measure_label()</td>
</tr>
<tr>
<td>9.151.4.70</td>
<td>output()</td>
</tr>
<tr>
<td>9.151.4.71</td>
<td>parent() [1/2]</td>
</tr>
<tr>
<td>9.151.4.72</td>
<td>parent() [2/2]</td>
</tr>
<tr>
<td>9.151.4.73</td>
<td>position()</td>
</tr>
<tr>
<td>9.151.4.74</td>
<td>redraw()</td>
</tr>
<tr>
<td>9.151.4.75</td>
<td>redraw_label()</td>
</tr>
<tr>
<td>9.151.4.76</td>
<td>resize()</td>
</tr>
<tr>
<td>9.151.4.77</td>
<td>selection_color() [1/2]</td>
</tr>
<tr>
<td>9.151.4.78</td>
<td>selection_color() [2/2]</td>
</tr>
<tr>
<td>9.151.4.79</td>
<td>set_active()</td>
</tr>
<tr>
<td>9.151.4.80</td>
<td>set_changed()</td>
</tr>
<tr>
<td>9.151.4.81</td>
<td>set_output()</td>
</tr>
<tr>
<td>9.151.4.82</td>
<td>set_visible()</td>
</tr>
<tr>
<td>9.151.4.83</td>
<td>set_visible_focus()</td>
</tr>
<tr>
<td>9.151.4.84</td>
<td>show()</td>
</tr>
<tr>
<td>9.151.4.85</td>
<td>size()</td>
</tr>
<tr>
<td>9.151.4.86</td>
<td>take_focus()</td>
</tr>
<tr>
<td>9.151.4.87</td>
<td>takesevents()</td>
</tr>
</tbody>
</table>

Generated by Doxygen
9.151.4.88 test_shortcut() [1/2] ......................................................... 1538
9.151.4.89 test_shortcut() [2/2] ......................................................... 1538
9.151.4.90 tooltip() [1/2] ................................................................. 1538
9.151.4.91 tooltip() [2/2] ................................................................. 1539
9.151.4.92 top_window() ................................................................. 1539
9.151.4.93 top_window_offset() ....................................................... 1539
9.151.4.94 type() [1/2] ................................................................. 1539
9.151.4.95 type() [2/2] ................................................................. 1540
9.151.4.96 user_data() [1/2] ............................................................ 1540
9.151.4.97 user_data() [2/2] ............................................................ 1540
9.151.4.98 visible() ...................................................................... 1540
9.151.4.99 visible_focus() [1/2] ........................................................ 1540
9.151.4.100 visible_focus() [2/2] ...................................................... 1541
9.151.4.101 visible_r() ................................................................. 1541
9.151.4.102 w() [1/2] .................................................................. 1541
9.151.4.103 w() [2/2] .................................................................. 1541
9.151.4.104 when() [1/2] ................................................................. 1541
9.151.4.105 when() [2/2] ................................................................. 1542
9.151.4.106 window() .................................................................. 1542
9.151.4.107 x() [1/2] .................................................................. 1542
9.151.4.108 x() [2/2] .................................................................. 1543
9.151.4.109 y() [1/2] .................................................................. 1543
9.151.4.110 y() [2/2] .................................................................. 1543

9.152 Fl_Widget_Tracker Class Reference ................................................. 1543
  9.152.1 Detailed Description ............................................................... 1544
  9.152.2 Member Function Documentation .............................................. 1544
    9.152.2.1 deleted() ............................................................... 1544
    9.152.2.2 exists() ............................................................... 1544
    9.152.2.3 widget() .............................................................. 1544

9.153 Fl_Window Class Reference ............................................................ 1544
  9.153.1 Detailed Description ............................................................... 1545
  9.153.2 Constructor & Destructor Documentation ..................................... 1555
    9.153.2.1 Fl_Window() [1/2] ...................................................... 1555
    9.153.2.2 Fl_Window() [2/2] ...................................................... 1555
    9.153.2.3 ~Fl_Window() ........................................................ 1556
  9.153.3 Member Function Documentation ............................................. 1556
    9.153.3.1 as_window() ......................................................... 1556
    9.153.3.2 border() .............................................................. 1556
    9.153.3.3 clear_border() ...................................................... 1556
    9.153.3.4 clear_modal_states() ............................................... 1556
    9.153.3.5 current() .............................................................. 1557
    9.153.3.6 cursor() [1/3] ......................................................... 1557
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.153.3.7 cursor()</td>
<td>1557</td>
</tr>
<tr>
<td>9.153.3.8 cursor()</td>
<td>1557</td>
</tr>
<tr>
<td>9.153.3.9 decorated_h()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.10 decorated_w()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.11 default_cursor()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.12 default_cursor()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.13 default_icon()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.14 default_icons()</td>
<td>1558</td>
</tr>
<tr>
<td>9.153.3.15 default_xclass()</td>
<td>1559</td>
</tr>
<tr>
<td>9.153.3.16 default_xclass()</td>
<td>1559</td>
</tr>
<tr>
<td>9.153.3.17 draw()</td>
<td>1559</td>
</tr>
<tr>
<td>9.153.3.18 flush()</td>
<td>1560</td>
</tr>
<tr>
<td>9.153.3.19 force_position()</td>
<td>1560</td>
</tr>
<tr>
<td>9.153.3.20 force_position()</td>
<td>1560</td>
</tr>
<tr>
<td>9.153.3.21 free_icons()</td>
<td>1560</td>
</tr>
<tr>
<td>9.153.3.22 free_position()</td>
<td>1560</td>
</tr>
<tr>
<td>9.153.3.23 fullscreen()</td>
<td>1561</td>
</tr>
<tr>
<td>9.153.3.24 fullscreen_screens()</td>
<td>1561</td>
</tr>
<tr>
<td>9.153.3.25 handle()</td>
<td>1561</td>
</tr>
<tr>
<td>9.153.3.26 hide()</td>
<td>1562</td>
</tr>
<tr>
<td>9.153.3.27 hotspot()</td>
<td>1562</td>
</tr>
<tr>
<td>9.153.3.28 icon()</td>
<td>1562</td>
</tr>
<tr>
<td>9.153.3.29 icon()</td>
<td>1562</td>
</tr>
<tr>
<td>9.153.3.30 icon()</td>
<td>1562</td>
</tr>
<tr>
<td>9.153.3.31 iconize()</td>
<td>1563</td>
</tr>
<tr>
<td>9.153.3.32 icons()</td>
<td>1563</td>
</tr>
<tr>
<td>9.153.3.33 make_current()</td>
<td>1563</td>
</tr>
<tr>
<td>9.153.3.34 modal()</td>
<td>1563</td>
</tr>
<tr>
<td>9.153.3.35 resize()</td>
<td>1563</td>
</tr>
<tr>
<td>9.153.3.36 set_menu_window()</td>
<td>1564</td>
</tr>
<tr>
<td>9.153.3.37 set_modal()</td>
<td>1564</td>
</tr>
<tr>
<td>9.153.3.38 set_non_modal()</td>
<td>1564</td>
</tr>
<tr>
<td>9.153.3.39 set_tooltip_window()</td>
<td>1564</td>
</tr>
<tr>
<td>9.153.3.40 shape()</td>
<td>1564</td>
</tr>
<tr>
<td>9.153.3.41 shape()</td>
<td>1565</td>
</tr>
<tr>
<td>9.153.3.42 show()</td>
<td>1565</td>
</tr>
<tr>
<td>9.153.3.43 show()</td>
<td>1566</td>
</tr>
<tr>
<td>9.153.3.44 shown()</td>
<td>1566</td>
</tr>
<tr>
<td>9.153.3.45 size_range()</td>
<td>1566</td>
</tr>
<tr>
<td>9.153.3.46 wait_for_expose()</td>
<td>1567</td>
</tr>
<tr>
<td>9.153.3.47 xclass()</td>
<td>1567</td>
</tr>
<tr>
<td>9.153.3.48 xclass()</td>
<td>1568</td>
</tr>
</tbody>
</table>
9.153.4 Member Data Documentation ................................................. 1568
  9.153.4.1 current_ .............................................................. 1568

9.154 Fl_Wizard Class Reference .................................................. 1568
  9.154.1 Detailed Description .................................................... 1576
  9.154.2 Constructor & Destructor Documentation ............................... 1576
    9.154.2.1 Fl_Wizard() .......................................................... 1576
  9.154.3 Member Function Documentation ....................................... 1576
    9.154.3.1 next() ................................................................. 1576

9.155 Fl_XBM_Image Class Reference ............................................ 1576
  9.155.1 Detailed Description .................................................... 1578
  9.155.2 Constructor & Destructor Documentation ............................... 1578
    9.155.2.1 Fl_XBM_Image() ...................................................... 1578

9.156 Fl_XColor Struct Reference ................................................ 1579
  9.156.1 Detailed Description .................................................... 1583
  9.156.2 Member Function Documentation ....................................... 1583
    9.156.2.1 class_name() ........................................................ 1583
    9.156.2.2 color() [1/2] ......................................................... 1583
    9.156.2.3 color() [2/2] ......................................................... 1583
    9.156.2.4 copy_offscreen() ................................................... 1583
    9.156.2.5 descent() ............................................................ 1584
    9.156.2.6 draw() [1/5] .......................................................... 1584
    9.156.2.7 draw() [2/5] .......................................................... 1584
    9.156.2.8 draw() [3/5] .......................................................... 1584
    9.156.2.9 draw() [4/5] .......................................................... 1584
    9.156.2.10 draw() [5/5] ......................................................... 1585
    9.156.2.11 draw_image() [1/2] ................................................. 1585
    9.156.2.12 draw_image() [2/2] ................................................. 1585
    9.156.2.13 draw_image_mono() [1/2] ......................................... 1585
    9.156.2.14 draw_image_mono() [2/2] ......................................... 1585
    9.156.2.15 font() ............................................................... 1586
    9.156.2.16 height() ............................................................. 1586
    9.156.2.17 rtl_draw() ......................................................... 1586
    9.156.2.18 text_extents() ..................................................... 1586
    9.156.2.19 width() [1/2] ....................................................... 1586
    9.156.2.20 width() [2/2] ....................................................... 1586

9.157 Fl_XPM_Image Class Reference ............................................ 1587
  9.157.1 Detailed Description .................................................... 1589
  9.157.2 Member Function Documentation ....................................... 1589
    9.157.2.1 class_name() ........................................................ 1589
    9.157.2.2 color() [1/2] ......................................................... 1589
    9.157.2.3 color() [2/2] ......................................................... 1589
    9.157.2.4 copy_offscreen() ................................................... 1589
    9.157.2.5 descent() ............................................................ 1589
    9.157.2.6 draw() [1/5] .......................................................... 1589
    9.157.2.7 draw() [2/5] .......................................................... 1589
    9.157.2.8 draw() [3/5] .......................................................... 1589
    9.157.2.9 draw() [4/5] .......................................................... 1589
    9.157.2.10 draw() [5/5] ......................................................... 1589
    9.157.2.11 draw_image() [1/2] ................................................. 1589
    9.157.2.12 draw_image() [2/2] ................................................. 1589
    9.157.2.13 draw_image_mono() [1/2] ......................................... 1589
    9.157.2.15 font() ............................................................... 1590
    9.157.2.16 height() ............................................................. 1590
    9.157.2.17 rtl_draw() ......................................................... 1590
    9.157.2.18 text_extents() ..................................................... 1590
    9.157.2.19 width() [1/2] ....................................................... 1590
    9.157.2.20 width() [2/2] ....................................................... 1590

9.158 Fl_XPM_Image Class Reference ............................................ 1590
  9.158.1 Detailed Description .................................................... 1590
  9.158.2 Constructor & Destructor Documentation ............................... 1590
    9.158.2.1 Fl_XPM_Image() ...................................................... 1590

9.159 Fl_Text_Editor::Key_Binding Struct Reference ........................... 1590
  9.159.1 Detailed Description .................................................... 1590
9.160 Fl_Graphics_Driver::matrix Struct Reference .................................................. 1590
  9.160.1 Detailed Description ....................................................................................... 1590
9.161 Fl_Preferences::Name Class Reference ............................................................... 1590
  9.161.1 Detailed Description ....................................................................................... 1590
  9.161.2 Constructor & Destructor Documentation ....................................................... 1591
    9.161.2.1 Name() [1/2] ..................................................................................... 1591
    9.161.2.2 Name() [2/2] ..................................................................................... 1591
9.162 Fl_Preferences::Node Class Reference ............................................................... 1591
9.163 Fl_Paged_Device::page_format Struct Reference .................................................. 1592
  9.163.1 Detailed Description ....................................................................................... 1592
9.164 Fl_Preferences::RootNode Class Reference .......................................................... 1592
9.165 Fl_Scroll::ScrollInfo Struct Reference .............................................................. 1592
  9.165.1 Detailed Description ....................................................................................... 1593
9.166 Fl_Window::shape_data_type Struct Reference ..................................................... 1593
  9.166.1 Detailed Description ....................................................................................... 1593
9.167 Fl_Text_Display::Style_Table_Entry Struct Reference ........................................ 1594
  9.167.1 Detailed Description ....................................................................................... 1594

10 File Documentation .................................................................................................. 1595
  10.1 abi-version.h ...................................................................................................... 1595
  10.2 dirent.h .............................................................................................................. 1595
  10.3 Enumerations.H File Reference ............................................................................ 1595
    10.3.1 Detailed Description ....................................................................................... 1605
    10.3.2 Macro Definition Documentation .................................................................. 1605
      10.3.2.1 FL_ABI_VERSION ............................................................................. 1605
      10.3.2.2 FL_API_VERSION ........................................................................... 1606
      10.3.2.3 FL_MAJOR_VERSION ....................................................................... 1606
      10.3.2.4 FL_MINOR_VERSION ....................................................................... 1606
      10.3.2.5 FL_PATCH_VERSION ........................................................................ 1606
      10.3.2.6 FL_VERSION .................................................................................... 1606
    10.3.3 Typedef Documentation .................................................................................. 1607
      10.3.3.1 Fl_Fontsize ....................................................................................... 1607
    10.3.4 Enumeration Type Documentation .................................................................. 1607
      10.3.4.1 anonymous enum .............................................................................. 1607
      10.3.4.2 Fl_Boxtype ....................................................................................... 1607
      10.3.4.3 Fl_Cursor ......................................................................................... 1608
      10.3.4.4 Fl_Damage ....................................................................................... 1609
      10.3.4.5 Fl_Event ......................................................................................... 1609
      10.3.4.6 Fl_Labeltype .................................................................................... 1612
      10.3.4.7 Fl_When ......................................................................................... 1613
    10.3.5 Function Documentation ............................................................................... 1613
      10.3.5.1 fl_box() .......................................................................................... 1613
10.29.2.1 Fl_Draw_Image_Cb .................................................. 1653
10.30 Fl_Device.H .............................................................. 1653
10.31 Fl_Dial.H ............................................................... 1658
10.32 Fl_Double_Window.H .................................................... 1659
10.33 fl_draw.H File Reference ................................................ 1659
   10.33.1 Detailed Description .............................................. 1664
10.34 fl_draw.H ............................................................... 1664
10.35 Fl_Export.H ............................................................. 1667
10.36 Fl_File_Browser.H ..................................................... 1668
10.37 Fl_File_Chooser.H ...................................................... 1669
10.38 Fl_File_ICON.H ........................................................ 1671
10.39 Fl_File_Input.H ........................................................ 1673
10.40 Fl_Fill_Dial.H .......................................................... 1673
10.41 Fl_Fill_Slider.H ........................................................ 1674
10.42 Fl_Float_Input.H ........................................................ 1674
10.43 Fl_FormsBitmap.H ....................................................... 1675
10.44 Fl_FormsPixmap.H ....................................................... 1675
10.45 Fl_Free.H ............................................................... 1676
10.46 Fl_GIF_IMAGE.H ........................................................ 1677
10.47 Fl_Gl_Window.H ........................................................ 1677
10.48 Fl_Group.H ............................................................. 1679
10.49 Fl_Help_Dialog.H ....................................................... 1680
10.50 Fl_Help_View.H ........................................................ 1681
10.51 Fl_Hold_Browser.H .................................................... 1684
10.52 Fl_Hor_Fill_Slider.H .................................................. 1684
10.53 Fl_Hor_Nice_Slider.H .................................................. 1685
10.54 Fl_Hor_Slider.H ......................................................... 1685
10.55 Fl_Hor_Value_Slider.H ................................................ 1686
10.56 Fl_Image.H File Reference ............................................. 1686
   10.56.1 Detailed Description .............................................. 1687
   10.56.2 Enumeration Type Documentation ................................ 1687
   10.56.2.1 Fl_RGB_Scaling .................................................. 1687
10.57 Fl_Image.H ............................................................. 1687
10.58 Fl_Image_Surface.H ................................................... 1689
10.59 Fl_Input.H ............................................................. 1690
10.60 Fl_Input_.H ............................................................. 1691
10.61 Fl_Input_Choice.H ..................................................... 1694
10.62 Fl_Int_Input.H ........................................................ 1695
10.63 Fl_JPEG_IMAGE.H ...................................................... 1696
10.64 Fl_Light_Button.H ..................................................... 1696
10.65 Fl_Line_Dial.H ........................................................ 1697
10.66 Fl_Menu.H .............................................................. 1697
<table>
<thead>
<tr>
<th>Section</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.137</td>
<td>FI_Tree_Item.H</td>
<td>1770</td>
</tr>
<tr>
<td>10.138</td>
<td>FI_Tree_Item_Array.H</td>
<td>File Reference 1774</td>
</tr>
<tr>
<td>10.138.1</td>
<td>Detailed Description 1774</td>
<td></td>
</tr>
<tr>
<td>10.139</td>
<td>FI_Tree_Item_Array.H</td>
<td>1774</td>
</tr>
<tr>
<td>10.140</td>
<td>FI_Tree_Prefs.H</td>
<td>File Reference 1776</td>
</tr>
<tr>
<td>10.140.1</td>
<td>Detailed Description 1776</td>
<td></td>
</tr>
<tr>
<td>10.140.2</td>
<td>Enumeration Type Documentation 1776</td>
<td></td>
</tr>
<tr>
<td>10.140.2.1</td>
<td>FL_Tree_Connector 1776</td>
<td></td>
</tr>
<tr>
<td>10.140.2.2</td>
<td>FI_Tree_Item_Draw_Mode 1777</td>
<td></td>
</tr>
<tr>
<td>10.140.2.3</td>
<td>FI_Tree_Item_Reselect_Mode 1777</td>
<td></td>
</tr>
<tr>
<td>10.140.2.4</td>
<td>FI_Tree_Select 1777</td>
<td></td>
</tr>
<tr>
<td>10.140.2.5</td>
<td>FI_Tree_Sort 1777</td>
<td></td>
</tr>
<tr>
<td>10.141</td>
<td>FI_Tree_Prefs.H</td>
<td>1777</td>
</tr>
<tr>
<td>10.142</td>
<td>fi_types.h</td>
<td>File Reference 1781</td>
</tr>
<tr>
<td>10.142.1</td>
<td>Detailed Description 1782</td>
<td></td>
</tr>
<tr>
<td>10.142.2</td>
<td>Typedef Documentation 1782</td>
<td></td>
</tr>
<tr>
<td>10.142.2.1</td>
<td>FL_CString 1782</td>
<td></td>
</tr>
<tr>
<td>10.142.2.2</td>
<td>FL_String 1782</td>
<td></td>
</tr>
<tr>
<td>10.143</td>
<td>fl_types.h</td>
<td>1782</td>
</tr>
<tr>
<td>10.144</td>
<td>fl_utf8.h</td>
<td>File Reference 1783</td>
</tr>
<tr>
<td>10.144.1</td>
<td>Detailed Description 1784</td>
<td></td>
</tr>
<tr>
<td>10.145</td>
<td>fl_utf8.h</td>
<td>1784</td>
</tr>
<tr>
<td>10.146</td>
<td>FI_Valuator.H</td>
<td>1787</td>
</tr>
<tr>
<td>10.147</td>
<td>FI_Value_Input.H</td>
<td>1788</td>
</tr>
<tr>
<td>10.148</td>
<td>FI_Value_Output.H</td>
<td>1789</td>
</tr>
<tr>
<td>10.149</td>
<td>FI_Value_Slider.H</td>
<td>1790</td>
</tr>
<tr>
<td>10.150</td>
<td>FI_Widget.H</td>
<td>File Reference 1790</td>
</tr>
<tr>
<td>10.150.1</td>
<td>Detailed Description 1791</td>
<td></td>
</tr>
<tr>
<td>10.150.2</td>
<td>Macro Definition Documentation 1791</td>
<td></td>
</tr>
<tr>
<td>10.150.2.1</td>
<td>FL_RESERVED_TYPE 1791</td>
<td></td>
</tr>
<tr>
<td>10.150.3</td>
<td>Typedef Documentation 1791</td>
<td></td>
</tr>
<tr>
<td>10.150.3.1</td>
<td>fl_intptr_t 1791</td>
<td></td>
</tr>
<tr>
<td>10.151</td>
<td>FI_Widget.H</td>
<td>1791</td>
</tr>
<tr>
<td>10.152</td>
<td>FI_Window.H</td>
<td>File Reference 1795</td>
</tr>
<tr>
<td>10.152.1</td>
<td>Detailed Description 1796</td>
<td></td>
</tr>
<tr>
<td>10.153</td>
<td>FI_Window.H</td>
<td>1796</td>
</tr>
<tr>
<td>10.154</td>
<td>FI_Wizard.H</td>
<td>1799</td>
</tr>
<tr>
<td>10.155</td>
<td>FI_XBM_Image.H</td>
<td>1800</td>
</tr>
<tr>
<td>10.156</td>
<td>FI_XPM_Image.H</td>
<td>1800</td>
</tr>
<tr>
<td>10.157</td>
<td>forms.H</td>
<td>1801</td>
</tr>
<tr>
<td>10.158</td>
<td>gl.h</td>
<td>File Reference 1810</td>
</tr>
<tr>
<td>10.158.1</td>
<td>Detailed Description 1811</td>
<td></td>
</tr>
</tbody>
</table>
10.158.2 Function Documentation

10.158.2.1 gl_color() ........................................ 1811
10.158.2.2 gl_draw() ........................................ 1812
10.158.2.3 gl_draw() ........................................ 1812
10.158.2.4 gl_draw() ........................................ 1812
10.158.2.5 gl_draw() ........................................ 1812
10.158.2.6 gl_draw() ........................................ 1812
10.158.2.7 gl_draw() ........................................ 1813
10.158.2.8 gl_draw() ........................................ 1813
10.158.2.9 gl_rect() ........................................ 1813
10.158.2.10 gl_rectf() ...................................... 1813

10.159 gl.h ..................................................... 1813
10.160 gl2opengl.h ........................................... 1814
10.161 gl_draw.H .............................................. 1815
10.162 glu.h .................................................. 1815
10.163 glut.H .................................................. 1816
10.164 mac.H File Reference ................................ 1821
  10.164.1 Detailed Description ................................ 1822
10.165 mac.H .................................................. 1822
10.166 math.h ................................................. 1826
10.167 names.h ............................................... 1826
10.168 platform.H ............................................ 1827
10.169 win32.H ............................................... 1828
10.170 x.H ..................................................... 1830
10.171 cgdebug.h ............................................. 1832
10.172 fastarrow.h .......................................... 1834
10.173 fl_arc.cxx File Reference ............................. 1835
  10.173.1 Detailed Description ................................ 1835
10.174 fl_arci.cxx File Reference ............................ 1835
  10.174.1 Detailed Description ................................ 1835
10.175 fl_ask.cxx File Reference ............................. 1835
  10.175.1 Detailed Description ................................ 1836
10.176 fl_boxtype.cxx File Reference ........................ 1836
  10.176.1 Detailed Description ................................ 1837
10.177 fl_cmap.h .............................................. 1839
10.178 fl_color.cxx File Reference ........................... 1842
  10.178.1 Detailed Description ................................ 1843
10.179 fl_compose.cxx File Reference ........................ 1843
  10.179.1 Detailed Description ................................ 1843
| 10.216 | gb2312.h | 2076 |
| 10.217 | georgian_academy.h | 2106 |
| 10.218 | georgian_ps.h | 2107 |
| 10.219 | iso8859_1.h | 2108 |
| 10.220 | iso8859_10.h | 2108 |
| 10.221 | iso8859_11.h | 2110 |
| 10.222 | iso8859_13.h | 2111 |
| 10.223 | iso8859_14.h | 2112 |
| 10.224 | iso8859_15.h | 2113 |
| 10.225 | iso8859_16.h | 2114 |
| 10.226 | iso8859_2.h | 2115 |
| 10.227 | iso8859_3.h | 2116 |
| 10.228 | iso8859_4.h | 2118 |
| 10.229 | iso8859_5.h | 2119 |
| 10.230 | iso8859_6.h | 2120 |
| 10.231 | iso8859_7.h | 2121 |
| 10.232 | iso8859_8.h | 2122 |
| 10.233 | iso8859_9.h | 2123 |
| 10.234 | iso8859_9e.h | 2124 |
| 10.235 | jisx0201.h | 2125 |
| 10.236 | jisx0208.h | 2126 |
| 10.237 | jisx0212.h | 2153 |
| 10.238 | koi8_c.h | 2178 |
| 10.239 | koi8_r.h | 2180 |
| 10.240 | koi8_u.h | 2181 |
| 10.241 | ksc5601.h | 2183 |
| 10.242 | mulelao.h | 2218 |
| 10.243 | tatar_cyr.h | 2219 |
| 10.244 | tcvn.h | 2220 |
| 10.245 | tis620.h | 2222 |
| 10.246 | ucs2be.h | 2223 |
| 10.247 | utf8.h | 2223 |
| 10.248 | viscii.h | 2224 |
| 10.249 | mk_wcwidth.c | 2226 |
| 10.250 | ucs2fontmap.c | 2230 |
| 10.251 | utf8Utils.c | 2234 |
| 10.252 | Ximint.h | 2237 |
| 10.253 | Xlibint.h | 2237 |
1.1 Preface

This manual describes the Fast Light Tool Kit ("FLTK") version 1.3.9, a C++ Graphical User Interface ("GUI") toolkit for UNIX, Microsoft Windows and Apple OS X.

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
- Designing a Simple Text Editor
- Drawing Things in FLTK
- Handling Events

- Advanced FLTK
  Unicode and UTF-8 Support

Appendices:

- FLTK Enumerations
- GLUT Compatibility
  – class Fl_Glut_Window
- Forms Compatibility
- Operating System Issues
- Migrating Code from FLTK 1.0 to 1.1
- Migrating Code from FLTK 1.1 to 1.3
- Developer Information
- Software License
- Example Source Code
- FAQ (Frequently Asked Questions)
- Adding and Extending Widgets
- Using OpenGL
- Programming with FLUID
- Advanced FLTK
- Unicode and UTF-8 Support
- FLTK Enumerations
- GLUT Compatibility
- Forms Compatibility
- Operating System Issues
- Migrating Code from FLTK 1.0 to 1.1
- Migrating Code from FLTK 1.1 to 1.3
- 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, and Windows 7. 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 1998-2023 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), Microsoft® Windows®, and Apple® macOS®.

FLTK provides modern GUI functionality without 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 on GitHub.

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 Introduction to FLTK

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, WIN32 or Cocoa) for maximum speed, and carefully optimized for code size and performance.

- Precise low-level compatibility between the X11, WIN32 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, WIN32, or QuickDraw.

- OpenGL/Mesa drawing area widget.

- Support for OpenGL overlay hardware on both X11 and WIN32, 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 Under UNIX and Apple OS X

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

For OS X, Xcode 3 project files can be found in the 'ide' directory.

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 WIN32

- **--enable-debug**
  
  Enable debugging code & symbols

- **--disable-gl**
  
  Disable OpenGL support

- **--enable-shared**
  
  Enable generation of shared libraries
-enable-threads

Enable multithreading support

-enable-xdbe

Enable the X double-buffer extension

-enable-xt

Enable the Xft library for anti-aliased fonts under X11

-enable-x11

When targeting cygwin, build with X11 GUI instead of windows GDI

-enable-cp936

Under X11, enable use of the GB2312 locale

-bindir=/path

Set the location for executables [default = $prefix/bin]

-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".
1.2.6  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.MSWindows.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 the supplied workspace and project files. 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!

The project files can be found in the "ide/" directory. Please read "ide/README.IDE" for more info about this.

1.2.6.1  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 OS X build and the notes above in the section on Building and Installing FLTK Under UNIX and Apple OS X 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:

```
make 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.
1.2.6.2 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.

1.2.7 Internet Resources

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

WWW

http://www.fltk.org/
http://www.fltk.org/str.php [for reporting bugs]
https://www.fltk.org/software.php [source code]
http://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]
http://fltk.org/newsgroups.php [web interface]

1.2.8 Reporting Bugs

To report a bug in FLTK, or for feature requests, please use the form at http://www.fltk.org/str.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 http://fltk.org/newsgroups.php.

1.3 FLTK Basics

This chapter teaches you the basics of compiling programs that use FLTK.
1.3.1 Writing Your First FLTK Program

All programs must include the file `<FL/Fl.H>`. 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 window in Figure 4.1. You can quit the program by closing the window or pressing the ESCape key.

![Hello, World!](image)

**Figure 1.1 The Hello, World! Window**
1.3 FLTK Basics

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:

\[ \text{Fl_Widget(x, y, width, height, label)} \]

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`) and the units are in pixels.

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

`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.

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(...)`.

1.3.1.3 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 in 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 set/get 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`. You can also specify typefaces directly.

The `labelsize()` method sets the height of the font in pixels.

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  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:

```bash
CC  -I/usr/local/include ... 
gcc -I/usr/local/include ... 
```

The `fltk-config` script included with FLTK can be used to get the options that are required by your compiler:

```bash
CC `fltk-config --cxxflags` ... 
```

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

```bash
CC  ... `fltk-config --ldflags` 
gcc  ... `fltk-config --ldflags` 
```

Aside from the "fltk" library, there is also a "fltk_forms" library for the XForms compatibility classes, "fltk_gl" for the OpenGL and GLUT classes, and "fltk_images" for the image file classes, `Fl_Help_Dialog` widget, and system icon support.

The libraries are named "fltk.lib", "fltkgl.lib", "fltkforms.lib", and "fltkimages.lib", respectively under Windows.

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

```bash
CC  ... 'fltk-config --ldflags'
```

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

```bash
CC  ... 'fltk-config --use-foo --ldflags'
```

Finally, you can use the `fltk-config` script to compile a single source file as a FLTK program:

```bash
fltk-config --compile filename.cpp 
fltk-config --use-foo --compile filename.cpp 
fltk-config --use-foo --compile filename.cpp 
fltk-config --use-foo --compile filename.cpp 
```

Any of these will create an executable named `filename`.
1.3 FLTK Basics

1.3.3 Compiling Programs with Makefiles

The previous section described how to use fltk-config to build a program consisting of a single source file 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 programs out of multiple source files:

```bash
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

%.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 CubeView.h CubeViewUI.h
 CubeViewUI.o: CubeViewUI.cxx CubeView.h

clean: $(TARGET) $(OBJS)
 rm -f *.o 2> /dev/null
 rm -f $(TARGET) 2> /dev/null
```

1.3.4 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 FLTK (FLTK.LIB or FLTKD.LIB) and the Windows Common Controls (COMCTL32.LIB) libraries to the "Link" settings. You must also define WIN32.

More information can be found in README.MSWindows.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.

1.3.5 Naming

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/...>. 

Generated by Doxygen
1.3.6  Header Files

The proper way to include FLTK header files is:

```c
#include <FL/Fl_xy.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:

```c
#include <FL\Fl_xy.H>
#include <fl/fl_xy.h>
#include <Fl/fl_xy.h>
```

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_xy_Button.H>` header file. The constructor takes the bounding box of the button and optionally a label string:

```c
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:

```c
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:

```c
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.
Figure 1.3 FLTK valuators widgets

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 \texttt{x()}, \texttt{y()}, \texttt{w()}, and \texttt{h()} methods.

You can change the size and position by using the \texttt{position()}, \texttt{resize()}, and \texttt{size()} methods:

\begin{verbatim}
button->position(x, y);
group->resize(x, y, width, height);
window->size(width, height);
\end{verbatim}

If you change a widget's size or position after it is displayed you will have to call \texttt{redraw()} on the widget's parent.

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:

- \texttt{FL\_BLACK}
- \texttt{FL\_RED}
- \texttt{FL\_GREEN}
- \texttt{FL\_YELLOW}
- \texttt{FL\_BLUE}
- \texttt{FL\_MAGENTA}
- \texttt{FL\_CYAN}
- \texttt{FL\_WHITE}
- \texttt{FL\_WHITE}

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

- \texttt{FL\_FOREGROUND\_COLOR}
- \texttt{FL\_BACKGROUND\_COLOR}
- \texttt{FL\_INACTIVE\_COLOR}
- \texttt{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 \texttt{fl\_rgb\_color()} function, and decomposed again with \texttt{Fl::get\_color()}:

\begin{verbatim}
Fl\_Color c = fl\_rgb\_color(85, 170, 255);  // RGB to Fl\_Color
Fl\_get\_color(c, r, g, b);  // Fl\_Color to RGB
\end{verbatim}

The widget color is set using the \texttt{color()} method:

\begin{verbatim}
button->color(FL\_RED);  // set color using named value
\end{verbatim}

Similarly, the label color is set using the \texttt{labelcolor()} method:

\begin{verbatim}
button->labelcolor(FL\_WHITE);
\end{verbatim}

The \texttt{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:

\begin{verbatim}
button->color(0x000000ff);  // colormap index #255 (FL\_WHITE)
\end{verbatim}

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

\begin{verbatim}
button->color(0xff000000);  // RGB: red
button->color(0x00ff0000);  // RGB: green
button->color(0x0000ff00);  // RGB: blue
button->color(0xffffff00);  // RGB: white
\end{verbatim}
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`. Figure 3-3 shows the standard box types included with FLTK.

![FLTK 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 Figure 3-3 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)**
\textbf{1.4 Common Widgets and Attributes}

\texttt{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 \texttt{Drawing Functions} for more details.

\texttt{Fl\_Boxtype fl\_frame(Fl\_Boxtype b)}

\texttt{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 \texttt{Drawing Functions} for more details.

\texttt{Fl\_Boxtype fl\_box(Fl\_Boxtype b)}

\texttt{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 \texttt{Drawing Functions} for more details.

Adding Your Box Type

The \texttt{Fl::set\_boxtype()} method adds or replaces the specified box type:

\texttt{#define XYZ\_BOX FL\_FREE\_BOXTYPE}

\texttt{Fl::set\_boxtype(XYZ\_BOX, xyz\_draw, 1, 1, 2, 2);}

The last 4 arguments to \texttt{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 \texttt{fl\_down(Fl\_Boxtype)} expects the neutral design on a boxtype with a numerical value evenly dividable by two. \texttt{fl\_frame(Fl\_Boxtype)} expects the UP\_BOX design at a value dividable by four.

1.4.8 Labels and Label Types

The \texttt{label()}, \texttt{align()}, \texttt{labelfont()}, \texttt{labelsize()}, \texttt{labeltype()}, \texttt{image()}, and \texttt{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. Figure 3-4 shows the available symbols.

![Figure 1.5 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>".
1.4 Common Widgets and Attributes

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 thought the text is raised.
- `FL_ICON_LABEL` - draws the icon associated with the text.
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);
```

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

```c
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);
```

**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.

**Hint:**

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); }
  ...
}
```

```c
w->callback(my_static_callback, (void *)this);
```

1.4.10 Shortcuts

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

```c
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 Designing a Simple Text Editor

This chapter takes you through the design of a simple FLTK-based text editor.
1.5.1 Determining the Goals of the Text Editor

Since this will be the first big project you'll be doing with FLTK, let's define what we want our text editor to do:

1. Provide a menubar/menus for all functions.
2. Edit a single text file, possibly with multiple views.
3. Load from a file.
4. Save to a file.
5. Cut/copy/delete/paste functions.
6. Search and replace functions.
7. Keep track of when the file has been changed.

1.5.2 Designing the Main Window

Now that we've outlined the goals for our editor, we can begin with the design of our GUI. Obviously the first thing that we need is a window, which we'll place inside a class called EditorWindow:

```cpp
class EditorWindow : public Fl_Double_Window {
public:
    EditorWindow(int w, int h, const char* t);
    ~EditorWindow();
    Fl_Window *replace_dlg;
    Fl_Input *replace_find;
    Fl_Input *replace_with;
    Fl_Button *replace_all;
    Fl_Return_Button *replace_next;
    Fl_Button *replace_cancel;
    Fl_Text_Editor *editor;
    char search[256];
};
```

1.5.3 Variables

Our text editor will need some global variables to keep track of things:

```cpp
int changed = 0;
char filename[256] = "";
Fl_Text_Buffer *textbuf;
```

The `textbuf` variable is the text editor buffer for our window class described previously. We'll cover the other variables as we build the application.

1.5.4 Menubars and Menus

The first goal requires us to use a menubar and menus that define each function the editor needs to perform. The `Fl_Menu_Item` structure is used to define the menus and items in a menubar:

```cpp
Fl_Menu_Item menuitems[] = {
    { "File", 0, 0, 0, FL_SUBMENU },
    { "New File", 0, 0, FL_CALLBACK, new_cb },
    { "Open File...", FL_COMMAND + 'o', (Fl_Callback *)open_cb },
    { "Insert File...", FL_COMMAND + 'i', (Fl_Callback *)insert_cb, 0, FL_MENU_DIVIDER },
    { "Save File", FL_COMMAND + 's', (Fl_Callback *)save_cb },
    { "Save File &As...", FL_COMMAND + FL_SHIFT + 's', (Fl_Callback *)saveas_cb, 0, FL_MENU_DIVIDER },
    { "New &View", FL_ALT + 'v', (Fl_Callback *)view_cb, 0 },
    { "Close &View", FL_COMMAND + 'w', (Fl_Callback *)close_cb, 0, FL_MENU_DIVIDER },
    { "E&xit", FL_COMMAND + 'q', (Fl_Callback *)quit_cb, 0 },
};
```
1.5 Designing a Simple Text Editor

Once we have the menus defined we can create the Fl_Menu_Bar widget and assign the menus to it with:

```cpp
Fl_Menu_Bar *m = new Fl_Menu_Bar(0, 0, 640, 30);
m->copy(menuitems);
```

We'll define the callback functions later.

1.5.5 Editing the Text

To keep things simple our text editor will use the Fl_Text_Editor widget to edit the text:

```cpp
w->editor = new Fl_Text_Editor(0, 30, 640, 370);
w->editor->buffer(textbuf);
```

So that we can keep track of changes to the file, we also want to add a "modify" callback:

```cpp
textbuf->add_modify_callback(changed_cb, w);
textbuf->call_modify_callbacks();
```

Finally, we want to use a mono-spaced font like FL_COURIER:

```cpp
w->editor->textfont(FL_COURIER);
```

1.5.6 The Replace Dialog

We can use the FLTK convenience functions for many of the editor's dialogs, however the replace dialog needs its own custom window. To keep things simple we will have a "find" string, a "replace" string, and "replace all", "replace next", and "cancel" buttons. The strings are just Fl_Input widgets, the "replace all" and "cancel" buttons are Fl_Button widgets, and the "replace next" button is a Fl_Return_Button widget:

![Figure 1.6 The search and replace dialog](image)

```cpp
FL_Window *replace_dlg = new FL_Window(300, 105, "Replace");
FL_Button *replace_find = new FL_Button(70, 10, 200, 25, "Find:");
FL_Button *replace_with = new FL_Button(70, 40, 200, 25, "Replace:");
FL_Button *replace_all = new FL_Button(10, 70, 90, 25, "Replace All");
FL/Button *replace_next = new FL/Button(105, 70, 120, 25, "Replace Next");
FL/Button *replace_cancel = new FL/Button(230, 70, 60, 25, "Cancel");
```
Now that we've defined the GUI components of our editor, we need to define our callback functions.

### 1.5.7.1 changed_cb()

This function will be called whenever the user changes any text in the editor widget:

```c
void changed_cb(int nInserted, int nDeleted, const char*, void* v) {
    if (nInserted || nDeleted) changed = 1;
    EditorWindow *w = (EditorWindow *)v;
    set_title(w);
    if (loading) w->editor->show_insert_position();
}
```

The `set_title()` function is one that we will write to set the changed status on the current file. We're doing it this way because we want to show the changed status in the window's title bar.

### 1.5.7.2 copy_cb()

This callback function will call `Fl_Text_Editor::kf_copy()` to copy the currently selected text to the clipboard:

```c
void copy_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    Fl_Text_Editor::kf_copy(0, e->editor);
}
```

### 1.5.7.3 cut_cb()

This callback function will call `Fl_Text_Editor::kf_cut()` to cut the currently selected text to the clipboard:

```c
void cut_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    Fl_Text_Editor::kf_cut(0, e->editor);
}
```

### 1.5.7.4 delete_cb()

This callback function will call `Fl_Text_Buffer::remove_selection()` to delete the currently selected text to the clipboard:

```c
void delete_cb(Fl_Widget*, void* v) {
    textbuf->remove_selection();
}
```

### 1.5.7.5 find_cb()

This callback function asks for a search string using the `fl_input()` convenience function and then calls the `find2_cb` function to find the string:

```c
void find_cb(Fl_Widget* w, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *val;
    val = fl_input("Search String:", e->search);
    if (val != NULL) {
        // User entered a string - go find it!
        strcpy(e->search, val);
        find2_cb(w, v);
    }
}
```
1.5.7.6 find2_cb()

This function will find the next occurrence of the search string. If the search string is blank then we want to pop up the search dialog:

```c
void find2_cb(Fl_Widget* w, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    if (e->search[0] == ' ') {
        // Search string is blank; get a new one...
        find_cb(w, v);
        return;
    }
    int pos = e->editor->insert_position();
    int found = textbuf->search_forward(pos, e->search, &pos);
    if (found) {
        // Found a match; select and update the position...
        textbuf->select(pos, pos+strlen(e->search));
        e->editor->insert_position(pos+strlen(e->search));
        e->editor->show_insert_position();
    } else fl_alert("No occurrences of \""%s\"\" found!", e->search);
}
```

If the search string cannot be found we use the fl_alert() convenience function to display a message to that effect.

1.5.7.7 new_cb()

This callback function will clear the editor widget and current filename. It also calls the check_save() function to give the user the opportunity to save the current file first as needed:

```c
void new_cb(Fl_Widget*, void*) {
    if (!check_save()) return;
    filename[0] = '\0';
    textbuf->select(0, textbuf->length());
    textbuf->remove_selection();
    changed = 0;
    textbuf->call_modify_callbacks();
}
```

1.5.7.8 open_cb()

This callback function will ask the user for a filename and then load the specified file into the input widget and current filename. It also calls the check_save() function to give the user the opportunity to save the current file first as needed:

```c
void open_cb(Fl_Widget*, void*) {
    if (!check_save()) return;
    char *newfile = fl_file_chooser("Open File?", ",*, filename);,
    if (newfile != NULL) load_file(newfile, -1);
}
```

We call the load_file() function to actually load the file.

1.5.7.9 paste_cb()

This callback function will call FL_Text_Editor::kf_paste() to paste the clipboard at the current position:

```c
void paste_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    FL_Text_Editor::kf_paste(0, e->editor);
}
```
1.5.7.10 quit_cb()

The quit callback will first see if the current file has been modified, and if so give the user a chance to save it. It then exits from the program:

```c
void quit_cb(Fl_Widget*, void*){
    if (changed && !check_save())
        return;
    exit(0);
}
```

1.5.7.11 replace_cb()

The replace callback just shows the replace dialog:

```c
void replace_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    e->replace_dlg->show();
}
```

1.5.7.12 replace2_cb()

This callback will replace the next occurrence of the replacement string. If nothing has been entered for the replacement string, then the replace dialog is displayed instead:

```c
void replace2_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *find = e->replace_find->value();
    const char *replace = e->replace_with->value();
    if (find[0] == '\0') {
        // Search string is blank; get a new one...
        e->replace_dlg->show();
        return;
    }
    e->replace_dlg->hide();
    int pos = e->editor->insert_position();
    int found = textbuf->search_forward(pos, find, &pos);
    if (found) {
        // Found a match; update the position and replace text...
        textbuf->select(pos, pos+strlen(find));
        textbuf->remove_selection();
        textbuf->insert(pos, replace);
        textbuf->select(pos, pos+strlen(replace));
        e->editor->insert_position(pos+strlen(replace));
        e->editor->show_insert_position();
    } else fl_alert("No occurrences of \"%s\" found!", find);
}
```

1.5.7.13 replall_cb()

This callback will replace all occurrences of the search string in the file:

```c
void replall_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *find = e->replace_find->value();
    const char *replace = e->replace_with->value();
    find = e->replace_find->value();
    if (find[0] == '\0') {
        // Search string is blank; get a new one...
        e->replace_dlg->show();
        return;
    }
    e->replace_dlg->hide();
    e->editor->insert_position(0);
```
1.5 Designing a Simple Text Editor

```c
int times = 0;
// Loop through the whole string
for (int found = 1; found;)
    int pos = e->editor->insert_position();
found = textbuf->search_forward(pos, find, &pos);
    if (found)
        // Found a match; update the position and replace text...
        textbuf->select(pos, pos+strlen(find));
        textbuf->remove_selection();
        textbuf->insert(pos, replace);
        e->editor->insert_position(pos+strlen(replace));
        times++;
    }
if (times)
    fl_message("Replaced %d occurrences.", times);
else
    fl_alert("No occurrences of '%s' found!", find);
```

1.5.7.14 replcan_cb()

This callback just hides the replace dialog:
```c
void replcan_cb(Fl_Widget*, void* v)
    EditorWindow* e = (EditorWindow*)v;
    e->replace_dlg->hide();
```

1.5.7.15 save_cb()

This callback saves the current file. If the current filename is blank it calls the "save as" callback:
```c
void save_cb(void)
    { char filename[256];
      if (filename[0] == '"')
        // No filename - get one!
        saveas_cb();
      return;
      else
        save_file(filename);
    }
```
The `save_file()` function saves the current file to the specified filename.

1.5.7.16 saveas_cb()

This callback asks the user for a filename and saves the current file:
```c
void saveas_cb(void)
    { char *newfile;
      newfile = fl_file_chooser("Save File As?", ",", filename);
      if (newfile != NULL) save_file(newfile);
    }
```
The `save_file()` function saves the current file to the specified filename.

1.5.8 Other Functions

Now that we've defined the callback functions, we need our support functions to make it all work:
1.5.8.1 check_save()

This function checks to see if the current file needs to be saved. If so, it asks the user if they want to save it:

```c
int check_save(void) {
    if (!changed) return 1;

    int r = fl_choice("The current file has not been saved.\n" "Would you like to save it now?", "Cancel", "Save", "Discard");

    if (r == 1) {
        save_cb(); // Save the file...
        return !changed;
    }

    return (r == 2) ? 1 : 0;
}
```

1.5.8.2 load_file()

This function loads the specified file into the textbuf variable:

```c
int loading = 0;
void load_file(char *newfile, int ipos) {
    loading = 1;
    int insert = (ipos != -1);
    changed = insert;
    if (!insert) strcpy(filename, "")
    int r;
    if (!insert) r = textbuf->loadfile(newfile);
    else r = textbuf->insertfile(newfile, ipos);
    if (r)
        fl_alert("Error reading from file \"%s\":\n%s.", newfile, strerror(errno));
    else
        if (!insert) strcpy(filename, newfile);
    loading = 0;
    textbuf->call_modify_callbacks();
}
```

When loading the file we use the Fl_Text_Buffer::loadfile() method to "replace" the text in the buffer, or the Fl_Text_Buffer::insertfile() method to insert text in the buffer from the named file.

1.5.8.3 save_file()

This function saves the current buffer to the specified file:

```c
void save_file(char *newfile) {
    if (textbuf->savefile(newfile))
        fl_alert("Error writing to file \"%s\":\n%s.", newfile, strerror(errno));
    else
        strcpy(filename, newfile);
    changed = 0;
    textbuf->call_modify_callbacks();
}
```

1.5.8.4 set_title()

This function saves the current buffer to the specified file:

```c
void set_title(Fl_Window* w) {
    if (filename[0] == '\0') strcpy(title, "Untitled");
    else {
        char *slash;
        slash = strrchr(filename, '/');
        #ifdef WIN32
        if (slash == NULL) slash = strrchr(filename, '\');
        #endif
        if (slash != NULL) strcpy(title, slash + 1);
        else strcpy(title, filename);
    }

    if (changed) strcat(title, " (modified)");
    w->label(title);
}
```
1.5.9 The main() Function

Once we've created all of the support functions, the only thing left is to tie them all together with the main() function. The main() function creates a new text buffer, creates a new view (window) for the text, shows the window, loads the file on the command-line (if any), and then enters the FLTK event loop:

```c
int main(int argc, char **argv) {
    textbuf = new Fl_Text_Buffer;
    Fl_Window* window = new_view();
    window->show(1, argv);
    if (argc > 1) load_file(argv[1], -1);
    return Fl::run();
}
```

1.5.10 Compiling the Editor

The complete source for our text editor can be found in the test/editor.cxx source file. Both the Makefile and Visual C++ workspace include the necessary rules to build the editor. You can also compile it using a standard compiler with:

```
CC -o editor editor.cxx -lfltk -lXext -lX11 -lm
```

or by using the fltk-config script with:

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

As noted in Compiling Programs with Standard Compilers, you may need to include compiler and linker options to tell them where to find the FLTK library. Also, the CC command may also be called gcc or c++ on your system.

Congratulations, you've just built your own text editor!

1.5.11 The Final Product

The final editor window should look like the image in Figure 4-2.
1.5.12 Advanced Features

Now that we've implemented the basic functionality, it is time to show off some of the advanced features of the Fl_Text_Editor widget.

1.5.12.1 Syntax Highlighting

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 http://www.nedit.org/, 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[] = { // Style table
    { 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
Fl_Text_Buffer *stylebuf;
w->editor->highlight_data(stylebuf, 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
textbuf->add_modify_callback(style_update, w->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 char on line
    *style, // Style data
    *text; // Text data

    // If this is just a selection change, just unselect the style buffer...
```
if (nInserted == 0 && nDeleted == 0)
{
    stylebuf->unselect();
    return;
}

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

// Select the area that was just updated to avoid unnecessary
// callbacks...
stylebuf->select(pos, pos + nInserted - nDeleted);

// Re-parse the changed region; we do this by parsing from the
// beginning of the line of the changed region to the end of
// the line of the changed region... Then we check the last
// style character and keep updating if we have a multi-line
// comment character...
start = textbuf->line_start(pos);
end = textbuf->line_end(pos + nInserted - nDeleted);
text = textbuf->text_range(start, end);
style = stylebuf->text_range(start, end);
last = style[end - start - 1];
style_parse(text, style, end - start);
stylebuf->replace(start, end, style);
((Fl_Text_Editor *)cbArg)->redisplay_range(start, end);

free(text);
free(style);
}

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:

```
void style_parse(const char *text,
    char *style,
    int length) {
    char current;
    int col;
    int last;
    char buf[255],
        *bufptr;
    const char *temp;
    for (current = *style, col = 0, last = 0; length > 0; length --, text ++) {
        if (current == 'A') {
            // Check for directives, comments, strings, and keywords...
            if (col == 0 && *text == '#') {
                // Set style to directive
                current = 'E';
            } else if (strncmp(text, '//', 2) == 0) {
                current = 'B';
            } else if (strncmp(text, "/*", 2) == 0) {
                current = 'C';
            } else if (strncmp(text, "*/", 2) == 0) {
                current = 'D';
            }
        } else if (current == 'B') {
            current = 'B';
        } else if (current == 'C') {
            current = 'C';
        } else if (current == 'D') {
            current = 'D';
        } else if (current == 'E') {
            current = 'E';
        } else if (current == 'F') {
            current = 'F';
        } else if (current == 'G') {
            current = 'G';
        } else if (current == 'H') {
            current = 'H';
        } else if (current == 'I') {
            current = 'I';
        } else if (current == 'J') {
            current = 'J';
        } else if (current == 'K') {
            current = 'K';
        } else if (current == 'L') {
            current = 'L';
        } else if (current == 'M') {
            current = 'M';
        } else if (current == 'N') {
            current = 'N';
        } else if (current == 'O') {
            current = 'O';
        } else if (current == 'P') {
            current = 'P';
        } else if (current == 'Q') {
            current = 'Q';
        } else if (current == 'R') {
            current = 'R';
        } else if (current == 'S') {
            current = 'S';
        } else if (current == 'T') {
            current = 'T';
        } else if (current == 'U') {
            current = 'U';
        } else if (current == 'V') {
            current = 'V';
        } else if (current == 'W') {
            current = 'W';
        } else if (current == 'X') {
            current = 'X';
        } else if (current == 'Y') {
            current = 'Y';
        } else if (current == 'Z') {
            current = 'Z';
        }
        bufptr = &buf[length - 1];
        *bufptr = current;
        current = *text;
    }
    style = new char[length + 1];
    memset(style, 'A', length);
    style[length] = '\0';
    stylebuf->replace(pos, pos + nDeleted, style);
    delete[] style;
    // Just delete characters in the style buffer...
    stylebuf->remove(pos, pos + nDeleted);
}
```

Generated by Doxygen
} else if (strncmp(text, "\\n", 2) == 0) {
    // Quoted quote...
    *style++ = current;
    *style++ = current;
    text ++;
    length --;
    col += 2;
    continue;
} else if (*text == '\n') {
    current = 'D';
} else if (!last && islower(*text)) {
    // Might be a keyword...
    for (temp = text, bufptr = buf;
        islower(*temp) && bufptr < (buf + sizeof(buf) - 1);
        bufptr++ = *temp+++);
    if (!islower(*temp)) {
        *bufptr = '\0';
        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 (current == 'C' && strncmp(text, "*/", 2) == 0) {
    // Close a C comment...
    *style++ = current;
    *style++ = current;
    text ++;
    length --;
    current = 'A';
    col += 2;
    continue;
} else if (current == 'D') {
    // Continuing in string...
    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;
    }
}

// Copy style info...
if (current == 'A' && (*text == '{' || *text == '}')) *style++ = 'G';
else *style++ = current;
1.6 Drawing Things in FLTK

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

1.6.1 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. Memorize what is the current drawing surface calling `Fl_Surface_Device::surface()`, and make your surface the new current drawing surface calling the surface’s `set_current()` 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 the previous surface’s `set_current()`.

1.6.1.1 What Drawing Unit do FLTK drawing functions use?

When drawing to the display or to instances of `Fl_Copy_Surface` and `Fl_Image_Surface`, the unit of drawing functions corresponds generally to one pixel. The so-called ‘retina’ displays of some recent Apple computers are an exception to this rule: one drawing unit corresponds to the width or the height of 2 display pixels on a retina display.

When drawing to surfaces that are instances of `Fl_Paged_Device` derived classes (i.e., `Fl_Printer` or `Fl_PostScript_File_Device`), the drawing unit is initially one point, that is, 1/72 of an inch. But this unit is changed after calls to `Fl_Paged_Device::scale()`.
1.6.2 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
- 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

1.6.2.1 Boxes

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

```c
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 boxtype `b` in the specified color `c`.

```c
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.
1.6 Drawing Things in FLTK

1.6.2.2 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 pixels 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_pop_clip()
```

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.
1.6.3 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 1.3.x 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:

```
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 1.3.x colormap or a direct RGB value on TrueColor screens.

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.

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()  
Fl::foreground()  
Fl::background()  
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()  
Fl::free\_color(Fl\_Color i, int overlay)  
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:
// COLOR MAP INDEX
color(0x00000000)
-------- |
 | |
 | Color map index (8 bits)
 Must be zero
button->color(0x000000ff); // colormap index #255 (FL_WHITE)

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

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

1.6.3.1 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.

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.
• FL_SOLID  ------
• FL_DASH  - - - -
• FL_DOT  ......  
• FL_DASHDOT  - . - .
• FL_DASHDOTDOT  - . . -
• FL_CAP_FLAT
• FL_CAP_ROUND
• FL_CAP_SQUARE (extends past end point 1/2 line width)
• FL_JOIN_MITER (pointed)
• FL_JOIN_ROUND
• FL_JOIN_BEVEL (flat)

`width` is the number of pixels 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 pixels. 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.

### 1.6.3.2 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).

```c
void fl_point(int x, int y)
```

Draw a single pixel at the given coordinates.

```c
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.

```c
void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)
```

Generated by Doxygen
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 \textit{inside} this bounding box.

```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 \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’oclock and are the starting and ending angle of the arc, a2 must be greater or equal to a1.

`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.

Todo add an Fl_Draw_Area_Cb typedef to allow fl_scroll(...) to be doxygenated?

```c
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. The contents of the rectangular area is first shifted by dx and dy pixels. The callback is then called for every newly exposed rectangular area.

### 1.6.3.3 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_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 \texttt{fl_transformed_vertex()}. 

\begin{verbatim}
void fl_begin_points()
void fl_end_points()
\end{verbatim}

Start and end drawing a list of points. Points are added to the list with \texttt{fl_vertex()}. 

\begin{verbatim}
void fl_begin_line()
void fl_end_line()
\end{verbatim}

Start and end drawing lines. 

\begin{verbatim}
void fl_begin_loop()
void fl_end_loop()
\end{verbatim}

Start and end drawing a closed sequence of lines. 

\begin{verbatim}
void fl_begin_polygon()
void fl_end_polygon()
\end{verbatim}

Start and end drawing a convex filled polygon. 

\begin{verbatim}
void fl_begin_complex_polygon()
void fl_gap()
void fl_end_complex_polygon()
\end{verbatim}

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 \texttt{fl_gap()} to separate loops of the path. It is unnecessary but harmless to call \texttt{fl_gap()} before the first vertex, after the last one, or several times in a row. 

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

\textbf{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. 

\begin{verbatim}
void fl_vertex(double x,double y)
\end{verbatim}
1.6 Drawing Things in FLTK

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 Bezier 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.

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

`fl_circle(...)` is equivalent to `fl_arc(..., 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.

### 1.6.3.4 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 \(n\) characters starting at the given 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.

```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 pixel width of a string or single character
- `fl_height()` – measure the pixel 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()`.

See also

- `fl_text_extents()`, `fl_measure()`, `fl_width()`, `fl_descent()`

```c
int fl_descent()
```

Recommended distance above the bottom of a `fl_height()` tall box to draw the text at so it looks centered vertically in that box.

```c
double fl_width(const char* txt)
double fl_width(const char* txt, int n)
double fl_width(unsigned int unicode_char)
```
Return the pixel width of a nul-terminated string, a sequence of \( n \) characters, or a single character in the current font.

See also

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

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

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

Given a string "txt" drawn using \texttt{fl\_draw(txt, x, y)} you would determine its pixel extents 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()}

\texttt{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.

1.6.3.5 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: \texttt{FL\_HELVETICA}, \texttt{FL\_TIMES}, \texttt{FL\_COURIER}, and modifier values \texttt{FL\_BOLD} and \texttt{FL\_ITALIC} which can be added to these, and \texttt{FL\_SYMBOL} and \texttt{FL\_ZAPF\_DINGBATS}. Faces greater than 255 cannot be used in \texttt{Fl\_Widget} labels, since \texttt{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.

\texttt{void fl\_font(int face, int size)}

Generated by Doxygen
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 *pixels* and not "points". Lines should be spaced *size* pixels apart or more.

```
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.

### 1.6.3.6 Character Encoding

FLTK 1.3 expects 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.

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

### 1.6.3.7 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.

```
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.

### 1.6.4 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.

---

Generated by Doxygen
1.6 Drawing Things in FLTK

1.6.4.1 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.

```c
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 \( b \) is 4 or more, you must fill in the unused bytes with zero.

```c
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:

```c
#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.

```c
int fl_measure_pixmap(char const* data, int &w, int &h)
int fl_measure_pixmap(const char const* cdata, int &w, int &h)
```

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.

### 1.6.4.2 Direct Image Reading

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

```c
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 \( W \times H \times 3 \) bytes when reading RGB images and \( W \times H \times 4 \) 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.
1.6.4.3 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.

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_JPEG_Image
- Fl_PNG_Image
- Fl_PNM_Image
- Fl_XBM_Image
- Fl_XPM_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. It can also be used to scale an image to a certain size in drawing units, independently from its size in pixels (see Fl_Shared_Image::scale()).

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 void Fl_Image::copy()
virtual Fl_Image* Fl_Image::copy(int w, int h)
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`.

```cpp
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`.

```cpp
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.

### 1.6.4.4 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/x.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**

The X11 platform requires an open display for offscreen drawing, i.e. you may need to call `fl_open_display()` prior to creating and using offscreen buffers, particularly if no window has been shown yet.
Note

In FLTK 1.3.x and earlier versions all offscreen drawing functions described below are implemented as macros and create certain temporary variables to save context information. You may need to create local scope blocks with curly braces { ... } if you use offscreen functions more than once in a function or method.

Example:

```c
fl_open_display(); // necessary before showing the first window
Fl_Offscreen oscr = fl_create_offscreen(120, 120);
{ // begin block
  fl_begin_offscreen(oscr);
  fl_color(FL_WHITE);
  fl_rectf(0, 0, 120, 120);
  fl_end_offscreen();
} // end block
// other code here
{ // begin block
  fl_begin_offscreen(oscr);
  fl_color(FL_BLACK);
  fl_rectf(10, 10, 100, 100);
  fl_end_offscreen();
} // end block
// other code here
fl_delete_offscreen(oscr);
```

Note

In FLTK 1.4.0 and later neither calling fl_open_display() nor using local blocks is necessary since the offscreen functions described below are real functions (not macros as in 1.3.x).

```c
Fl_Offscreen fl_create_offscreen(int w, int h)
```

Create an RGB offscreen buffer with $w \times h$ pixels.

```c
void fl_delete_offscreen(Fl_Offscreen)
```

Delete a previously created offscreen buffer. All drawings are lost.

```c
void fl_begin_offscreen(Fl_Offscreen)
```

Send all subsequent drawing commands to this offscreen buffer.

```c
void fl_end_offscreen()
```

Quit sending drawing commands to this offscreen buffer.

```c
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 buffer at $(x, y)$.
1.7 Handling Events

This chapter discusses the FLTK event model and how to handle events in your program or widget.

1.7.1 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.

1.7.2 Mouse Events

1.7.2.1 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.

1.7.2.2 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`.

1.7.2.3 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`.

1.7.2.4 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`. 
1.7 Handling Events

1.7.2.5 FL_MOUSEWHEEL

The user has moved the mouse wheel. The Fl::event_dx() and Fl::event_dy() methods can be used to find the amount to scroll horizontally and vertically.

1.7.3 Focus Events

1.7.3.1 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 handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.

1.7.3.2 FL_LEAVE

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.

1.7.3.3 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 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.

1.7.3.4 FL_UNFOCUS

This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.
1.7.4 Keyboard Events

1.7.4.1 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, you need Fl::event_key(int) to detect if the key is being held down during FL_KEYUP or not.

1.7.4.2 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.

1.7.5 Widget Events

1.7.5.1 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).

1.7.5.2 FL_ACTIVATE

This widget is now active, due to activate() being called on it or one of its parents.
1.7 Handling Events

1.7.5.3 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).

1.7.5.4 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.

1.7.6 Clipboard Events

1.7.6.1 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()`.

1.7.6.2 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.

1.7.7 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. Files are received as a list of full path and file names, separated by newline.

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://`).

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`.

Generated by Doxygen
1.7.7.1 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.

1.7.7.2 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.

1.7.7.3 FL_DND_LEAVE

The mouse has moved out of the widget.

1.7.7.4 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.

1.7.8 Other events

1.7.8.1 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.

1.7.8.2 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.

1.7.9 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()
1.7 Handling Events

- 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()

1.7.10 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.

### 1.7.11 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.

### 1.8 Adding and Extending Widgets

This chapter describes how to add your own widgets or extend existing widgets in FLTK.
1.8 Adding and Extending Widgets

1.8.1 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, and Fl_Window) 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.

1.8.2 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.

1.8.3 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);
labelfont(FL_NORMAL_LABEL);
labelstyle(FL_NORMAL_STYLE);
labelsize(FL_NORMAL_SIZE);
labelcolor(FL_NORMAL_COLOR);
align(FL_ALIGN_CENTER);
callback(default_callback,0);
flags(ACTIVE|VISIBLE);
image(0);
delimage(0);
1.8.4 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 behaviour 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 \texttt{t} as the box type and \texttt{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 \texttt{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 \texttt{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 \texttt{set_flag(SHORTCUT_LABEL)} modifies the behavior of \texttt{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 \texttt{Fl_Widget::hide()} and \texttt{Fl_Widget::show()}. These do not send the \texttt{FL_HIDE} and \texttt{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 \texttt{Fl_Widget::label()} against the current event (which should be a \texttt{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 \texttt{SHORTCUT_LABEL} flag is off, if the label is \texttt{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.

Todo Clarify Fl_Widget::testShortcut() explanations. Fl_Widget.h says Internal Use only, but subclassing chapter gives details!

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).

1.8.5 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:

```cpp
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;
```
1.8 Adding and Extending Widgets

```cpp
    redraw();
    }
    return 1;
  case FL_RELEASE:
    if (highlight) {
      highlight = 0;
      redraw();
      // 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.:

```cpp
#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)"
 ..
}
```

1.8.6 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.

1.8.7 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.
1.8.8 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:

```
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:

```
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:

```
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:

```
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:

```
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_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()

```
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.

### 1.8.9 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.
1.8.10 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

1.8.11 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.

1.9 Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

1.9.1 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. 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/g1.h> header file. It will include the file <GL/g1.h>, define some extra drawing functions provided by FLTK, and include the <windows.h> header file needed by WIN32 applications.

Some simple coding rules (see OpenGL and 'retina' displays) allow to write cross-platform code that will draw high resolution OpenGL graphics if run on 'retina' displays with Mac OS X.
1.9 Using OpenGL

1.9.2 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.

1.9.2.1 Defining the Subclass

To define the subclass you just subclass the Fl_Gl_Window class:

```c
class MyWindow : public Fl_Gl_Window {
    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.)

1.9.2.2 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 ...
}
```
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 ...
        ... position in `Fl::event_x()` and `Fl::event_y()`
        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(true); // 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.
1.9 Using OpenGL

1.9.3 Using OpenGL in Normal FLTK Windows

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, WIN32, 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`!

1.9.4 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!

```cpp
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.

```cpp
void gl_font(Fl_Font fontid, int size)
```

Sets the current OpenGL font to the same font you get by calling `fl_font()`.

```cpp
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.

```cpp
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.

```cpp
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.

```cpp
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()`.

### 1.9.5 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.
1.9.6 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

```cpp
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...
  csCamera *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_ = (csDrawAction *)0;
      scene_ = (csGroup *)0;
      camera_ = (csCamera *)0;
      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

  // 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.

### 1.9.7 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 Unix/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.
1.10 Programming with FLUID

This chapter shows how to use the Fast Light User-Interface Designer ("FLUID") to create your GUIs.

Subchapters:
1.10.1 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.

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 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 1.9 FLUID organization](image)

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.
1.10 Programming with FLUID

1.10.2 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 host:port`
- `-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 like 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.

1.10.3 Running FLUID Under Microsoft Windows

To run FLUID under WIN32, double-click on the `FLUID.exe` file. You can also run FLUID from the Command Prompt window. FLUID always runs in the background under WIN32.

1.10.4 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 $<
```

Most versions of make support rules that cause `.fl` files to be compiled:

```
.SUFFIXES: .fl .cxx .h
.my_panels.h .my_panels.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.
1.10.5 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 1.10 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 callbacks member functions of a custom widget classes.
3. Subclass an Fl_Gl_Window to suit your purposes.

1.10.5.1 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
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.
  double v_angle(float angle){vAng=angle;};
  // Return the rotation about the vertical (y) axis.
  float v_angle(){return vAng;};
  // Set the rotation about the horizontal (x) axis.
  void h_angle(float angle){hAng=angle;};
  // Return the rotation about the horizontal (x) axis.
  float h_angle(){return hAng;};
  // Sets the x shift of the cube view camera.
  void panx(float x){xshift=x;};
  // Sets the y shift of the cube view camera.
  void pany(float y){yshift=y;};
  // The widget class draw() override.
  void draw();

  private:
  // Draw the cube boundaries
  void draw();
  // Draw the faces of the cube using the boxv[] vertices, using
  // GL_LINE_LOOP for the faces. The color is #defined by
  // CUBECOLOR.
  void drawCube();
};
```

The CubeView Class Implementation

Here is the CubeView implementation. It is very similar to the "cube" demo included with FLTK.

```cpp
#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)
{
  vAng = 0.0; hAng=0.0; size=10.0;
  // 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;
}
```
\};

// The color used for the edges of the bounding cube.
#define CUBECOLOR 255,255,255,255

void CubeView::drawCube() {
    /* Draw a colored cube */
    #define ALPHA 0.5
    glShadeModel(GL_FLAT);
    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);
    glVertex3fv(boxv3);
    glVertex3fv(boxv7);
    glVertex3fv(boxv4);
    glColor4f(0.0, 1.0, 0.0, ALPHA);
    glVertex3fv(boxv1);
    glVertex3fv(boxv5);
    glVertex3fv(boxv6);
    glVertex3fv(boxv2);
    glEnd;
    glBegin(GL_LINES);
    glVertex3fv(boxv0);
    glVertex3fv(boxv1);
    glVertex3fv(boxv1);
    glVertex3fv(boxv2);
    glVertex3fv(boxv2);
    glVertex3fv(boxv3);
    glVertex3fv(boxv3);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv5);
    glVertex3fv(boxv5);
    glVertex3fv(boxv6);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glVertex3fv(boxv7);
    glVertex3fv(boxv4);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv1);
    glVertex3fv(boxv5);
    glVertex3fv(boxv2);
    glVertex3fv(boxv6);
    glVertex3fv(boxv3);
    glVertex3fv(boxv7);
    glEnd();
}
void CubeView::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0, 0, w(), h());
        glOrtho(-10, 10, -10, 10, -20000, 10000);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    }
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(xshift, yshift, 0);
    glRotatef(hAng, 0, 1, 0);
    glRotatef(vAng, 1, 0, 0);
    glScalef(float(size), float(size), float(size));
    drawCube();
    glPopMatrix();
}

1.10.5.2 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.
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:

![FLUID window containing CubeView demo](image)

**Figure 1.12 FLUID window containing CubeView demo**

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 a `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.

![Figure 1.13 CubeView methods](image)

### 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 in 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 1.14 CubeView constructor](image)

Once the new method has been added, highlight its name and select **New->Code->Code**. Enter the method's code in the code window.

### 1.10.5.3 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->Code->Code**. Add any required code.

### 1.10.5.4 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->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.

### 1.10.6 FLUID Reference

The following sections describe each of the windows in FLUID.
1.10.6.1 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).

1.10.6.2 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/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)

This isn't implemented yet. You should do save often so you can recover from any mistakes you make.

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+c)

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/Open... (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)
1.10 Programming with FLUID

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/Overlays on/off (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/Project Settings... (Alt+p)**

Displays the project settings panel.

Under the "Output" 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 "Internationalization" tab are the **internationalization** options, described later in this chapter.

![FLUID Project Settings Window](image)

*Figure 1.15 FLUID Project Settings Window*
Edit/GUI Settings... (Shift+Alt+p)

Displays the GUI Settings panel, used to control the user interface settings.

![FLUID GUI Settings Window](image1)

**Figure 1.16 FLUID GUI Settings Window**

Edit/Global FLTK Settings... (Shift+Alt+g)

Displays the FLTK Global Settings ("Preferences") panel, used to control fluid's user specific and/or system wide settings.

Toolips provide descriptions of each option.

At the lower-right, "User Settings" causes changes to only affect the current user, "System Settings" causes changes to be applied to all users on the current machine.

![FLUID Global Settings Window](image2)

**Figure 1.17 FLUID Global Settings Window**
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 a `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 (`void`).

It is possible to make the `.cxx` output be a self-contained program that can be compiled and executed. This is done by deleting the function name so `main(argc,argv)` is used. The function will call `show()` on all the windows it creates and then call `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.

New/Window

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.
Align all selected widgets to the first widget in the selection.

Space all selected widgets evenly inside the selected space. Widgets will be sorted from first to last.

Make all selected widgets the same size as the first selected widget.

Center all selected widgets relative to their parent widget.

Displays the grid settings panel.

This panel controls the grid 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.

Figure 1.18 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/About FLUID

Pops up a panel showing the version of FLUID.

Help/On FLUID

Shows this chapter of the manual.

Help/Manual

Shows the contents page of the manual.
1.10.6.3 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 "no overlay" 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.

"Revert" or "Cancel" put everything back to when you last brought up the panel or hit OK. However in the current version of FLUID, changes to "visible" attributes (such as the color, label, box) are not undone by revert or cancel. Changes to code like the callbacks are undone, however.

![Figure 1.19 The FLUID widget GUI attributes](image)

1.10.7 GUI Attributes

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.
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 resizeable. You can get more complex behavior by making invisible boxes the resizeable 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 1.20 The FLUID widget Style attributes

1.10.7.1 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.

![FL Button](image)

**Figure 1.21 The FLUID widget C++ attributes**

### 1.10.7.2 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. Currently the only way to get a double-buffered window is to change this field for the window to "Fl_Double_Window" and to add

```cpp
#include <FL/Fl_Double_Window.h>
```

to the extra code.

**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*`.

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.
1.10.8 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 \texttt{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 \texttt{Fl_Widget}. There are three extra items:

1.10.9 Image Labels

The contents of the image files in the \texttt{Image} and \texttt{Inactive} text fields are written to the \texttt{.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 \texttt{.fl} file so you will need the image files as well to read the \texttt{.fl} file. Filenames are relative to the location of the \texttt{.fl} file and not necessarily the current directory. We recommend you either put the images in the same directory as the \texttt{.fl} file, or use absolute path names.

Notes for All Image Types

\texttt{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 "\texttt{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 WIN32 applications for icons. FLUID converts BMP files into (modified) XPM format and uses a 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 a 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 a 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.

## 1.10.10 Internationalization with FLUID

FLUID supports internationalization (I18N for short) of label strings used by widgets. The preferences window (Ctrl+p) provides access to the I18N options.
1.10 Programming with FLUID

1.10.10 I18N Methods

FLUID supports three methods of I18N: use none, use GNU gettext, and use POSIX catgets. The “use 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.

1.10.10.2 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. Two new input fields will then appear to control the include file and function/macro name to use when retrieving the localized label strings.

![Figure 1.22 Internationalization using GNU gettext](image)

The `#include` field controls the header file to include for I18N; by default this is `<libintl.h>`, the standard I18N file for GNU gettext.

The `Function:` field controls the function (or macro) that will retrieve the localized message; by default the `gettext` function will be called.
1.10.10.3 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.

![Preferences Window](image)

**Figure 1.23 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.

1.10.11 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.

1.11 Advanced FLTK

This chapter explains advanced programming and design topics that will help you to get the most out of FLTK.
1.11 Advanced FLTK

1.11.1 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 main() thread of the process is permitted to handle system events, create or destroy windows and open or close windows. Further, only the main() thread of the process can safely write to the display.

To support this in a portable way, all FLTK draw() methods are executed in the 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. (NOTE: A special case exists for 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.

1.11.2 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.
1.11.3 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 */
        ... Store new values in your data structures...
        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. */
            ... Process the message...
        }
    }
    /* 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()`.
1.11 Advanced FLTK

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 reli-
able 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.

1.11.4 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?
1.11.5 FLTK multithreaded Constraints

FLTK supports multiple platforms, some of which allow only the `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 call `show()` or `hide()` anything that contains `Fl_Window` based widgets from a worker thread. This includes any windows, dialogs, file choosers, subwindows or widgets using `Fl_Gl_Window`. Note that this constraint also applies to non-window widgets that have tooltips, since the tooltip will contain a `Fl_Window` object. The safe and portable approach is never to call `show()` or `hide()` on any widget from the context of a worker thread. Instead you can use the `Fl_Awake_Handler` variant of `Fl::awake()` to request the `main()` thread to create, destroy, show or hide the widget on behalf of the worker thread.

- Don't call `Fl::run()`, `Fl::wait()`, `Fl::flush()`, `Fl::check()` or any related methods that will handle system messages from a worker thread

- Don't intermix use of `Fl::awake(Fl_Awake_Handler cb, void* userdata)` and `Fl::awake(void* message)` calls in the same program as they may interact unpredictably on some platforms; choose one or other style of `Fl::awake(<thing>)` mechanism and use that. (Intermixing calls to `Fl::awake()` should be safe with either however.)

- Don't start or cancel timers from a worker thread

- Don't change window decorations or titles from a worker thread

- The `make_current()` method will probably not work well for regular windows, but should always work for a `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: `Fl::lock()`, `Fl::unlock()`, `Fl::awake()`, `Fl::awake(Fl_Awake_Handler cb, void* userdata)`, `Fl::awake(void* message)`, `Fl::thread_message()`.

1.12 Unicode and UTF-8 Support

This chapter explains how FLTK handles international text via Unicode and UTF-8.

Unicode support was only recently added to FLTK and is still incomplete. This chapter is Work in Progress, reflecting the current state of Unicode support.
1.12.1 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:

- [http://www.unicode.org](http://www.unicode.org)
- [http://www.iso.org](http://www.iso.org)
- [http://www.cl.cam.ac.uk/~mgk25/unicode.html](http://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 100,000 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** Do we need this info about planes?

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+0FFFF. 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.

<table>
<thead>
<tr>
<th>Unicode range</th>
<th>Byte sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+00000000 – U+0000007F</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>U+00000080 – U+000007FF</td>
<td>110xxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00000800 – U+0000FFFF</td>
<td>1110xxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00010000 – U+001FFFFF</td>
<td>11110xxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00200000 – U+03FFFFF</td>
<td>11110xxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+04000000 – U+7FFFFFFF</td>
<td>1111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
</tbody>
</table>
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.

1.12.2 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().

1.12.3 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.
1.12 Unicode and UTF-8 Support

- 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.

1.12.4 FLTK Unicode and UTF-8 Functions

This section currently 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

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.

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)
```

```c
int fl_utf8bytes(unsigned ucs)
```

Returns the number of bytes needed to encode `ucs` in UTF-8.

```c
int fl_utf8len(char c)
```

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_nonspace(unsigned int ucs)
```

Returns true if `ucs` is a non-spacing character.

```c
const char * fl_utf8back(const char *p, const char *start, const char *end)
```

```c
const char * fl_utf8fwd(const char *p, const char *start, const char *end)
```

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)
```

```c
int fl_utf8encode(unsigned ucs, char *buf)
```

`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)
```

```c
unsigned int fl_utf8toa(const char *src, unsigned srclen, char *dst, unsigned dstlen)
```

`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`. 

Generated by Doxygen
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 src string, including the zero terminator. Therefore setting dstlen to zero is a way of measuring how much storage would be needed before doing the real conversion.

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
These functions convert between UTF-8 and the locale-specific multi-byte encodings used on some systems for filenames, etc. If `fl_utf8locale()` returns true, these functions don't do anything useful. [Is this all worded correctly?]

```c
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
```

`fl_tolower()` and `fl_toupper()` convert a single Unicode character from upper to lower case, and vice versa. `fl_utf_tolower()` and `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, `buf` length must be at least `3*len` [for 16-bit Unicode]

```c
int fl_utf_strcasecmp(const char *s1, const char *s2) OksiD
int fl_utf_strncasecmp(const char *s1, const char *s2, int n) OksiD
```

`fl_utf_strcasecmp()` is a UTF-8 aware string comparison function that converts the strings to lower case Unicode as part of the comparison. `flt_utf_strncasecmp()` only compares the first `n` characters [bytes?]

### 1.12.5 FLTK Unicode Versions of System Calls

- `int fl_access(const char* f, int mode) OksiD`
- `int fl_chmod(const char* f, int mode) OksiD`
- `int fl_execvp(const char* file, char* const* argv) OksiD`
- `FILE* fl_fopen(const char* f, const char* mode) OksiD`
- `char* fl_getcwd(char* buf, int maxlen) OksiD`
- `char* fl_getenv(const char* name) OksiD`
- `char* fl_make_path(const char* path) - returns char ? OksiD`
- `void fl_make_path_for_file(const char* path) OksiD`
- `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`
1.13 FLTK 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.

1.13.1 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 3
- FL_PATCH_VERSION - The patch release number, currently 6
- FL_VERSION - [Deprecated] A combined floating-point version number for the major, minor, and patch release numbers, currently 1.0306
- FL_API_VERSION - A combined integer version number for the major, minor, and patch release numbers, currently 10306 (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 10300 (default)

Note
The ABI version (FL_ABI_VERSION) is usually constant throughout one major/minor release version, for instance 10300 if FL_API_VERSION is 10305. 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 features", see CHANGES).
1.13.2 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 without 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.

1.13.3 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.
1.13 FLTK Enumerations

1.13.4 `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

1.13.5 `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 `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; use `FL_KP + n` for number `n`.
- `FL_KP_Enter` - The enter key on the keypad.
- `FL_F` - One of the function keys; use `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.

1.13.6 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 WIN32 and X11, or FL_META 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.

1.13.7 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.
1.13 FLTK Enumerations

- 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.

1.13.8 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.
1.13.9 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.

1.13.9.1 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
1.13 FLTK Enumerations

- FL_DARK2
- FL_DARK1
- FL_LIGHT1
- FL_LIGHT2
- FL_LIGHT3

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.

1.13.10 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 - l-beam
- FL_CURSOR_HAND - hand (uparrow on MSWindows)
- 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

1.13.11 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.
1.13.12 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.

1.14 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.

1.14.1 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.

1.14.2 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))`
1.14 GLUT Compatibility

- 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_DAMAGED) 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").

1.14.3 Mixing GLUT and FLTK Code

You can make your GLUT window a child of a FLWindow 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.
1.14.4 class Fl_Glut_Window

1.14.4.1 Class Hierarchy

```
Fl_Gl_Window
  
  +----Fl_Glut_Window
```

1.14.4.2 Include Files

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

1.14.4.3 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()`.

1.14.4.4 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>
## 1.14.4.5 Methods

```cpp
Fl_Glut_Window::Fl_Glut_Window(int x, int y, int w, int h, const char *title = 0)
Fl_Glut_Window::Fl_Glut_Window(int w, int h, const char *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.

```cpp
virtual Fl_Glut_Window::~Fl_Glut_Window()
```

Destroys the GLUT window.

```cpp
void Fl_Glut_Window::make_current()
```

Switches all drawing functions to the GLUT window.

## 1.15 Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

**Warning: The Forms compatibility is deprecated and no longer maintained in FLTK 1.3, and is likely to be removed completely in FLTK 1.4**

### 1.15.1 Importing Forms Layout Files

FLUID can read the .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. 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 FLUID. If you are not interested in this you may have more immediate luck with the forms compatibility header, `<FL/forms.H>`.

### 1.15.2 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 forms.h file supplied with FLTK is included. The forms.h file simply pulls in `<FL/forms.H>` so you don't need to change your source code. Take a look at `<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.

### 1.15.3 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 `"boxtype"` 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_activate()` `fl_set_form_deactivate()`
- `fl_set_form_property()`
- `fl_set_app_mainform(), fl_get_app_mainform()`
1.15 Forms Compatibility

- `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()`
- `flEnumerateFonts()`

Most drawing functions

- `fl_set_coordunit()` *(FLTK uses pixels all the time)*
- `fl_ringbell()`
- `fl_gettime()`
- `flWin*()` *(all these functions)*
- `fl_initialize(argc, argv, x, y, z)` *ignores last 3 arguments*
- `fl_read_bitmapfile()`, `fl_read_pixmapfile()`
- `flAddToBrowserChars()`
- `FL_Menu_Butto` *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()`
- `flShowOneliner()`
- `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
1.15.4 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:
### 1.16 Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

- Accessing the OS Interfaces
- The UNIX (X11) Interface
- The Windows (WIN32) Interface
- The Apple OS X Interface

#### 1.16.1 Accessing the OS Interfaces

All programs that need to access the operating system specific interfaces must include the following header file:

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

Despite the name, this header file will define the appropriate interface for your environment.

**Note**

This header file name "x.H" is changed in FLTK 1.4.0 to the better name "platform.H". Since FLTK 1.3.5 there is a compatibility header file FL/platform.H that includes FL/x.H to help you move to FLTK 1.4.0. If your code is targeted at FLTK 1.3.5 or higher you can safely change it to include FL/platform.H instead. FLTK 1.4.x will keep the file "x.H" for a few releases for backwards compatibility.

The pages that follow describe the functionality that is provided for each operating system.
1.16.2 The UNIX (X11) Interface

The UNIX interface provides access to the X Window System state information and data structures.

1.16.2.1 Handling Other X Events

```c
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`.

```c
extern XEvent ∗fl_xevent
```

This variable contains the most recent X event.

```c
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.

```c
Window fl_xid(const Fl_Window ∗)
```

Returns the XID for a window, or zero if not `shown()`.

```c
Fl_Window ∗fl_find(ulong xid)
```
1.16 Operating System Issues

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.

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.

1.16.2.2 Drawing using Xlib

The following global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current():
extern Display *fl_display;
extern Window fl_window;
extern GC fl_gc;
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:
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.

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.

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.

extern XFontStruct *fl_xfont

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 Xft support, fl_xfont will usually be 0 and fl_xftfont will contain a pointer to the XftFont structure instead.

extern void *fl_xftfont

If FLTK was compiled with Xft 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*.
1.16.2.3 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 WIN32 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_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
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);
```
1.16 Operating System Issues

1.16.2.4 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:

```cpp
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:

```cpp
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 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);
}
```

```cpp
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.

```cpp
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.

```cpp
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::i(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:

```c
void MyWindow::flush() {
    fl_clip_region(Fl_X::i(this)->region);
    Fl_X::i(this)->region = 0;
    if (damage() != 2) {... draw things into backing store ...}
    ... copy backing store to 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:

```c
void MyWindow::hide() {
    if (mypixmap) {
        XFreePixmap(fl_display,mypixmap);
        my pixmap = 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.

1.16.2.5 Setting the Icon of a Window

FLTK currently supports setting a window's icon before it is shown using the Fl_Window::icon() method.

```c
void Fl_Window::icon(const void *)
```
Sets the icon for the window to the passed pointer. You will need to cast the icon Pixmap to a 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 -lXpm).

**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.

### 1.16.2.6 X Resources

When the `Fl_Window::show(int argc, char** argv)` method is called, FLTK looks for the following X resources:

- **background** - The default background color for widgets (color).
- **dndTextOps** - The default setting for drag and drop text operations (boolean).
- **foreground** - The default foreground (label) color for widgets (color).
- **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).
1.16.3 The Windows (WIN32) Interface

The Windows interface provides access to the WIN32 GDI state information and data structures.

1.16.3.1 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.

1.16.3.2 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.

1.16.3.3 Handling Other WIN32 Messages

By default a single WNDCLASSEX 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 WIN32 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.

```cpp
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 WIN32 calls the handle procedures directly for some events (sigh).

```cpp
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()`.

```cpp
HWND fl_xid(const Fl_Window ∗)
```

Returns the window handle for a Fl_Window, or zero if not `shown()`.

```cpp
Fl_Window ∗fl_find(HWND xid)
```

Returns the Fl_Window that corresponds to the given handle, or NULL if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.
1.16.3.4 Drawing Things Using the WIN32 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:

```c
extern HINSTANCE fl_display;
extern HWND fl_window;
extern HDC fl_gc;
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:

```c
DrawSomething(fl_gc, ..., fl_brush());
```

It may also be useful to refer to Fl_Window::current() to get the window’s size or position.

1.16.3.5 Setting the Icon of a Window

FLTK currently supports setting a window’s icon before it is shown using the Fl_Window::icon() method.

```c
void Fl_Window::icon(const void *)
```

Sets 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:

```c
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.

1.16.3.6 How to Not Get a MSDOS Console Window

WIN32 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 WIN32 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.**

WIN32 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, WIN32 applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.
1.16.3.7 Known WIN32 Bugs and Problems

The following is a list of known bugs and problems in the WIN32 version of FLTK:

- If a program is deactivated, \texttt{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 WIN32 and probably impossible to get around.

- \texttt{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.

- \texttt{SetCapture} (used by \texttt{Fl::grab()}) doesn't work, and the main window title bar turns gray while menus are popped up.

- Compilation with \texttt{gcc 3.4.4 and -Os} exposes an optimisation bug in \texttt{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 \texttt{-O2 and -O3} seem to work OK. More details can be found in STR#1656

1.16.4 The Apple OS X Interface

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 \texttt{FL_CTRL}, the 'option' key to \texttt{FL_ALT} and the 'Apple' key to \texttt{FL_META}. Furthermore, \texttt{FL_COMMAND} designates the 'Apple' key on Mac OS X and the 'control' key on other platforms.

Keyboard events return the key name in \texttt{Fl::event_key()} and the keystroke translation in \texttt{Fl::event_text()}. For example, typing Option-Y on a Mac US keyboard will set \texttt{FL_ALT} in \texttt{Fl::event_state()}, set \texttt{Fl::event_key()} to 'y' and return the Y en symbol in \texttt{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:

- \texttt{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:

```cpp
    case FL_PUSH:
    |
    int btn = Fl::event_button();
    ifdef __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
    ifdef __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 reacts sending an FL_CLOSE event to all open windows. If any window remains open, the termination request aborts, and the app continues. If all windows close, FLTK default behaviour is to terminate the application immediately, without letting Fl::run() return. Consequently, potential cleanup code placed after the Fl::run() call does not run, and potential global destructors that would run after main() would return do not run. All code that should run so the app cleanly terminates must therefore be placed in window callbacks (which run when windows are closed) or in atexit() functions. Alternatively, FLTK can be directed to just terminate the event loop and therefore let potential cleanup code placed after return from Fl::run() and from main() execute. This is obtained setting global variable fl_mac_quit_early to 0.

**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.

**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.
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.

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.

1.16.4.1 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.

1.16.4.2 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 global variable fl_gc (of type CGContextRef) is the appropriate Quartz 2D drawing environment. Include FL/x.H to declare the fl_gc variable.

1.16.4.3 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.
1.16.4.4 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 synonyms of w() and h() on non-Mac OS X platforms, so the source code remains cross-platform.

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.

1.16.4.5 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.

1.16.4.6 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

1.17 Migrating Code from FLTK 1.0 to 1.1

This appendix describes the differences between the FLTK 1.0.x and FLTK 1.1.x functions and classes.

1.17.1 Color Values

Color values are now stored in a 32-bit unsigned integer instead of the unsigned character in 1.0.x. This allows for the specification of 24-bit RGB values or 8-bit FLTK color indices.

FL_BLACK and FL_WHITE now remain black and white, even if the base color of the gray ramp is changed using Fl::background(). FL_DARK3 and FL_LIGHT3 can be used instead to draw a very dark or a very bright background hue.

Widgets use the new color symbols FL_FOREGROUND_COLOR, FL_BACKGROUND_COLOR, FL_BACKGROUND2_COLOR, FL_INACTIVE_COLOR, and FL_SELECTION_COLOR. More details can be found in the chapter FLTK Enumerations.

1.17.2 Cut and Paste Support

The FLTK clipboard is now broken into two parts - a local selection value and a cut-and-paste value. This allows FLTK to support things like highlighting and replacing text that was previously cut or copied, which makes FLTK applications behave like traditional GUI applications.

1.17.3 File Chooser

The file chooser in FLTK 1.1.x is significantly different than the one supplied with FLTK 1.0.x. Any code that directly references the old FCB class or members will need to be ported to the new Fl_File_Chooser class.

1.17.4 Function Names

Some function names have changed from FLTK 1.0.x to 1.1.x in order to avoid name space collisions. You can still use the old function names by defining the FLTK_1_0_COMPAT symbol on the command-line when you compile (-DFLTK_1_0_COMPAT) or in your source, e.g.:

```cpp
#define FLTK_1_0_COMPAT
#include <FL/Fl.H>
#include <FL/Enumerations.H>
#include <FL/filename.H>
```

The following table shows the old and new function names:
## 1.18 Migrating Code from FLTK 1.1 to 1.3

This appendix describes the differences between the FLTK 1.1.x and FLTK 1.3.x functions and classes.

### 1.18.1 Migrating From FLTK 1.0

If you want to migrate your code from FLTK 1.0 to FLTK 1.3, then you should first consult Appendix Migrating Code from FLTK 1.0 to 1.1.

<table>
<thead>
<tr>
<th>Old 1.0.x Name</th>
<th>New 1.1.x Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>contrast()</td>
<td>fl_contrast()</td>
</tr>
<tr>
<td>down()</td>
<td>fl_down()</td>
</tr>
<tr>
<td>filename_absolute()</td>
<td>fl_filename_absolute()</td>
</tr>
<tr>
<td>filename_expanded()</td>
<td>fl_filename_expanded()</td>
</tr>
<tr>
<td>filename_ext()</td>
<td>fl_filename_ext()</td>
</tr>
<tr>
<td>filename_isdir()</td>
<td>fl_filename_isdir()</td>
</tr>
<tr>
<td>filename_list()</td>
<td>fl_filename_list()</td>
</tr>
<tr>
<td>filename_match()</td>
<td>fl_filename_match()</td>
</tr>
<tr>
<td>filename_name()</td>
<td>fl_filename_name()</td>
</tr>
<tr>
<td>filename_relative()</td>
<td>fl_filename_relative()</td>
</tr>
<tr>
<td>filename_setext()</td>
<td>fl_filename_setext()</td>
</tr>
<tr>
<td>frame()</td>
<td>fl_frame()</td>
</tr>
<tr>
<td>inactive()</td>
<td>fl_inactive()</td>
</tr>
<tr>
<td>numericsort()</td>
<td>fl_numericsort()</td>
</tr>
</tbody>
</table>

### 1.17.5 Image Support

Image support in FLTK has been significantly revamped in 1.1.x. The Fl_Image class is now a proper base class, with the core image drawing functionality in the Fl_Bitmap, Fl_Pixmap, and Fl_RGB_Image classes.

BMP, GIF, JPEG, PNG, XBM, and XPM image files can now be loaded using the appropriate image classes, and the Fl_Shared_Image class can be used to cache images in memory.

Image labels are no longer provided as an add-on label type. If you use the old label() methods on an image, the widget's image() method is called to set the image as the label.

Image labels in menu items must still use the old labeltype mechanism to preserve source compatibility.

### 1.17.6 Keyboard Navigation

FLTK 1.1.x now supports keyboard navigation and control with all widgets. To restore the old FLTK 1.0.x behavior so that only text widgets get keyboard focus, call the Fl::visible_focus() method to disable it:

```cpp
Fl::visible_focus(0);
```
1.18.2 Fl_Scroll Widget

Fl_Scroll::scroll_to(int x, int y) replaces Fl_Scroll::position(int x, int y).

This change was needed because Fl_Scroll::position(int,int) redefined Fl_Widget::position(int,int), but with a completely different function (moving the scrollbars instead of moving the widget).

Please be aware that you need to change your application's code for all Fl_Scroll-derived widgets, if you used Fl_Scroll::position(int x, int y) to position the scrollbars (not the widget itself).

The compiler will not detect any errors, because your calls to position(int x, int y) will be calling Fl_Widget::position(int x, int y).

1.18.3 Unicode (UTF-8)

FLTK 1.3 uses Unicode (UTF-8) encoding internally. If you are only using characters in the ASCII range (32-127), there is a high probability that you don't need to modify your code. However, if you use international characters (128-255), encoded as e.g. Windows codepage 1252, ISO-8859-1, ISO-8859-15 or any other encoding, then you will need to update your character string constants and widget input data accordingly.

Please refer to the Unicode and UTF-8 Support chapter for more details.

Note

It is important that, although your software uses only ASCII characters for input to FLTK widgets, the user may enter non-ASCII characters, and FLTK will return these characters with UTF-8 encoding to your application, e.g. via Fl_Input::value(). You will need to re-encode them to your (non-UTF-8) encoding, otherwise you might see or print garbage in your data.

1.18.4 Widget Coordinate Representation

FLTK 1.3 changed all Widget coordinate variables and methods, e.g. Fl_Widget::x(), Fl_Widget::y(), Fl_Widget::w(), Fl_Widget::h(), from short (16-bit) to int (32-bit) representation. This should not affect any existing code, but makes it possible to use bigger scroll areas (e.g. Fl_Scroll widget).

1.19 Developer Information

This chapter describes FLTK development and documentation.
Example

/** 
 * Fl_Clock, Fl_Clock_Output widgets. */

/**
 * \class 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"
 * \image html round_clock.png "width=10cm */

 * Returns the displayed time.
 * Returns the time in seconds since the UNIX epoch (January 1, 1970).
 * \see value(ulong)
 */
 ulong value() const {return value_;}

 /**
 * Set the displayed time.
 * Set the time in seconds since the UNIX epoch (January 1, 1970).
 * \param[in] v seconds since epoch
 * \see value()
 */
 void Fl_Clock_Output::value(ulong v) {
 [...]
 }

 /**
 * Create an Fl_Clock widget using the given position, size, and label string.
 * The default boxtype is \c FL_NO_BOX.
 * \param[in] X, Y, W, H position and size of the widget
 * \param[in] L widget label, default is no label
 */
 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.
 * \param[in] t boxtype
 * \param[in] X, Y, W, H position and size of the widget
 * \param[in] L widget label, default is no label
 */
 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

Generated by Doxygen
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:

```c
Gizmo.H
/** 
 * \defgroup Gizmo...
 * A gizmo that does everything
 */
class Gizmo {
    etc
};
extern int popup_gizmo(...);
```

```c
Gizmo.cxx:
/** \relatesalso Gizmo
 * Pops up a gizmo dialog with a Gizmo in it
 */
int popup_gizmo(...);
```

### 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.

The following text is a developer comment.
```html
<!-- *** This *** is *** invisible *** -->
```

This will be visible again.

will be shown as:

```html
The following text is a developer comment.
<!-- *** This *** is *** invisible *** -->
```

This will be visible again.

### 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.

```html
<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>
```

### 1.19.1 Non-ASCII Characters

Doxygen understands many HTML quoting characters like `&quot;`, `&uuml;`, `&ccedil;`, `&Ccedil;`, but not all HTML quoting characters.

This will appear in the document:
Doxygen understands many HTML quoting characters like &quot;, &uuml;, &times;euml;, &ccedil;, but not all HTML quoting characters.

For further informations about HTML quoting characters see http://www.doxygen.org/htmlcmds.html
Alternatively you can use UTF-8 encoding within Doxygen comments.

1.19.2 Document Structure

• \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 "<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.

This page has \
page development I - Developer Information 
at its top.

This section is 
\section development_structure Document Structure 
The following section is 
\section development_links Creating Links

1.19.3 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/htmlcmds.html
  - see <a href="http://www.nedit.org/">Nedit</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/htmlcmds.html
• see Nedit creates a standard HTML link

Generated by Doxygen
1.19.4 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 "html" tags work
\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.
1.20 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:  http://www.fltk.org/str.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 (http://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

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

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by

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 incom-
patible 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.

END OF TERMS AND CONDITIONS

1.21 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 http://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.

1.21.1 Example Applications

<table>
<thead>
<tr>
<th>adjuster</th>
<th>arc</th>
<th>ask</th>
<th>bitmap</th>
<th>blocks</th>
<th>boxtyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>browser</td>
<td>button</td>
<td>buttons</td>
<td>checkers</td>
<td>clock</td>
<td>colbrowser</td>
</tr>
<tr>
<td>color_chooser</td>
<td>cube</td>
<td>CubeView</td>
<td>cursor</td>
<td>curve</td>
<td>demo</td>
</tr>
<tr>
<td>device</td>
<td>doublebuffer</td>
<td>editor</td>
<td>fast_slow</td>
<td>file_chooser</td>
<td>fluid</td>
</tr>
<tr>
<td>fonts</td>
<td>forms</td>
<td>fractals</td>
<td>fullscreen</td>
<td>gl_overlay</td>
<td>glpuzzle</td>
</tr>
<tr>
<td>hello</td>
<td>help</td>
<td>iconize</td>
<td>image</td>
<td>inactive</td>
<td>input</td>
</tr>
<tr>
<td>input_choice</td>
<td>keyboard</td>
<td>label</td>
<td>line_style</td>
<td>list_visuals</td>
<td>mandelbrot</td>
</tr>
<tr>
<td>menubar</td>
<td>message</td>
<td>minimum</td>
<td>navigation</td>
<td>output</td>
<td>overlay</td>
</tr>
<tr>
<td>pack</td>
<td>pixmap_browser</td>
<td>pixmap</td>
<td>preferences</td>
<td>radio</td>
<td>resizebox</td>
</tr>
<tr>
<td>resize</td>
<td>scroll</td>
<td>shape</td>
<td>subwindow</td>
<td>sudoku</td>
<td>symbols</td>
</tr>
<tr>
<td>tabs</td>
<td>threads</td>
<td>tile</td>
<td>tiled_image</td>
<td>unittests</td>
<td>utf8</td>
</tr>
<tr>
<td>valuatorts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.21.1.1 adjuster
adjuster shows a nifty little widget for quickly setting values in a great range.

1.21.1.2 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.

1.21.1.3 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...

1.21.1.4 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.

1.21.1.5 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.

1.21.1.6 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.

1.21.1.7 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.

1.21.1.8 button

The button test is a simple demo of push-buttons and callbacks.

1.21.1.9 buttons

buttons shows a sample of FLTK button types.

1.21.1.10 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.

1.21.1.11 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.
1.21.1.12 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.

1.21.1.13 color_chooser

The color_chooser gives a short demo of FLTK's palette based color chooser and of the RGB based color wheel.

1.21.1.14 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.

1.21.1.15 CubeView

CubeView shows how to create a UI containing OpenGL with Fluid.

1.21.1.16 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).

1.21.1.17 curve

curve draws a nice Bezier curve into a custom widget. The points option for splines is not supported on all platforms.

1.21.1.18 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.

1.21.1.19 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().

1.21.1.20 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.
1.21 Example Source Code

1.21.1.21 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.

1.21.1.22 fast_slow

`fast_slow` shows how an application can use the `Fl_Widget::when()` setting to receive different kinds of callbacks.

1.21.1.23 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.

1.21.1.24 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.

1.21.1.25 forms

`forms` is an XForms program with very few changes. Search for "fltk" to find all changes necessary to port to fltk. This demo shows the different boxtypes. Note that some boxtypes are not appropriate for some objects.

1.21.1.26 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.

1.21.1.27 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.

1.21.1.28 gl_overlay

`gl_overlay` shows OpenGL overlay plane rendering. If no hardware overlay plane is available, FLTK will simulate it for you.

1.21.1.29 glpuzzle

The `glpuzzle` test shows how most Glut source code compiles easily under FLTK.

1.21.1.30 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.
help 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.

iconize demonstrates the effect of the window functions hide(), iconize(), and show().

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).

inactive tests the correct rendering of inactive widgets. To see the inactive version of images, you can check out the pixmap or image test.

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().

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.

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.

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.

Advanced line drawing can be tested with line_style. Not all platforms support all line styles.

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.
1.21.1.41 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.

1.21.1.42 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.

1.21.1.43 message

message pops up a few of FLTK's standard message boxes.

1.21.1.44 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.

1.21.1.45 navigation

navigation demonstrates how the text cursor moves from text field to text field when using the arrow keys, tab, and shift-tab.

1.21.1.46 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.

1.21.1.47 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.

1.21.1.48 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.

1.21.1.49 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.

1.21.1.50 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.
1.21.1.51 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.

1.21.1.52 radio

The radio tool was created entirely with fluid. It shows some of the available button types and tests radio button behavior.

1.21.1.53 resizebox

resizebox shows some possible ways of FLTK's automatic resize behavior.

1.21.1.54 resize

The resize demo tests size and position functions with the given window manager.

1.21.1.55 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.

1.21.1.56 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.

1.21.1.57 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.

1.21.1.58 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.

1.21.1.59 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.).

1.21.1.60 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.
1.21.1.61 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.

1.21.1.62 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.

1.21.1.63 tiled_image

The tiled_image demo uses an 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.

1.21.1.64 unittests

unittests exercises all of FLTK's drawing features (e.g., text, lines, circles, images), as well as scrollbars and schemes.

1.21.1.65 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.

1.21.1.66 valuators

valuators shows all of FLTK's nifty widgets to change numeric values.

1.21.1.67 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 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?

1.22.1 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.

1.22.2 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.

1.22.3 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:

[from the top of the FLTK LGPL License section on exceptions]

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 (http://www.fltk.org).

1.22.4 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;
}...
```

...in main():

```c
Fl::add_handler(my_handler);
...
```

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:
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:
void my_callback(Fl_Widget*, void*) {
    if (Fl::event()==FL_SHORTCUT && Fl::event_key()==FL_Escape)
        return; // ignore Escape
    exit(0);
}
It is very common to ask for confirmation before exiting, this can be done with:
void my_callback(Fl_Widget*, void*) {
    if (fl_ask("Are you sure you want to quit?"))
        exit(0);
}
Chapter 2

Todo List

Page Adding and Extending Widgets
- Clarify Fl_Window::damage(uchar) handling - seems confused/wrong? ORing value doesn't match setting behaviour in FL_Widget.h!
- Clarify Fl_Widget::test_shortcut() explanations. Fl_Widget.h says Internal Use only, but subclassing chapter gives details!

Page Drawing Things in FLTK
- add an Fl_Draw_Area_Cb typedef to allow fl_scroll(...) to be doxygenated?

Member Fl_Browser::-::scrollbar_width (int width)
- This method should eventually be removed in 1.4+

Member Fl_Browser::-::scrollbar_width () const
- This method should eventually be removed in 1.4+

Member Fl_Browser::-::sort (int flags=0)
- Add a flag to ignore case

Class Fl_Button
- Refactor the doxygen comments for Fl_Button when() documentation.
- Refactor the doxygen comments for Fl_Button type() documentation.

Class Fl_Chart
- Refactor Fl_Chart::type() information.

Class Fl_Choice
- Refactor the doxygen comments for Fl_Choice changed() documentation.

Class Fl_Counter
- Refactor the doxygen comments for Fl_Counter type() documentation.

Member Fl_Cursor
- enum Fl_Cursor needs maybe an image.

Member Fl_File_Input::errorcolor () const
- Better docs for Fl_File_Input::errorcolor() - is it even used?

Member Fl_Group::::sizes ()
- Should the internal representation of the sizes() array be documented?

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_Input::-::handle_mouse (int, int, int, int keepmark=0)
- Add comment and parameters

Member Fl_Input::-::handletext (int e, int, int, int)
- Add comment and parameters
Member `fl_intptr_t`
typedefs `fl_intptr_t` and `fl,uintptr_t` should be documented.

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-factorig is not in place in this release.

Member `Fl_Labeltype`
The doxygen comments are incomplete, and some labeltypes start with an underscore. Also, there are three external functions undocumented (yet):

- `fl_define_FL_SHADOW_LABEL()`
- `fl_define_FL_ENGRAVED_LABEL()`
- `fl_define_FL_EMBOSSED_LABEL()`

Member `Fl_Menu::add` (const char *, int shortcut, Fl_Callback *, void *, int=0)
Raw integer shortcut needs examples. Dependent on responses to http://fltk.org/newsgroups.php?g=fltk.development+v:10086 and results of STR#2344

Member `fl_old_shortcut` (const char *)
Fix these silly legacy issues in a future release to support more predictable behavior for the modifier keys.

Member `Fl_Preferences::get` (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)
maxSize should receive the number of bytes that were read.

Member `fl_reset_spot` (void)
provide user documentation for `fl_reset_spot` function

Member `Fl_Scroll::bbox` (int &, int &, int &, int &)
The visibility of the scrollbars ought to be checked/calculated outside of the `draw()` method (STR #1895).

Member `fl_set_spot` (int font, int size, int X, int Y, int W, int H, Fl_Window *)
provide user documentation for `fl_set_spot` function

Member `fl_set_status` (int X, int Y, int W, int H)
provide user documentation for `fl_set_status` function

Member `Fl_String`
FIXME: temporary (?) typedef to mark UTF-8 and Unicode conversions

Member `Fl_Text_Display::display_insert` ()
Unicode?

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_line` (int pos, int *lineNum) const
What does this do?

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::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::wrap_mode` (int wrap, int wrap_margin)
  we need new wrap modes to wrap at the window edge and based on pixel width or average character width.

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?
  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?

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)
  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_When`
  doxygen comments for values are incomplete and maybe wrong or unclear

Member `Fl_Widget::argument` () const
  The user data value must be implemented using `intptr_t` or similar to avoid 64-bit machine incompatibilities.

Member `Fl_Widget::argument` (long v)
  The user data value must be implemented using `intptr_t` or similar to avoid 64-bit machine incompatibilities.

Member `Fl_Widget::type` () const
  Explain "simulate RTTI" (currently only used to decide if a widget is a window, i.e. `type() >= FL_WINDOW ?`). Is
  `type()` really used in a way that ensures "Forms compatibility"?

Member `Fl_Window::show` (int argc, char ++argv)
  explain which system parameters are set up.

Member `Fl_Window::show` ()
  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 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_utf8toUtf16()`.
  Work through the code and this documentation to harmonize the [OksiD] and [Fltk2] functions.
  Do we need this info about planes?
Chapter 3

Deprecated List

Member Fl::release ()
Use Fl::grab(0) instead.

Member Fl::set_idle (Fl_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_clip
fl_clip(int, int, int, int) is deprecated and will be removed from future releases. Please use fl_push_clip(int x, int y, int w, int h) instead.

Member Fl_Group::focus (Fl_Widget ∗W)
This is for backwards compatibility only. You should use W->take_focus() instead.

Member Fl_Menu_Item::check ()

Member Fl_Menu_Item::checked () const

Member Fl_Menu_Item::setonly ()
This method is dangerous if radio items are first in the menu. Use Fl_Menu::setonly(Fl_Menu_Item+) instead.

Member Fl_Menu_Item::uncheck ()

Member Fl_Spinner::maxinum () const

Member Fl_Spinner::mininum () const

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
Chapter 4

Topic Index

4.1 Topics

Here is a list of all topics with brief descriptions:

- Callback function typedefs .......................................................... 195
- Windows handling functions ......................................................... 196
- Events handling functions ............................................................ 198
- Selection & Clipboard functions ................................................... 212
- Screen functions ........................................................................... 215
- Color & Font functions .................................................................. 218
- Drawing functions .......................................................................... 227
- Multithreading support functions .................................................. 251
- Safe widget deletion support functions ......................................... 252
- Cairo Support Functions and Classes .............................................. 255
- Unicode and UTF-8 functions ......................................................... 257
- Mac OS X-specific symbols ............................................................. 270
- Common Dialogs classes and functions .......................................... 272
- File names and URI utility functions .............................................. 283
Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Fl_Preferences::Entry .................................................. 291
- Fl ................................................................. 291
- Fl_Cairo_State .......................................................... 397
- FL_CHART_ENTRY ....................................................... 419
- Fl_Device ............................................................... 489
  - Fl_Graphics_Driver .................................................. 647
    - Fl_GDI_Graphics_Driver .......................................... 600
      - Fl_GDI_Printer_Graphics_Driver ............................. 608
    - Fl_PostScript_Graphics_Driver ............................ 1002
    - Fl_Quartz_Graphics_Driver ........................................ 1049
    - Fl_Xlib_Graphics_Driver .................................... 1579
- Fl_Surface_Device .................................................. 1212
- Fl_Clip_Surface ..................................................... 476
- Fl_Display_Device ................................................... 500
- Fl_Image_Surface ................................................... 755
- Fl_Paged_Device ..................................................... 966
  - Fl_PostScript_File_Device ........................................ 995
    - Fl_PostScript_Printer ............................................. 1016
  - Fl_Printer ....................................................... 1034
  - Fl_System_Printer ................................................ 1227
- Fl_End .............................................................. 514
- Fl_File_Chooser ..................................................... 528
- Fl_File_Icon ........................................................ 533
- Fl_FLTK_File_Chooser ................................................ 572
  - Fl_GTK_File_Chooser ............................................. 689
- Fl_Font_Descriptor .................................................. 573
- Fl_Fontdesc .......................................................... 574
- Fl_Gl_Choice .......................................................... 616
- Fl_Glut_Bitmap_Font ................................................ 634
- Fl_Glut_StrokeChar ................................................... 634
- Fl_Glut_StrokeFont ................................................... 634
- Fl_Glut_StrokeStrip ................................................... 634
- Fl_Glut_StrokeVertex ................................................. 635
- Fl_Help_Block ........................................................ 690
- Fl_Help_Dialog ........................................................ 691
- Fl_Help_Font_Stack .................................................... 693
- Fl_Help_Font_Style ..................................................... 693
- Fl_Help_Link ........................................................... 694
- Fl_Help_Target ........................................................ 694
Hierarchical Index

Fl_Image .......................................................... 750
  Fl_Bitmap .................................................. 321
    Fl_XBM_Image ........................................... 1576
  Fl_Pixmap ................................................ 973
    Fl_GIF_Image ........................................... 614
    Fl_XPM_Image ........................................... 1587
  Fl_RGB_Image ............................................... 1095
    Fl_BMP_Image ........................................... 324
    Fl_JPEG_Image .......................................... 810
    Fl_PNG_Image ........................................... 981
    Fl_PNM_Image .......................................... 985
  Fl_Shared_Image .......................................... 1166
  Fl_Tiled_Image ........................................... 1371
  Fl_Label ................................................... 813
  Fl_Mac_App_Menu .......................................... 829
  Fl_Menu_Item ................................................ 869
  Fl_Multi_Label ........................................... 904
  Fl_Native_File Chooser .................................... 923
  Fl_Plugin .................................................. 977
    Fl_Device_Plugin ....................................... 490
  Fl_Preferences ........................................... 1020
    Fl_Plugin_Manager ...................................... 978
  Fl_Scroll::Fl Region_LRTB ................................ 1078
  Fl_Scroll::Fl Region_XYWH ................................ 1078
  Fl_Scroll::Fl Scrollbar_Data ............................. 1144
  Fl_Text_Buffer ........................................... 1281
  Fl_Text_Selection ......................................... 1359
  Fl_Tooltip ................................................ 1390
  Fl_Tree_Item ................................................ 1441
  Fl_Tree_Item_Array ....................................... 1462
  Fl_Tree_Prefs ............................................. 1465
  Fl_Widget ................................................... 1509
  Fl_Box ...................................................... 327
  Fl_Button .................................................. 387
    Fl_Light_Button ......................................... 814
      Fl_Check_Button ...................................... 431
      Fl_Radio_Light_Button ................................ 1064
      Fl_Round_Button ...................................... 1108
        Fl_Radio_Round_Button ................................ 1071
    Fl_Radio_Button ......................................... 1057
    Fl_Repeat_Button ....................................... 1079
    Fl_Return_Button ....................................... 1087
    Fl_Toggle_Button ....................................... 1382
  Fl_Chart ................................................... 409
  Fl_Clock_Output ........................................... 457
    Fl_Clock ................................................ 449
      Fl_Round_Clock ........................................ 1115
  Fl_FormsBitmap ........................................... 574
  Fl_FormsPixmap ............................................ 580
  Fl_FormsText ............................................... 587
  Fl_Free ..................................................... 593
  Fl_Group .................................................. 675
  Fl_Browser ................................................. 365
    Fl_Browser ............................................... 334
      Fl_File_Browser ...................................... 514
      Fl_Hold_Browser ...................................... 707
Hierarchical Index

Fl_Value_Slider ................................................................. 1499
Fl_Hor_Value_Slider ......................................................... 742
Fl_Value_Input ................................................................. 1479
Fl_Value_Output ............................................................... 1490
Fl_Widget_Tracker .............................................................. 1543
Fl_XColor ...................................................................... 1579
Fl_Text_Editor::Key_Binding .................................................. 1589
Fl_Graphics_Driver::matrix .................................................... 1590
Fl_Preferences::Name ......................................................... 1590
Fl_Preferences::Node ......................................................... 1591
Fl_Paged_Device::page_format ............................................... 1592
Fl_Preferences::RootNode ..................................................... 1592
Fl_Scroll::ScrollInfo ....................................................... 1592
Fl_Window::shape_data_type ................................................. 1593
Fl_Text_Display::Style_Table_Entry ......................................... 1594
Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- **Fl_Preferences::Entry**
  
- **Fl**
  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_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 a pre-configured cairo fltk window

- **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

- **Fl_Choice**
  A button that is used to pop up a menu

- **Fl_Clock**
  This widget provides a round analog clock display

- **Fl_Clock_Output**
  This widget can be used to display a program-supplied time

- **Fl_Color_Chooser**
  Standard RGB color chooser

Generated by Doxygen
<table>
<thead>
<tr>
<th>Class Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fl_Copy_Surface</strong></td>
</tr>
<tr>
<td><strong>Fl_Counter</strong></td>
</tr>
<tr>
<td><strong>Fl_Device</strong></td>
</tr>
<tr>
<td><strong>Fl_Device_Plugin</strong></td>
</tr>
<tr>
<td><strong>Fl_Dial</strong></td>
</tr>
<tr>
<td><strong>Fl_Display_Device</strong></td>
</tr>
<tr>
<td><strong>Fl_Double_Window</strong></td>
</tr>
<tr>
<td><strong>Fl_End</strong></td>
</tr>
<tr>
<td><strong>Fl_File_Browser</strong></td>
</tr>
<tr>
<td><strong>Fl_File_Chooser</strong></td>
</tr>
<tr>
<td><strong>Fl_File_Icon</strong></td>
</tr>
<tr>
<td><strong>Fl_File_Input</strong></td>
</tr>
<tr>
<td><strong>Fl_Fill_Dial</strong></td>
</tr>
<tr>
<td><strong>Fl_Fill_Slider</strong></td>
</tr>
<tr>
<td><strong>Fl_Float_Input</strong></td>
</tr>
<tr>
<td><strong>Fl_FLTK_File_Chooser</strong></td>
</tr>
<tr>
<td><strong>Fl_Font_Descriptor</strong></td>
</tr>
<tr>
<td><strong>Fl_Fontdesc</strong></td>
</tr>
<tr>
<td><strong>Fl_FormsBitmap</strong></td>
</tr>
<tr>
<td><strong>Fl_FormsPixmap</strong></td>
</tr>
<tr>
<td><strong>Fl_FormsText</strong></td>
</tr>
<tr>
<td><strong>Fl_Free</strong></td>
</tr>
<tr>
<td><strong>Fl_GDI_Graphics_Driver</strong></td>
</tr>
<tr>
<td><strong>Fl_GDI_Printer_Graphics_Driver</strong></td>
</tr>
<tr>
<td><strong>Fl_GIF_Image</strong></td>
</tr>
<tr>
<td><strong>Fl_Gl_Choice</strong></td>
</tr>
<tr>
<td><strong>Fl_Gl_Window</strong></td>
</tr>
<tr>
<td><strong>Fl_Glut_Bitmap_Font</strong></td>
</tr>
<tr>
<td><strong>Fl_Glut_StrokeChar</strong></td>
</tr>
<tr>
<td><strong>Fl_Glut_StrokeFont</strong></td>
</tr>
<tr>
<td><strong>Fl_Glut_StrokeStrip</strong></td>
</tr>
<tr>
<td><strong>Fl_Glut_StrokeVertex</strong></td>
</tr>
</tbody>
</table>
6.1 Class List

Fl_Glut_Window
GLUT is emulated using this window class and these static variables (plus several more static
variables hidden in glut_compatibility.cxx):

Fl_Graphics_Driver
A virtual class subclassed for each graphics driver FLTK uses

Fl_Group
FLTK container widget

Fl_GTK_Filechooser

Fl_Help_Block

Fl_Help_Dialog
Displays a standard help dialog window using the Fl_Help_View widget

Fl_Help_Font_Stack

Fl_Help_Font_Style
Fl_Help_View font stack element definition

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

Fl_Hor_Nice_Slider

Fl_Hor_Slider
Horizontal Slider class

Fl_Hor_Value_Slider

Fl_Image
Base class for image caching and drawing

Fl_Image_Surface
Directs all graphics requests to an Fl_Image

Fl_Input
This is the FLTK text input widget

Fl_Input_
This class provides a low-overhead text input field

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
0xaef)

Fl_JPEG_Image
Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Inter-
change 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

Fl_Mac_App_Menu
Mac OS-specific class allowing to customize and localize the application menu

Fl_Menu
Base class of all widgets that have a menu in FLTK

Fl_Menu_Bar
This widget provides a standard menubar interface

Fl_Menu_Item
This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array
of Fl_Menu_Item objects
The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

Fl_Menu_Window
Window type used for menus.

Fl_Multi_Browser
Subclass of Fl_Browser which lets the user select any set of the lines.

Fl_Multi_Label
This struct allows multiple labels to be added to objects that might normally have only one label.

Fl_Multiline_Input
This input field displays 'n' characters as new lines rather than \^J, and accepts the Return, Tab, and up and down arrow keys.

Fl_Multiline_Output
This widget is a subclass of Fl_Output that displays multiple lines of text.

Fl_Native_File_Chooser
This class lets an FLTK application easily and consistently access the operating system's native file chooser.

Fl_Nice_Slider

Fl_Output
This widget displays a piece of text.

Fl_Overlay_Window
This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image.

Fl_Pack
This widget was designed to add the functionality of compressing and aligning widgets.

Fl_Paged_Device
Represents page-structured drawing surfaces.

Fl_Pixmap
Supports caching and drawing of colormap ( pixmap) images, including transparency.

Fl_Plugin
Fl_Plugin allows link-time and run-time integration of binary modules.

Fl_Plugin_Manager
Fl_Plugin_Manager manages link-time and run-time plugin binaries.

Fl_PNG_Image
Supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

Fl_PNM_Image
Supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.

Fl_Positioner
This class is provided for Forms compatibility.

Fl_PostScript_File_Device
To send graphical output to a PostScript file.

Fl_PostScript_Graphics_Driver
PostScript graphical backend.

Fl_PostScript_Printer
Print support under Unix/Linux.

Fl_Preferences
Fl_Preferences provides methods to store user settings between application starts.

Fl_Printer
OS-independent print support.

Fl_Progress
Displays a progress bar for the user.

Fl_Quartz_Graphics_Driver
The Mac OS X-specific graphics class.

Fl_Radio_Button
Fl_Radio_Light_Button
Fl_Radio_Round_Button

Fl_Scroll::Fl_Region_LRTB
A local struct to manage a region defined by left/right/top/bottom.
6.1 Class List

Fl_Scroll::Fl_Region_XYWH
A local struct to manage a region defined by xywh ........................................ 1078

Fl_Repeate_Button
The Fl_Repete_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 .................. 1079

Fl_Return_Button
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 .............................................................. 1087

Fl_RGB_Image
Supports caching and drawing of full-color images with 1 to 4 channels of color information .. 1095

Fl_Roller
"dolly" control commonly used to move 3D objects ........................................... 1100

Fl_Round_Button
Buttons generate callbacks when they are clicked by the user ............................. 1108

Fl_Round_Clock
A clock widget of type FL_ROUND_CLOCK ....................................................... 1115

Fl_Scroll
This container widget lets you maneuver around a set of widgets much larger than your window 1122

Fl_Scrollbar
Displays a slider with arrow buttons at the ends of the scrollbar ......................... 1134

Fl_Scroll::Fl_Scrollbar_Data
A local struct to manage a scrollbar's xywh region and tab values .................... 1144

Fl_Secret_Input
Subclass of Fl_Input that displays its input as a string of placeholders ................ 1144

Fl_Select_Browser
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 ........................................................................... 1153

Fl_Shared_Image
This class supports caching, loading, scaling, and drawing of image files ............... 1166

Fl_Simple_Counter
This widget creates a counter with only 2 arrow buttons .................................... 1174

Fl_Single_Window
This is the same as Fl_Window ........................................................................... 1182

Fl_Slider
Sliding knob inside a box .................................................................................... 1192

Fl_Spinner
This widget is a combination of the input widget and repeat buttons .................... 1202

Fl_Surface_Device
A drawing surface that's susceptible to receive graphical output ......................... 1212

Fl_Sys_Menu_Bar
A class to create, modify and delete menus that appear on Mac OS X in the menu bar at the top
of the screen ................................................................................................... 1214

Fl_System_Printer
Print support under MSWindows and Mac OS ..................................................... 1227

Fl_Table
A table of widgets or other content ..................................................................... 1233

Fl_Table_Row
A table with row selection capabilities ............................................................. 1255

Fl_Tabs
"file card tabs" interface that allows you to put lots and lots of buttons and switches in a panel, as
popularized by many toolkits ........................................................................ 1269

Fl_Text_Buffer
This class manages Unicode text displayed in one or more Fl_Text_Display widgets .... 1281

Fl_Text_Display
Rich text display widget ..................................................................................... 1296

Fl_Text_Editor
This is the FLTK text editor widget ..................................................................... 1337
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_Text_Selection</td>
<td>This is an internal class for FL_Text_Buffer to manage text selections.</td>
<td>1359</td>
</tr>
<tr>
<td>FL_Tile</td>
<td>Lets you resize its children by dragging the border between them.</td>
<td>1362</td>
</tr>
<tr>
<td>FL_Tiled_Image</td>
<td>This class supports tiling of images over a specified area.</td>
<td>1371</td>
</tr>
<tr>
<td>FL_Timer</td>
<td>This is provided only to emulate the Forms Timer widget.</td>
<td>1375</td>
</tr>
<tr>
<td>FL_Toggle_Button</td>
<td>The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off.</td>
<td>1382</td>
</tr>
<tr>
<td>FL_Tooltip</td>
<td>Tooltip support for all FLTK widgets.</td>
<td>1390</td>
</tr>
<tr>
<td>FL_Tree</td>
<td>Tree widget.</td>
<td>1394</td>
</tr>
<tr>
<td>FL_Tree_Item</td>
<td>Tree widget item.</td>
<td>1441</td>
</tr>
<tr>
<td>FL_Tree_Item_Array</td>
<td>Manages an array of FL_Tree_Item pointers.</td>
<td>1462</td>
</tr>
<tr>
<td>FL_Tree_Prefs</td>
<td>Tree widget's preferences.</td>
<td>1465</td>
</tr>
<tr>
<td>FL_Valuator</td>
<td>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.</td>
<td>1470</td>
</tr>
<tr>
<td>FL_Value_Input</td>
<td>Displays a numeric value.</td>
<td>1479</td>
</tr>
<tr>
<td>FL_Value_Output</td>
<td>Displays a floating point value.</td>
<td>1490</td>
</tr>
<tr>
<td>FL_Value_Slider</td>
<td>FL_Slider widget with a box displaying the current value.</td>
<td>1499</td>
</tr>
<tr>
<td>FL_Widget</td>
<td>FL_Widget is the base class for all widgets in FLTK.</td>
<td>1509</td>
</tr>
<tr>
<td>FL_Widget_Tracker</td>
<td>This class should be used to control safe widget deletion.</td>
<td>1543</td>
</tr>
<tr>
<td>FL_Window</td>
<td>This widget produces an actual window.</td>
<td>1544</td>
</tr>
<tr>
<td>FL_Wizard</td>
<td>This widget is based off the FL_Tabs widget, but instead of displaying tabs it only changes &quot;tabs&quot; under program control.</td>
<td>1568</td>
</tr>
<tr>
<td>FL_XBM_Image</td>
<td>Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.</td>
<td>1576</td>
</tr>
<tr>
<td>FL_XColor</td>
<td></td>
<td>1579</td>
</tr>
<tr>
<td>FL_Xlib_Graphics_Driver</td>
<td>The Xlib-specific graphics class.</td>
<td>1579</td>
</tr>
<tr>
<td>FL_XPM_Image</td>
<td>Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.</td>
<td>1587</td>
</tr>
<tr>
<td>FL_Text_Editor::Key_Binding</td>
<td>Simple linked list item associating a key/state to a function.</td>
<td>1589</td>
</tr>
<tr>
<td>FL_Graphics_Driver::matrix</td>
<td>A 2D coordinate transformation matrix.</td>
<td>1590</td>
</tr>
<tr>
<td>FL_Preferences::Name</td>
<td>'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.</td>
<td>1590</td>
</tr>
<tr>
<td>FL_Preferences::Node</td>
<td></td>
<td>1591</td>
</tr>
<tr>
<td>FL_Paged_Device::page_format</td>
<td>Width, height and name of a page format.</td>
<td>1592</td>
</tr>
<tr>
<td>FL_Preferences::RootNode</td>
<td></td>
<td>1592</td>
</tr>
</tbody>
</table>
6.1 Class List

**Fl::Scroll::ScrollInfo**
Structure to manage scrollbar and widget interior sizes ........................................ 1592

**Fl::Window::shape_data_type**
Data supporting a non-rectangular window shape .................................................... 1593

**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 ................................................................. 1594
# Chapter 7

## File Index

### 7.1 File List

Here is a list of all documented files with brief descriptions:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abi-version.h</td>
<td></td>
<td>1595</td>
</tr>
<tr>
<td>dirent.h</td>
<td></td>
<td>1595</td>
</tr>
<tr>
<td>Enumerations.H</td>
<td>This file contains type definitions and general enumerations</td>
<td>1595</td>
</tr>
<tr>
<td>filename.H</td>
<td>File names and URI utility functions</td>
<td>1622</td>
</tr>
<tr>
<td>Fl.H</td>
<td>Fl static class</td>
<td>1625</td>
</tr>
<tr>
<td>Fl_Adjuster.H</td>
<td></td>
<td>1632</td>
</tr>
<tr>
<td>fl_ask.H</td>
<td>API for common dialogs</td>
<td>1632</td>
</tr>
<tr>
<td>Fl_Bitmap.H</td>
<td></td>
<td>1635</td>
</tr>
<tr>
<td>Fl_BMP_Img.H</td>
<td></td>
<td>1636</td>
</tr>
<tr>
<td>Fl_Box.H</td>
<td></td>
<td>1636</td>
</tr>
<tr>
<td>Fl_Browser.H</td>
<td></td>
<td>1637</td>
</tr>
<tr>
<td>Fl_Browser_.H</td>
<td></td>
<td>1639</td>
</tr>
<tr>
<td>Fl_Button.H</td>
<td></td>
<td>1641</td>
</tr>
<tr>
<td>Fl_Cairo.H</td>
<td></td>
<td>1642</td>
</tr>
<tr>
<td>Fl_Cairo_Window.H</td>
<td></td>
<td>1642</td>
</tr>
<tr>
<td>Fl_Chart.H</td>
<td></td>
<td>1643</td>
</tr>
<tr>
<td>Fl_Check_Browser.H</td>
<td></td>
<td>1644</td>
</tr>
<tr>
<td>Fl_Check_Button.H</td>
<td></td>
<td>1646</td>
</tr>
<tr>
<td>FlChoice.H</td>
<td></td>
<td>1646</td>
</tr>
<tr>
<td>Fl_Clock.H</td>
<td></td>
<td>1647</td>
</tr>
<tr>
<td>Fl_Color_Chooser.H</td>
<td>Fl_Color_Chooser widget</td>
<td>1648</td>
</tr>
<tr>
<td>Fl_Copy_Surface.H</td>
<td></td>
<td>1649</td>
</tr>
<tr>
<td>Fl_Counter.H</td>
<td></td>
<td>1651</td>
</tr>
<tr>
<td>Fl.Dial.H</td>
<td></td>
<td>1658</td>
</tr>
<tr>
<td>Fl_Dbl_Window.H</td>
<td></td>
<td>1659</td>
</tr>
<tr>
<td>fl_draw.H</td>
<td>Utility header to pull drawing functions together</td>
<td>1659</td>
</tr>
<tr>
<td>Fl_Export.H</td>
<td></td>
<td>1667</td>
</tr>
<tr>
<td>Fl_File_Browser.H</td>
<td></td>
<td>1668</td>
</tr>
<tr>
<td>Fl_File_Chooser.H</td>
<td></td>
<td>1669</td>
</tr>
<tr>
<td>Fl_File_Icon.H</td>
<td></td>
<td>1671</td>
</tr>
<tr>
<td>Fl_File_Input.H</td>
<td></td>
<td>1673</td>
</tr>
</tbody>
</table>

Generated by Doxygen
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Line Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Fill_Dial.H</td>
<td>1673</td>
</tr>
<tr>
<td>Fl_Fill_Slider.H</td>
<td>1674</td>
</tr>
<tr>
<td>Fl_Float_Input.H</td>
<td>1674</td>
</tr>
<tr>
<td>Fl_FormsBitmap.H</td>
<td>1675</td>
</tr>
<tr>
<td>Fl_FormsPixmap.H</td>
<td>1675</td>
</tr>
<tr>
<td>Fl_Free.H</td>
<td>1676</td>
</tr>
<tr>
<td>Fl_GIF_Image.H</td>
<td>1677</td>
</tr>
<tr>
<td>Fl_Gi_Window.H</td>
<td>1677</td>
</tr>
<tr>
<td>Fl_Group.H</td>
<td>1679</td>
</tr>
<tr>
<td>Fl_Help_Dialog.H</td>
<td>1680</td>
</tr>
<tr>
<td>Fl_Help_View.H</td>
<td>1681</td>
</tr>
<tr>
<td>Fl_Hold_Browser.H</td>
<td>1684</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider.H</td>
<td>1684</td>
</tr>
<tr>
<td>Fl_Hor_Nice_Slider.H</td>
<td>1685</td>
</tr>
<tr>
<td>Fl_Hor_Slider.H</td>
<td>1685</td>
</tr>
<tr>
<td>Fl_Hor_Value_Slider.H</td>
<td>1686</td>
</tr>
<tr>
<td>Fl_Image.H</td>
<td>1686</td>
</tr>
<tr>
<td>Fl_Image_Surface.H</td>
<td>1689</td>
</tr>
<tr>
<td>Fl_Input.H</td>
<td>1690</td>
</tr>
<tr>
<td>Fl_Input_.H</td>
<td>1691</td>
</tr>
<tr>
<td>Fl_Input_Choice.H</td>
<td>1694</td>
</tr>
<tr>
<td>Fl_Int_Input.H</td>
<td>1695</td>
</tr>
<tr>
<td>Fl_JPEG_Image.H</td>
<td>1696</td>
</tr>
<tr>
<td>Fl_Light_Button.H</td>
<td>1696</td>
</tr>
<tr>
<td>Fl_Line_Dial.H</td>
<td>1697</td>
</tr>
<tr>
<td>Fl_Menu.H</td>
<td>1697</td>
</tr>
<tr>
<td>Fl_Menu_.H</td>
<td>1698</td>
</tr>
<tr>
<td>Fl_Menu_Bar.H</td>
<td>1699</td>
</tr>
<tr>
<td>Fl_Menu_Button.H</td>
<td>1700</td>
</tr>
<tr>
<td>Fl_Menu_Item.H</td>
<td>1700</td>
</tr>
<tr>
<td>Fl_Menu_Window.H</td>
<td>1703</td>
</tr>
<tr>
<td>Fl_message.H</td>
<td>1704</td>
</tr>
<tr>
<td>Fl_Multi_Browser.H</td>
<td>1704</td>
</tr>
<tr>
<td>Fl_Multi_Label.H</td>
<td>1705</td>
</tr>
<tr>
<td>Fl_Multiline_Input.H</td>
<td>1705</td>
</tr>
<tr>
<td>Fl_Multiline_Output.H</td>
<td>1706</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser.H</td>
<td>1706</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser.widget</td>
<td>1706</td>
</tr>
<tr>
<td>Fl_Nice_Slider.H</td>
<td>1710</td>
</tr>
<tr>
<td>Fl_Object.H</td>
<td>1710</td>
</tr>
<tr>
<td>Fl_Output.H</td>
<td>1711</td>
</tr>
<tr>
<td>Fl_Overlay_Window.H</td>
<td>1711</td>
</tr>
<tr>
<td>Fl_Pack.H</td>
<td>1712</td>
</tr>
<tr>
<td>Fl_Paged_Device.H</td>
<td>1713</td>
</tr>
<tr>
<td>Fl_Pixmap.H</td>
<td>1714</td>
</tr>
<tr>
<td>Fl_Plugin.H</td>
<td>1716</td>
</tr>
<tr>
<td>Fl_PNG_Image.H</td>
<td>1716</td>
</tr>
<tr>
<td>Fl_PNM_Image.H</td>
<td>1717</td>
</tr>
<tr>
<td>Fl_Positioner.H</td>
<td>1717</td>
</tr>
<tr>
<td>Fl_PostScript.H</td>
<td>1718</td>
</tr>
<tr>
<td>Fl_Preferences.H</td>
<td>1721</td>
</tr>
<tr>
<td>Fl_Printer.H</td>
<td>1724</td>
</tr>
<tr>
<td>Fl_Progress.H</td>
<td>1726</td>
</tr>
<tr>
<td>Fl_Radio_Button.H</td>
<td>1727</td>
</tr>
</tbody>
</table>

Declaration of classes Fl_Printer, Fl_System_Printer and Fl_PostScript_Printer

Declaration of classes Fl_PostScript_Graphics_Driver, Fl_PostScript_File_Device

Declaration of class Fl_Paged_Device

Fl_Native_File_Chooser widget
<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Radio_Light_Button.H</td>
<td>1727</td>
</tr>
<tr>
<td>Fl_Radio_Round_Button.H</td>
<td>1728</td>
</tr>
<tr>
<td>Fl_Repeat_Button.H</td>
<td>1728</td>
</tr>
<tr>
<td>Fl_Return_Button.H</td>
<td>1729</td>
</tr>
<tr>
<td>Fl_RGB_Image.H</td>
<td>1729</td>
</tr>
<tr>
<td>Fl_Roller.H</td>
<td>1730</td>
</tr>
<tr>
<td>Fl_Round_Button.H</td>
<td>1730</td>
</tr>
<tr>
<td>Fl_Round_Clock.H</td>
<td>1731</td>
</tr>
<tr>
<td>Fl_Scroll.H</td>
<td>1731</td>
</tr>
<tr>
<td>Fl_Scrollbar.H</td>
<td>1733</td>
</tr>
<tr>
<td>Fl_Secret_Input.H</td>
<td>1733</td>
</tr>
<tr>
<td>Fl_Select_Browser.H</td>
<td>1734</td>
</tr>
<tr>
<td>Fl_Shared_Image.H</td>
<td>1734</td>
</tr>
<tr>
<td>fl_show_colormap.H</td>
<td>1734</td>
</tr>
<tr>
<td>fl_show_input.H</td>
<td>1736</td>
</tr>
<tr>
<td>Fl_Simple.Counter.H</td>
<td>1737</td>
</tr>
<tr>
<td>Fl_Single_Window.H</td>
<td>1737</td>
</tr>
<tr>
<td>Fl_Slider.H</td>
<td>1738</td>
</tr>
<tr>
<td>Fl_Spinner.H</td>
<td>1739</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar.H</td>
<td>1741</td>
</tr>
<tr>
<td>Fl_Table.H</td>
<td>1743</td>
</tr>
<tr>
<td>Fl_Table_Row.H</td>
<td>1749</td>
</tr>
<tr>
<td>Fl_Tabs.H</td>
<td>1751</td>
</tr>
<tr>
<td>Fl_Text_Buffer.H</td>
<td>1751</td>
</tr>
<tr>
<td>Fl_Text_Display.H</td>
<td>1755</td>
</tr>
<tr>
<td>Fl_Text_Editor.H</td>
<td>1759</td>
</tr>
<tr>
<td>Fl_Tile.H</td>
<td>1761</td>
</tr>
<tr>
<td>Fl_Tiled_Image.H</td>
<td>1761</td>
</tr>
<tr>
<td>Fl_Timer.H</td>
<td>1762</td>
</tr>
<tr>
<td>Fl_Toggle/Button.H</td>
<td>1763</td>
</tr>
<tr>
<td>Fl_Toggle_Light_Button.H</td>
<td>1763</td>
</tr>
<tr>
<td>Fl_Toggle_Round_Button.H</td>
<td>1764</td>
</tr>
<tr>
<td>Fl_Tooltip.H</td>
<td>1764</td>
</tr>
<tr>
<td>Fl_Tree.H</td>
<td>1765</td>
</tr>
<tr>
<td>Fl_Tree_Item.H</td>
<td>1769</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array.H</td>
<td>1774</td>
</tr>
<tr>
<td>Fl_Tree_Prefs.H</td>
<td>1774</td>
</tr>
<tr>
<td>Fl_types.h</td>
<td>1776</td>
</tr>
<tr>
<td>fl_utf8.h</td>
<td>1781</td>
</tr>
<tr>
<td>Fl_Valuator.H</td>
<td>1787</td>
</tr>
<tr>
<td>Fl_Value_Input.H</td>
<td>1788</td>
</tr>
<tr>
<td>Fl_Value_Output.H</td>
<td>1789</td>
</tr>
<tr>
<td>Fl_Value_Slider.H</td>
<td>1790</td>
</tr>
<tr>
<td>Fl_Widget.H</td>
<td>1790</td>
</tr>
<tr>
<td>Fl_Widget, Fl_Label classes</td>
<td>1790</td>
</tr>
<tr>
<td>Fl_Window.H</td>
<td>1795</td>
</tr>
<tr>
<td>Fl_Wizard.H</td>
<td>1799</td>
</tr>
<tr>
<td>Fl_XBM_Image.H</td>
<td>1800</td>
</tr>
</tbody>
</table>

The `fl_show_colormap()` function hides the implementation classes used to provide the popup window and color selection mechanism.
Fl_XPM_Image.H ................................................................. 1800
forms.H ................................................................................. 1801
gl.h
This file defines wrapper functions for OpenGL in FLTK ........... 1810
gl2opengl.h ................................................................. 1814
gl_draw.H .............................................................................. 1815
glu.h ................................................................................. 1815
glut.H ................................................................................. 1816
mac.H
Mac OS X-specific symbols ................................................ 1821
math.h ................................................................................. 1826
names.h ................................................................................. 1826
platform.H ............................................................................. 1827
win32.H ................................................................................. 1828
x.H .................................................................................... 1830
cgdebug.h ............................................................................ 1832
fastarrow.h ........................................................................... 1834
fl_arc.cxx
Utility functions for drawing arcs and circles ....................... 1835
fl_arci.cxx
Utility functions for drawing circles using integers ............... 1835
fl_ask.cxx
Utility Functions for Common Dialogs ................................. 1835
fl_boxtype.cxx
Drawing code for common box types .................................. 1836
fl_cmap.h ............................................................................... 1839
fl_color.cxx
Color handling ................................................................. 1842
fl_compose.cxx
Utility functions to support text input ............................... 1843
fl_curve.cxx
Utility for drawing Bezier curves, adding the points to the current fl_begin/fl_vertex/fl_end path 1843
fl_dnd_x.cxx ................................................................. 1843
Fl_Double_Window.cxx
Fl_Double_Window implementation ....................................... 1846
Fl_Font.H ............................................................................... 1846
fl_font_x.cxx ....................................................................... 1847
Fl_Gl_Choice.H ...................................................................... 1851
fl_line_style.cxx
Line style drawing utility hiding different platforms .............. 1853
Fl_Native_File_Chooser_common.cxx .................................. 1853
Fl_Native_File_Chooser_FLTK.cxx ....................................... 1854
Fl_Native_File_Chooser_GTK.cxx ......................................... 1859
Fl_Paged_Device.cxx
Implementation of class Fl_Paged_Device ............................. 1867
fl_rect.cxx
Drawing and clipping routines for rectangles ...................... 1868
fl_set_fonts_x.cxx ............................................................. 1868
fl_vertex.cxx
Portable drawing code for drawing arbitrary shapes with simple 2D transformations 1872
Fl_XColor.H ........................................................................... 1872
flstring.h .............................................................................. 1873
freeglut_teapot_data.h ......................................................... 1874
mediumarrow.h ..................................................................... 1876
print_panel.h ....................................................................... 1876
scandir_posix.c .................................................................. 1876
slowarrow.h ........................................................................ 1879
Xutf8.h ................................................................................. 1879
<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>case.h</td>
<td>1881</td>
</tr>
<tr>
<td>dingbats_.h</td>
<td>1902</td>
</tr>
<tr>
<td>spacing.h</td>
<td>1908</td>
</tr>
<tr>
<td>symbol_.h</td>
<td>1931</td>
</tr>
<tr>
<td>imKStoUCS.c</td>
<td>1944</td>
</tr>
<tr>
<td>armsci_8.h</td>
<td>1948</td>
</tr>
<tr>
<td>ascii.h</td>
<td>1949</td>
</tr>
<tr>
<td>big5.h</td>
<td>1949</td>
</tr>
<tr>
<td>big5_emacs.h</td>
<td>1997</td>
</tr>
<tr>
<td>cp1133.h</td>
<td>1999</td>
</tr>
<tr>
<td>cp1251.h</td>
<td>2000</td>
</tr>
<tr>
<td>cp1255.h</td>
<td>2001</td>
</tr>
<tr>
<td>cp1256.h</td>
<td>2003</td>
</tr>
<tr>
<td>cp936ext.h</td>
<td>2004</td>
</tr>
<tr>
<td>gb2312.h</td>
<td>2076</td>
</tr>
<tr>
<td>georgian_academy.h</td>
<td>2106</td>
</tr>
<tr>
<td>georgian_ps.h</td>
<td>2107</td>
</tr>
<tr>
<td>iso8859_1.h</td>
<td>2108</td>
</tr>
<tr>
<td>iso8859_10.h</td>
<td>2108</td>
</tr>
<tr>
<td>iso8859_11.h</td>
<td>2110</td>
</tr>
<tr>
<td>iso8859_13.h</td>
<td>2111</td>
</tr>
<tr>
<td>iso8859_14.h</td>
<td>2112</td>
</tr>
<tr>
<td>iso8859_15.h</td>
<td>2113</td>
</tr>
<tr>
<td>iso8859_16.h</td>
<td>2114</td>
</tr>
<tr>
<td>iso8859_2.h</td>
<td>2115</td>
</tr>
<tr>
<td>iso8859_3.h</td>
<td>2116</td>
</tr>
<tr>
<td>iso8859_4.h</td>
<td>2118</td>
</tr>
<tr>
<td>iso8859_5.h</td>
<td>2119</td>
</tr>
<tr>
<td>iso8859_6.h</td>
<td>2120</td>
</tr>
<tr>
<td>iso8859_7.h</td>
<td>2121</td>
</tr>
<tr>
<td>iso8859_8.h</td>
<td>2122</td>
</tr>
<tr>
<td>iso8859_9.h</td>
<td>2123</td>
</tr>
<tr>
<td>iso8859_9e.h</td>
<td>2124</td>
</tr>
<tr>
<td>jisx0201.h</td>
<td>2125</td>
</tr>
<tr>
<td>jisx0208.h</td>
<td>2126</td>
</tr>
<tr>
<td>jisx0212.h</td>
<td>2153</td>
</tr>
<tr>
<td>koi8_c.h</td>
<td>2178</td>
</tr>
<tr>
<td>koi8_r.h</td>
<td>2180</td>
</tr>
<tr>
<td>koi8_u.h</td>
<td>2181</td>
</tr>
<tr>
<td>ksc5601.h</td>
<td>2183</td>
</tr>
<tr>
<td>mulelao.h</td>
<td>2218</td>
</tr>
<tr>
<td>tatar_cyr.h</td>
<td>2219</td>
</tr>
<tr>
<td>tcvn.h</td>
<td>2220</td>
</tr>
<tr>
<td>tis620.h</td>
<td>2222</td>
</tr>
<tr>
<td>ucs2be.h</td>
<td>2223</td>
</tr>
<tr>
<td>utf8.h</td>
<td>2223</td>
</tr>
<tr>
<td>viscii.h</td>
<td>2224</td>
</tr>
<tr>
<td>mk_wcwidth.c</td>
<td>2226</td>
</tr>
<tr>
<td>ucs2fontmap.c</td>
<td>2230</td>
</tr>
<tr>
<td>utf8Utils.c</td>
<td>2234</td>
</tr>
<tr>
<td>Ximint.h</td>
<td>2237</td>
</tr>
<tr>
<td>Xlibint.h</td>
<td>2237</td>
</tr>
</tbody>
</table>
Chapter 8

Topic Documentation

8.1 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_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 some timeout callback functions passed as parameters.
### 8.1.1 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

### 8.1.2 Typedef Documentation

#### 8.1.2.1 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)

### 8.2 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.

### 8.2.1 Detailed Description

Windows and standard dialogs handling declared in `<FL/Fl.H>`
8.2 Windows handling functions

8.2.2 Function Documentation

8.2.2.1 default_atclose()

```cpp
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.

8.2.2.2 first_window() [1/2]

```cpp
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.

8.2.2.3 first_window() [2/2]

```cpp
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.

8.2.2.4 grab() [1/2]

```cpp
static Fl_Window * Fl::grab ( ) [inline], [static]
```

Returns the window that currently receives all events. Returns the window that currently receives all events, or NULL if event grabbing is currently OFF.

8.2.2.5 grab() [2/2]

```cpp
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 WIN32 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.

8.2.2.6 modal()

```cpp
static Fl_Window * Fl::modal ( ) [inline], [static]
```

Returns the top-most modal() window currently shown. 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).
8.2.2.7 next_window()

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

| in | window | must be shown and not NULL |

8.2.2.8 set_atclose()

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*)

8.2.3 Variable Documentation

8.2.3.1 atclose

void(* Fl::atclose)(Fl_Window *, void *)=default_atclose [static], [default]

Back compatibility: default window callback handler.

See also

Fl::set_atclose()

8.3 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 int Fl::compose (int &del)
  Any text editing widget should call this for each FL_KEYBOARD event.
  If the user moves the cursor, be sure to call Fl::compose_reset().
- static void Fl::compose_reset ()
  Disables the system input methods facilities.
- static void Fl::disable_im ()
  Enables the system input methods facilities.
- static void Fl::enable_im ()
  Returns the last event that was processed.
8.3 Events handling functions

- 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::testShortcut (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_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.

8.3.1 Detailed Description

Fl class events handling API declared in <FL/FL.H>
8.3.2 Function Documentation

8.3.2.1 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_FULLSCREEN** events sent to a window that enters or leaves fullscreen mode.
- 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 *)`

8.3.2.2 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ha</td>
<td>The event handler function to register</td>
</tr>
<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)`

8.3.2.3 belowmouse() [1/2]

```c
static Fl_Widget * Fl::belowmouse ( ) [inline], [static]
```

Gets the widget that is below the mouse.
See also

\[ \text{belowmouse(Fl_Widget*)} \]

### 8.3.2.4 belowmouse()

`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()).

### 8.3.2.5 compose()

`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.

On the Mac OS platform, text input can involve marked text, that is, temporary text replaced by other text during the input process. This occurs, e.g., 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. It’s the widget’s task to underline marked text. Widgets should also call `void Fl::reset←_marked_text()` when processing FL_UNFOCUS events. Optionally, widgets can also call `void Fl::insertion_point_location(int x, int y, int height)` 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 appear just below the insertion point. If widgets don’t do that, auxiliary windows appear at the widget’s bottom. 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 the feature introduced with Mac OS 10.7 “Lion” where pressing and holding a key on the keyboard opens an accented-character menu window. Though the current implementation returns immediately, future versions may take quite awhile, as they may pop up a window or do other user-interface things to allow characters to be selected.

### 8.3.2.6 compose_reset()

`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.

### 8.3.2.7 disable_im()

`static void Fl::disable_im () [static]`

Disables the system input methods facilities.

See also

\[ \text{enable_im()} \]
8.3.2.8 **enable_im()**

```c
static void Fl::enable_im ( ) [static]
```

Enables the system input methods facilities.  This is the default.

See also

```c
disable_im()
```

8.3.2.9 **event()**

```c
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.

8.3.2.10 **event_button()**

```c
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

- **FL_LEFT_MOUSE**
- **FL_MIDDLE_MOUSE**
- **FL_RIGHT_MOUSE**

See also

```c
Fl::event_buttons()
```

8.3.2.11 **event_button1()**

```c
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()`.

8.3.2.12 **event_button2()**

```c
static int Fl::event_button2 ( ) [inline], [static]
```

Returns non-zero if button 2 is currently held down.  For more details, see `Fl::event_buttons()`.

8.3.2.13 **event_button3()**

```c
static int Fl::event_button3 ( ) [inline], [static]
```

Returns non-zero if button 3 is currently held down.  For more details, see `Fl::event_buttons()`.

8.3.2.14 **event_buttons()**

```c
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.
8.3.2.15 event_clicks() [1/2]

static int Fl::event_clicks ( ) [inline], [static]
Returns non zero if we had a double click event.

Return values

<table>
<thead>
<tr>
<th>Non-zero</th>
<th>if the most recent FL_PUSH or FL_KEYBOARD was a &quot;double click&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>for N clicks. A double click is counted if the same button is pressed again while event_is_click() is true.</td>
</tr>
</tbody>
</table>

8.3.2.16 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()

8.3.2.17 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_Image * when the result of Fl::event_clipboard_type() is Fl::clipboard_image.

8.3.2.18 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.

8.3.2.19 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...
  return 0;
}
```
8.3 Events handling functions

```c
} 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*)

### 8.3.2.20 event_dx()

```c
static int Fl::event_dx () [inline], [static]
```

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event. Right is positive.

### 8.3.2.21 event_dy()

```c
static int Fl::event_dy () [inline], [static]
```

Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event. Down is positive.

### 8.3.2.22 event_inside() [1/2]

```c
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

8.3.2.23  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

8.3.2.24  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.

8.3.2.25  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.

8.3.2.26  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&).
8.3.2.27  event_key() [2/2]

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. The highest possible value is FL_KP_Last so you can range-check to see if something is on the keypad.
- All number 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 WIN32 Fl::get_key(FL_KP_Enter) and Fl::event_key(FL_KP_Enter) do not work.

8.3.2.28  event_length()

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.

8.3.2.29  event_original_key()

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.

8.3.2.30  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.

8.3.2.31 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().

8.3.2.32 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.

8.3.2.33 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().

8.3.2.34 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().

8.3.2.35 focus() [1/2]

static Fl_Widget * Fl::focus () [inline], [static]

Gets the current Fl::focus() widget.

See also

Fl::focus(Fl_Widget*)
8.3 Events handling functions

8.3.2.36  **focus()** [2/2]

```c
void Fl::focus ( Fl_Widget * o ) [static]
```

Sets the widget that will receive FL_KEYBOARD events. 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()).

See also

Fl_Widget::take_focus()

8.3.2.37  **get_key()**

```c
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)

8.3.2.38  **get_mouse()**

```c
static 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.

8.3.2.39  **handle()**

```c
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_() 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_Event_Handler)
Fl::event_dispatch(Fl_Event_Dispatch)
8.3.2.40 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*)`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>the event type (Fl::event_number() is not yet set)</td>
</tr>
<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)`

8.3.2.41 pushed() [1/2]

```c
static Fl_Widget * Fl::pushed ( ) [inline], [static]
```
Gets the widget that is being pushed.

See also

`void pushed(Fl_Widget*)`

8.3.2.42 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()`).

8.3.2.43 remove_handler()

```c
void Fl::remove_handler (  
    Fl_Event_Handler ha ) [static]
```
Removes a previously added event handler.

See also

`Fl::handle(int, Fl_Window*)`

8.3.2.44 remove_system_handler()

```c
void Fl::remove_system_handler (  
    Fl_System_Handler ha ) [static]
```
Removes a previously added system event handler.
8.3 Events handling functions

Parameters

| ha | The event handler function to remove |

See also

- Fl::add_system_handler(Fl_System_Handler)

8.3.2.45 test_shortcut()

```cpp
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.

8.3.3 Variable Documentation

8.3.3.1 fl_eventnames

```cpp
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:**

```cpp
#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]"..
    //...
}
```

8.3.3.2 fl_fontnames

```cpp
const char* const fl_fontnames[]
```

Initial value:

```cpp
= {
    "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:**

```cpp
#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]"..
    //...
}
```
8.4 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.*

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.*

8.4.1 Detailed Description

FLTK global copy/cut/paste functions declared in `<FL/Fl.H>`

8.4.2 Function Documentation

8.4.2.1 `add_clipboard_notify()`

```c
void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data = 0) [static]
```

*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);
    ...
    ...
```
8.4 Selection & Clipboard functions

Note

Some systems require polling to monitor the clipboard and may therefore have some delay in detecting changes.

8.4.2.2 clipboard_contains()

```c
static int Fl::clipboard_contains (const char ∗type) [static]
```

Returns non 0 if the clipboard contains data matching type.
type can be Fl::clipboard_plain_text or Fl::clipboard_image.

8.4.2.3 copy()

```c
static 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.

8.4.2.4 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)

8.4.2.5 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)

8.4.2.6 paste() [2/2]

```c
static 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. While processing the FL_PASTE event:

- if `type` is `Fl::clipboard_plain_text`, the text string from the specified `source` is in `Fl::event_text()` with UTF-8 encoding, and the number of bytes 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 `type` is `Fl::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. Furthermore, starting with FLTK 1.3.4, the image is of type `Fl_RGB_Image` across all platforms. 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.
- MSWindows 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.

8.4.2.7 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.

8.4.2.8 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

- `Fl_Widget* selection_owner(Fl_Widget*)`

8.4.2.9 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.
8.5 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 int Fl::screen_count ()**
  
  Gets the number 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 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 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.

8.5.1 Detailed Description

fl global screen functions declared in `<FL/Fl.H>`

8.5.2 Function Documentation

8.5.2.1 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

<table>
<thead>
<tr>
<th>out</th>
<th>h,v</th>
<th>horizontal and vertical resolution</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>

Generated by Doxygen
8.5.2.2 screen_num() [1/2]

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

| in | x, y | the absolute screen position |

8.5.2.3 screen_num() [2/2]

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

| in | x, y, w, h | the rectangle to search for intersection with |

8.5.2.4 screen_work_area() [1/3]

static void Fl::screen_work_area (  
    int & X,  
    int & Y,  
    int & W,  
    int & H ) [inline], [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)

8.5.2.5 screen_work_area() [2/3]

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.
8.5 Screen functions

Parameters

| out | X,Y,W,H | the work area bounding box |
| in  | mx,my   | the absolute screen position |

8.5.2.6 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) |

See also

```c
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)
```

8.5.2.7 screen_xywh() [1/4]

```c
static void Fl::screen_xywh (  
    int & X,  
    int & Y,  
    int & W,  
    int & H ) [inline], [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)
```

8.5.2.8 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 |
8.5.2.9  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.

**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

```c
void screen_xywh(int &X, int &Y, int &W, int &H, int n)
```

8.5.2.10  screen_xywh() [4/4]

```c
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 MSWindows, Mac OS X, and 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>

See also

```c
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)
```

8.6  Color & Font functions

Fl global color, font functions.

**Functions**

- **Fl_Color fl_color ()**  
  Returns the last fl_color() that was set.
- **void fl_color (Fl_Color c)**  
  Sets 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)**
8.6 Color & Font functions

Sets 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)**
  Returns a color that contrasts with the background color.

- **int fl_descent ()**
  Returns 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.

- **Fl_Font fl_font ()**
  Returns 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.

- **int fl_height ()**
  Returns the recommended minimum line spacing for the current font.

- **FL_EXPORT int fl_height (int font, int size)**
  This function returns the actual height of the specified font and size.

- **Fl_Color fl_inactive (Fl_Color c)**
  Returns the inactive, dimmed version of the given color.

- **FL_EXPORT const char ∗ fl_latin1_to_local (const char ∗ t, int n=-1)**
  Converts text from Windows/X11 latin1 character set to local encoding.

- **FL_EXPORT const char ∗ fl_local_to_latin1 (const char ∗ t, int n=-1)**
  Converts text from local encoding to Windows/X11 latin1 character set.

- **FL_EXPORT const char ∗ fl_local_to_mac_roman (const char ∗ t, int n=-1)**
  Converts text from local encoding to Mac Roman character set.

- **FL_EXPORT const char ∗ fl_mac_roman_to_local (const char ∗ t, int n=-1)**
  Converts text from Mac Roman character set to local encoding.

- **FL_EXPORT Fl_Color fl_show_colormap (Fl_Color oldcol)**
  Pops up a window to let the user pick a colormap entry.

- **Fl_Fontsize fl_size ()**
  Returns the size set by the most recent call to `fl_font()`.

- **FL_EXPORT void fl_text_extents (const char ∗ t, int &dx, int &dy, int &w, int &h)**
  Determines the minimum pixel dimensions of a nul-terminated string.

- **void fl_text_extents (const char ∗ t, int n, int &dx, int &dy, int &w, int &h)**
  Determines the minimum pixel dimensions of a sequence of n characters.

- **FL_EXPORT double fl_width (const char ∗ txt)**
  Returns the typographical width of a nul-terminated string using the current font face and size.

- **double fl_width (const char ∗ txt, int n)**
  Returns the typographical width of a sequence of n characters using the current font face and size.

- **double fl_width (unsigned int c)**
  Returns the typographical width of a single character using the current font face and size.

- **ulong fl_xpixel (Fl_Color i)**
  Returns the X pixel number used to draw the given FLTK color index.

- **ulong fl_xpixel (uchar r, uchar g, uchar b)**
  Returns the X pixel number used to draw the given rgb color.

- **static void Fl::free_color (Fl_Color i, int overlay=0)**
  Frees the specified color from the colormap, if applicable.

- **static unsigned Fl::get_color (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.
220 Topic Documentation

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, unsigned c)

Sets an entry in the fl_color index table.

- static void Fl::set_color (Fl_Color, 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.

8.6.1 Detailed Description

fl global color, font functions.
These functions are declared in <FL/Fl.H> or <FL/fl_draw.H>.

8.6.2 Function Documentation

8.6.2.1 fl_color() [1/3]

Fl_Color fl_color ( ) [inline]
Returns the last fl_color() that was set.
This can be used for state save/restore.

8.6.2.2 fl_color() [2/3]

void fl_color ( Fl_Color c ) [inline]
Sets the color for all subsequent drawing operations.
For colormapped 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

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>color</th>
</tr>
</thead>
</table>

8.6.2.3 fl_color() [3/3]

void fl_color ( uchar r, uchar g, uchar b ) [inline]
Sets 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

<table>
<thead>
<tr>
<th>in</th>
<th>r.g.b</th>
<th>color components</th>
</tr>
</thead>
</table>
8.6 Color & Font functions

8.6.2.4 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

<table>
<thead>
<tr>
<th>in</th>
<th>color1,color2</th>
<th>boundary colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>weight</td>
<td>weighting factor</td>
</tr>
</tbody>
</table>

8.6.2.5 fl_contrast()

Fl_Color fl_contrast (  
    Fl_Color fg,  
    Fl_Color bg )

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.

Parameters

| in  | fg.bg         | foreground and background colors |

Returns

contrasting color

8.6.2.6 fl_font() [1/2]

Fl_Font fl_font ( ) [inline]

Returns the face set by the most recent call to fl_font().  
This can be used to save/restore the font.

8.6.2.7 fl_font() [2/2]

void fl_font (  
    Fl_Font face,  
    Fl_Fontsize fsize ) [inline]

Sets the current font, which is then used in various drawing routines.  
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 pixels and not "points". Lines should be spaced size pixels apart or more.

8.6.2.8 fl_height() [1/2]

int fl_height ( ) [inline]

Returns the recommended minimum line spacing for the current font.  
You can also use the value of size passed to fl_font()
8.6.2.9  fl_height() [2/2]

FL_EXPORT int fl_height (  
    int font,  
    int size )

This function returns the actual height of the specified \texttt{font} and \texttt{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 \texttt{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.

8.6.2.10  fl_latin1_to_local()

FL_EXPORT const char * fl_latin1_to_local (  
    const char * t,  
    int n = -1 )

Converts text from Windows/X11 latin1 character set to local encoding.

Parameters

<table>
<thead>
<tr>
<th>in t</th>
<th>character string (latin1 encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in n</td>
<td>optional number of characters to convert (default is all)</td>
</tr>
</tbody>
</table>

Returns

pointer to internal buffer containing converted characters

8.6.2.11  fl_local_to_latin1()

FL_EXPORT const char * fl_local_to_latin1 (  
    const char * t,  
    int n = -1 )

Converts text from local encoding to Windowx/X11 latin1 character set.

Parameters

<table>
<thead>
<tr>
<th>in t</th>
<th>character string (local encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in n</td>
<td>optional number of characters to convert (default is all)</td>
</tr>
</tbody>
</table>

Returns

pointer to internal buffer containing converted characters

8.6.2.12  fl_local_to_mac_roman()

FL_EXPORT const char * fl_local_to_mac_roman (  
    const char * t,  
    int n = -1 )

Converts text from local encoding to Mac Roman character set.
8.6 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

8.6.2.13 fl_mac_roman_to_local()

FL_EXPORT const char * fl_mac_roman_to_local ( 
    const char * t, 
    int n = -1 )

Converts 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

8.6.2.14 fl_show_colormap()

FL_EXPORT Fl_Color fl_show_colormap ( 
    Fl_Color oldcol )

Pops up a window to let the user pick a colormap entry.

Figure 8.1 fl_show_colormap
Parameters

| in oldcol | color to be highlighted when grid is shown. |

Return values

| FL_Color | value of the chosen colormap entry. |

See also

- Fl_Color_chooser

8.6.2.15 fl_size()

FL_Fontsize fl_size ( ) [inline]
Returns the size set by the most recent call to fl_font(). This can be used to save/restore the font.

8.6.2.16 fl_text_extents() [1/2]

FL_EXPORT void fl_text_extents ( const char * c, int & dx, int & dy, int & w, int & h )
Determines the minimum pixel dimensions of a nul-terminated string.
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.
No FLTK symbol expansion will be performed.

8.6.2.17 fl_text_extents() [2/2]

void fl_text_extents ( const char * t, int n, int & dx, int & dy, int & w, int & h ) [inline]
Determines the minimum pixel dimensions of a sequence of n characters.
See also

- fl_text_extents(const char*, int& dx, int& dy, int& w, int& h)

8.6.2.18 fl_width()

double fl_width ( unsigned int c ) [inline]
Returns 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 ftk window is available when called.
8.6.2.19 fl_xpixel() [1/2]

ulong fl_xpixel (Fl_Color i)

Returns the X pixel number used to draw the given FLTK color index. This is the X pixel that fl_color() would use.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>color index</th>
</tr>
</thead>
</table>

Returns

X pixel number

8.6.2.20 fl_xpixel() [2/2]

ulong fl_xpixel (uchar r, uchar g, uchar b)

Returns the X pixel number used to draw the given rgb color. This is the X pixel that fl_color() would use.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>r,g,b</th>
<th>color components</th>
</tr>
</thead>
</table>

Returns

X pixel number

8.6.2.21 free_color()

void Fl::free_color (Fl_Color i, int overlay = 0) [static]

Frees the specified color from the colormap, if applicable. Free color i if used, and clear mapping table entry. If overlay is non-zero then the color is freed from the overlay colormap.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>color index</th>
</tr>
</thead>
</table>

| in | overlay | 0 for normal, 1 for overlay color |

8.6.2.22 get_color() [1/2]

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.
8.6.2.23  get_color() [2/2]

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 unsigned get_color(Fl_Color c)

8.6.2.24  get_font()

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.

8.6.2.25  get_font_name()

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 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.

8.6.2.26  get_font_sizes()

int Fl::get_font_sizes ( 
    Fl_Font fnum, 
    int *& sizep ) [static]

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.

8.6.2.27  set_color() [1/2]

void Fl::set_color ( 
    Fl_Color i, 
    unsigned c ) [static]

Sets an entry in the fl_color index table. 
Set color mapping table entry i to color c. 
You can set it to any 8-bit RGB color. The color is not allocated until fl_color(i) is used.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>color index</td>
</tr>
<tr>
<td>c</td>
<td>color</td>
</tr>
</tbody>
</table>
8.7 Drawing functions

8.6.2.28 set_color() [2/2]

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.

8.6.2.29 set_font()

void Fl::set_font (  
  Fl_Font fnum,  
  const char * name ) [static]

Changes a face.  
The string pointer is simply stored, the string is not copied, so the string must be in static memory.

8.6.2.30 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 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 WIN32 NULL selects fonts with ISO8859-1 encoding and non-NULL selects all fonts.  
The return value is how many faces are in the table after this is done.

8.7 Drawing functions

FLTK global graphics and GUI drawing functions.

Macros

- #define fl_clip fl_push_clip  
  Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

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

- void Fl_Quartz_Graphics_Driver::copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)  
  see fl_copy_offscreen()  
- FL_EXPORT int fl_add_symbol (const char *name, void(*drawit)(Fl_Color), int scalable)  
  Adds a symbol to the system.  
- void fl_arc (double x, double y, double r, double start, double end)  
  Adds 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** ()
  Starts drawing a complex filled polygon.

- void **fl_begin_line** ()
  Starts drawing a list of lines.

- void **fl_begin_loop** ()
  Starts 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** ()
  Starts drawing a list of points.

- void **fl_begin_polygon** ()
  Starts drawing a convex filled polygon.

- FL_EXPORT char **fl_can_do_alpha_blending** ()
  Checks whether platform supports true alpha blending for RGBA images.

- FL_EXPORT void **fl_chord** (int x, int y, int w, int h, double a1, double a2)
  \textit{fl_chord} declaration is a place holder - the function does not yet exist

- void **fl_circle** (double x, double y, double r)
  \textit{fl_circle()} is equivalent to \textit{fl_arc(x,y,r,0,360)}, but may be faster.

- int **fl_clip_box** (int x, int y, int w, int h, Fl_OFFSCREEN pixmap, int srcx, int srcy)
  Intersects the rectangle with the current clip region and returns the bounding box of the result.

- Fl_Region **fl_clip_region** ()
  Returns the current clipping region.

- void **fl_clip_region** (Fl_Region r)
  Replaces 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.

- FL_EXPORT void **fl_cursor** (Fl_Cursor)
  Sets the cursor for the current window to the specified shape and colors.

- FL_EXPORT void **fl_cursor** (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)

- void **fl_curve** (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
  Adds a series of points on a Bezier 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.

- FL_EXPORT void **fl_draw** (const char *str, int x, int y)
  Draws a null-terminated UTF-8 string starting at the given \(x, y\) location.

- 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)
  Fancy string drawing function which is used to draw all the labels.

- FL_EXPORT void **fl_draw** (const char *str, int x, int y, int w, int h, Fl_Align align, void(*callthis)(const char *, int, int), Fl_IMAGE *img=0, int draw_symbols=1)
  The same as **fl_draw(const char*, int, int, Fl_Align, Fl_IMAGE*, int)** with the addition of the \texttt{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 \(\text{angle}\) degrees counter-clockwise.

- FL_EXPORT void **fl_draw** (int angle, const char *str, int x, int y)
Draws a null-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counterclockwise.

- **FL_EXPORT 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_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)**
  
  Draws 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)**
  
  Draws 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)**
  
  Draws 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)**
  
  Draws a gray-scale image using a callback function to generate image data.

- **FL_EXPORT int fl_draw_pixmap (char const *data, int x, int y, Fl_Color=FL_GRAY)**
  
  Draws XPM image data, with the top-left corner at the given position.

- **FL_EXPORT int fl_draw_pixmap (const char const *cdata, int x, int y, Fl_Color=FL_GRAY)**
  
  Draws XPM image data, with the top-left corner at the given position.

- **FL_EXPORT int fl_draw_symbol (const char *label, int x, int y, int w, int h, Fl_Color)**
  
  Draws the named symbol in the given rectangle using the given color.

- **void fl_end_complex_polygon ()**
  
  Ends complex filled polygon, and draws.

- **void fl_end_line ()**
  
  Ends list of lines, and draws.

- **void fl_end_loop ()**
  
  Ends closed sequence of lines, and draws.

- **void fl_end_offscreen ()**
  
  Quits sending drawing commands to the current offscreen buffer.

- **void fl_end_points ()**
  
  Ends list of points, and draws.

- **void fl_end_polygon ()**
  
  Ends convex filled polygon, and draws.

- **FL_EXPORT 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.

- **FL_EXPORT void fl_frame (const char *s, int x, int y, int w, int h)**
  
  Draws a series of line segments around the given box.

- **FL_EXPORT 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 ()**
  
  Call fl_gap() to separate loops of the path.

- **void fl_line (int x, int y, int x1, int y1)**
  
  Draws a line from (x,y) to (x1,y1)

- **void fl_line (int x, int y, int x1, int y1, int x2, int y2)**
  
  Draws 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)**
  
  Sets how to draw lines (the "pen").

- **void fl_loop (int x, int y, int x1, int y1, int x2, int y2)**
  
  Outlines 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)**
  
  Outlines a 4-sided polygon with lines.

- **FL_EXPORT 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.
• FL_EXPORT int fl_measure_pixmap (char * const data, int &w, int &h)
  Get the dimensions of a pixmap.
• FL_EXPORT 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)
  Concatenates 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?
• FL_EXPORT unsigned int fl_old_shortcut (const char * s)
  Emulation of XForms named shortcuts.
• FL_EXPORT void fl_overlay_clear ()
  Erase a selection rectangle without drawing a new one.
• FL_EXPORT void fl_overlay_rect (int x, int y, int w, int h)
  Draws a selection rectangle, erasing a previous one by XOR’ing it first.
• 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)
  Draws a single pixel at the given coordinates.
• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2)
  Fills a 3-sided polygon.
• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Fills a 4-sided polygon.
• void fl_pop_clip ()
  Restores the previous clip region.
• void fl_pop_matrix ()
  Restores the current transformation matrix from the stack.
• void fl_push_clip (int x, int y, int w, int h)
  Intersects the current clip region with a rectangle and pushes this new region onto the stack.
• void fl_push_matrix ()
  Saves the current transformation matrix on the stack.
• void fl_push_no_clip ()
  Pushes an empty clip region onto the stack so nothing will be clipped.
• FL_EXPORT 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 (int x, int y, int w, int h)
  Draws a 1-pixel border inside the given bounding box.
• void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Draws with passed color a 1-pixel border inside the given bounding box.
• void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Colors with current color a rectangle that exactly fills the given bounding box.
• void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Colors with passed color a rectangle that exactly fills the given bounding box.
• FL_EXPORT void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)
  Colors a rectangle with "exactly" the passed r, g, b color.
• FL_EXPORT void fl_reset_spot (void)
• void fl_rotate (double d)
  Concatenates rotation transformation onto the current one.
• void fl_rtl_draw (const char * str, int n, int x, int y)
  Draws a UTF-8 string of length n bytes right to left starting at the given x, y location.
8.7 Drawing functions

- void fl_scale (double x)
  Concatenates scaling transformation onto the current one.

- void fl_scale (double x, double y)
  Concatenates scaling transformation onto the current one.

- 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)
  Scroll a rectangle and draw the newly exposed portions.

- 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_set_status (int X, int Y, int W, int H)
- FL_EXPORT const char *flShortcut_label (unsigned int shortcut)
  Get a human-readable string from a shortcut value.

- FL_EXPORT const char *flShortcut_label (unsigned int shortcut, const char **eom)
  Get a human-readable string from a shortcut value.

- double fl_transform_dx (double x, double y)
  Transforms distance using current transformation matrix.

- double fl_transform_dy (double x, double y)
  Transforms distance using current transformation matrix.

- double fl_transform_x (double x, double y)
  Transforms coordinate using the current transformation matrix.

- double fl_transform_y (double x, double y)
  Transforms coordinate using the current transformation matrix.

- void fl_transformed_vertex (double xf, double yf)
  Adds coordinate pair to the vertex list without further transformations.

- void fl_translate (double x, double y)
  Concatenates translation transformation onto the current one.

- void fl_vertex (double x, double y)
  Adds a single vertex to the current path.

- void fl_xyline (int x, int y, int x1)
  Draws a horizontal line from (x,y) to (x1,y)

- void fl_xyline (int x, int y, int x1, int y2)
  Draws 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)
  Draws 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)
  Draws a vertical line from (x,y) to (x,y1)

- void fl_yxline (int x, int y, int y1, int x2)
  Draws 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)
  Draws 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

- const int stack_max = 16

8.7.1 Detailed Description

FLTK global graphics and GUI drawing functions. These functions are declared in `<FL/fl_draw.H>`, and in `<FL/x.H>` for offscreen buffer-related ones.
8.7.2 Macro Definition Documentation

8.7.2.1 fl_clip

#define fl_clip fl_push_clip
Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

Parameters

| in       | x, y, w, h | position and size |

Deprecated fl_clip(int, int, int) is deprecated and will be removed from future releases. Please use fl_push_clip(int x, int y, int w, int h) instead.

8.7.3 Enumeration Type Documentation

8.7.3.1 anonymous enum

Anonymous enum

Enumerator

| FL_SOLID   | line style: ____________ |
| FL_DASH    | line style: _ _ _ _ _ _ |
| FL_DOT     | line style: . . . . . . |
| FL_DASHDOT | line style: _ _ _ _ _ _ |
| FL_DASHDOTDOT | line style: _ _ _ _ _ |
| FL_CAP_FLAT | cap style: end is flat |
| FL_CAP_ROUND | cap style: end is round |
| FL_CAP_SQUARE | cap style: end wraps end point |
| FL_JOIN_MITER | join style: line join extends to a point |
| FL_JOIN_ROUND | join style: line join is rounded |
| FL_JOIN_BEVEL | join style: line join is tidied |

8.7.4 Function Documentation

8.7.4.1 copy_offscreen()

void Fl_Quartz_Graphics_Driver::copy_offscreen (
    int x,
    int y,
    int w,
    int h,
    Fl_Offscreen pixmap,
    int srcx,
    int srcy ) [virtual]

see fl_copy_offscreen()
Reimplemented from Fl_Graphics_Driver.

8.7.4.2 fl_add_symbol()

FL_EXPORT 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

8.7.4.3 fl_arc() [1/2]

```c
void fl_arc {
    double x,
    double y,
    double r,
    double start,
    double end ) [inline]
```

Adds 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>

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
fl_begin_line();
fl_arc(200.0, 100.0, 50.0, 0.0, 180.0);
fl_end_line();

// Draw filled arc
fl_begin_polygon();
fl_arc(300.0, 100.0, 50.0, 0.0, 180.0);
fl_end_polygon();

8.7.4.4 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 WIN32. 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 o’clock and are the starting and ending angle of the arc, a2 must be greater or equal to a1.
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 than the double version fl_arc(double x, double y, double r, double start, double end)
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>bounding box of complete circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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>

8.7.4.5  fl_begin_complex_polygon()

void fl_begin_complex_polygon ( ) [inline]

Starts drawing a complex filled polygon. The polygon may be concave, may have holes in it, or may be several disconnected pieces. Call fl_gap() to separate loops of the path. To outline the polygon, use fl_begin_loop() and replace each fl_gap() with 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.

8.7.4.6  fl_begin_offscreen()

void fl_begin_offscreen ( Fl_Offscreen ctx )

Send all subsequent drawing commands to this offscreen buffer.

Parameters

ctx  the offscreen buffer.

8.7.4.7  fl_begin_points()

void fl_begin_points ( ) [inline]

Starts drawing a list of points. Points are added to the list with fl_vertex()

8.7.4.8  fl_can_do_alpha_blending()

FL_EXPORT char fl_can_do_alpha_blending ( )

Checks 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

8.7.4.9  fl_circle()

void fl_circle ( double x, double y, double r ) [inline]

fl_circle() 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()

Parameters

| in | x,y,r | center and radius of circle |
8.7.4.10 fl_clip_box()

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

Intersects the rectangle with the current clip region and returns the bounding box of the result. Returns non-zero if the resulting rectangle is different to the original. This can be used to limit the necessary drawing to a rectangle. 

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size of rectangle</td>
</tr>
<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.

8.7.4.11 fl_clip_region()

```c
void fl_clip_region (  
    Fl_Region r )  
    {inline}
```

Replaces the top of the clipping stack with a clipping region of any shape. Fl_Region is an operating system specific type.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>r</td>
<td>clipping region</td>
</tr>
</tbody>
</table>

8.7.4.12 fl_copy_offscreen()

```c
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>position where to draw the copied rectangle</td>
<td></td>
</tr>
<tr>
<td>w,h</td>
<td>size of the copied rectangle</td>
<td></td>
</tr>
<tr>
<td>pixmap</td>
<td>offscreen buffer containing the rectangle to copy</td>
<td></td>
</tr>
<tr>
<td>srcx,srcy</td>
<td>origin in offscreen buffer of rectangle to copy</td>
<td></td>
</tr>
</tbody>
</table>
8.7.4.13 fl_create_offscreen()

Fl_Offscreen fl_create_offscreen (  
    int w,  
    int h  )

Creation of an offscreen graphics buffer.

Parameters

| w, h | width and height in pixels of the buffer. |

Returns

the created graphics buffer.

8.7.4.14 fl_cursor()

FL_EXPORT 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.

8.7.4.15 fl_curve()

void fl_curve (  
    double X0,  
    double Y0,  
    double X1,  
    double Y1,  
    double X2,  
    double Y2,  
    double X3,  
    double Y3 )  [inline]

Adds a series of points on a Bezier curve to the path. The curve ends (and two of the points) are at X0,Y0 and X3,Y3.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X0,Y0</th>
<th>curve start point</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X1,Y1</td>
<td>curve control point</td>
</tr>
<tr>
<td>in</td>
<td>X2,Y2</td>
<td>curve control point</td>
</tr>
<tr>
<td>in</td>
<td>X3,Y3</td>
<td>curve end point</td>
</tr>
</tbody>
</table>

8.7.4.16 fl_delete_offscreen()

void fl_delete_offscreen (  
    Fl_Offscreen ctx  )

Deletion of an offscreen graphics buffer.

Parameters

| ctx | the buffer to be deleted. |
8.7 Drawing functions

8.7.4.17 fl_draw() [1/4]

FL_EXPORT void fl_draw (const char * str, int x, int y)
Draws a nul-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.

8.7.4.18 fl_draw() [2/4]

FL_EXPORT 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.

8.7.4.19 fl_draw() [3/4]

void fl_draw (int angle, const char * str, int n, int x, int y) [inline]
Draws 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.

8.7.4.20 fl_draw() [4/4]

FL_EXPORT 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. This version of fl_draw provides direct access to the text drawing function of the underlying OS and is supported by Xft, Win32 and MacOS fltk subsets.

8.7.4.21 fl_draw_box()

FL_EXPORT 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></th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td>position</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>color</td>
</tr>
</tbody>
</table>

8.7.4.22 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]

Draws an 8-bit per color RGB or luminance image.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>position where to put top-left corner of image</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>size of the image</td>
</tr>
<tr>
<td>W</td>
<td></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>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></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>
<tr>
<td>L</td>
<td></td>
<td></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.

8.7.4.23 fl_draw_image() [2/2]

void fl_draw_image ( 
    Fl_Draw_Image_Cb cb,
    void * data,
    int X,
Draws 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 easily.

### Parameters

- **cb**: callback function to generate scan line data
- **data**: user data passed to callback function
- **X, Y**: screen position of top left pixel
- **W, H**: image width and height
- **D**: data size 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.

### 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]
```

Draws 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)`

### 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]
```

Draws a gray-scale image using a callback function to generate image data.
See also

\[ \text{fl\_draw\_image(Fl\_Draw\_Image\_Cb\ cb, void* data, int X, int Y, int W, int H, int D)} \]

### 8.7.4.26 fl\_draw\_pixmap() [1/2]

```c
FL_EXPORT int fl_draw_pixmap ( char *const * data,
    int x,
    int y,
    Fl_Color bg )
```

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**

<table>
<thead>
<tr>
<th>in</th>
<th>data</th>
<th>pointer to XPM image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
<td>position of top-left corner</td>
</tr>
<tr>
<td>in</td>
<td>bg</td>
<td>background color</td>
</tr>
</tbody>
</table>

**Returns**

0 if there was any error decoding the XPM data.

### 8.7.4.27 fl\_draw\_pixmap() [2/2]

```c
FL_EXPORT int fl_draw_pixmap ( const char *const * cdata,
    int x,
    int y,
    Fl_Color bg )
```

Draw XPM image data, with the top-left corner at the given position.

See also

\[ \text{fl\_draw\_pixmap(char* const* data, int x, int y, Fl\_Color bg)} \]

### 8.7.4.28 fl\_draw\_symbol()

```c
FL_EXPORT int fl_draw_symbol ( const char * label,
    int x,
    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</th>
<th>label</th>
<th>name of symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
<td>position of symbol</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
<td>size of symbol</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>color of symbox</td>
</tr>
</tbody>
</table>
Returns

1 on success, 0 on failure

### 8.7.4.29 fl_expand_text() (FL_EXPORT 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.

### 8.7.4.30 fl_frame()

\[ \text{FLEXPORT void fl_frame (const char * s, int x, int y, int w, int h)} \]

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.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>sets of 4 grayscale values in top, left, bottom, right order</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
</tbody>
</table>

### 8.7.4.31 fl_frame2()

\[ \text{FLEXPORT void fl_frame2 (const char * s, int x, int y, int w, int h)} \]

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**

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>sets of 4 grayscale values in bottom, right, top order</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
</tbody>
</table>
8.7.4.32  fl_gap()

void fl_gap ( ) [inline]
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 vertex, or several times in a row.

8.7.4.33  fl_line_style()

void fl_line_style ( 
    int style,
    int width = 0,
    char * dashes = 0 ) [inline]
Sets 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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>style</td>
<td>A bitmask which is a bitwise-OR of a line style, a cap style, and a join style. 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.</td>
</tr>
<tr>
<td>in</td>
<td>width</td>
<td>The thickness of the lines in pixels. Zero results in the system defined default, which on both X and Windows is somewhat different and nicer than 1.</td>
</tr>
<tr>
<td>in</td>
<td>dashes</td>
<td>A pointer to an array of dash lengths, measured in pixels. 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.</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 Windows 95, 98 or Me, since those operating systems do not support complex line styles.

8.7.4.34  fl_measure()

FL_EXPORT 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
<td>nul-terminated string</td>
</tr>
<tr>
<td>out</td>
<td>w,h</td>
<td>width and height of string in current font</td>
</tr>
<tr>
<td>in</td>
<td>draw_symbols</td>
<td>non-zero to enable @symbol handling [default=1]</td>
</tr>
</tbody>
</table>

// Example: Common use case for fl_measure()
const char *s = "This is a test";
int wi=0, hi=0; // 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, wi, hi); // returns pixel width/height of string in current font

### 8.7.4.35 fl_measure_pixmap() [1/2]

FL_EXPORT 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.

**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

### 8.7.4.36 fl_measure_pixmap() [2/2]

FL_EXPORT 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)

### 8.7.4.37 fl_mult_matrix()

void fl_mult_matrix(
    double a,
    double b,
    double c,
    double d,
    double x,
    double y ) [inline]

Concatenates another transformation onto the current one.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>a,b,c,d,x,y</th>
<th>transformation matrix elements such that X' = aX + cY + x and Y' = bX +dY + y</th>
</tr>
</thead>
</table>

### 8.7.4.38 fl_not_clipped()

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.

Note

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

8.7.4.39 fl_old_shortcut()

FL_EXPORT unsigned int fl_old_shortcut (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_ALT|FL_CTRL|FL_META|'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.

If you want... Then use...
--- ---------------------------- -------------------------------
"#" as the shortcut.. "0x23" (instead of just "#").
"+" as the shortcut.. "0x2b" (instead of just "+").
"^" as the shortcut.. "0x5e" (instead of just "^").
Alt-" as the shortcut.. "0x2b" (instead of "+").
Alt-^ as the shortcut.. "0x5e" (instead of "^"), etc..
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.

**Todo** Fix these silly legacy issues in a future release to support more predictable behavior for the modifier keys.

### 8.7.40 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 `fl_arc()`, but `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`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x,y,w,h</code></td>
<td>bounding box of complete circle</td>
</tr>
<tr>
<td><code>a1,a2</code></td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. <code>a2</code> must be greater than or equal to <code>a1</code>.</td>
</tr>
</tbody>
</table>

### 8.7.41 fl_polygon() [1/2]

```c
void fl_polygon (  
    int x,  
    int y,  
    int x1,  
    int y1,  
    int x2,  
    int y2 ) [inline]
```

Fills a 3-sided polygon.
The polygon must be convex.

### 8.7.42 fl_polygon() [2/2]

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

Fills a 4-sided polygon.
The polygon must be convex.

### 8.7.43 fl_pop_clip()

```c
void fl_pop_clip ( ) [inline]
```

Restores 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.
8.7.4.44 fl_push_clip()

void fl_push_clip (  
    int x,  
    int y,  
    int w,  
    int h )  [inline]

Intersects the current clip region with a rectangle and pushes this new region onto the stack.

Parameters

| in | x, y, w, h | position and size |

8.7.4.45 fl_push_matrix()

void fl_push_matrix ( )  [inline]

Saves the current transformation matrix on the stack. The maximum depth of the stack is 32.

8.7.4.46 fl_read_image()

FL_EXPORT 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.

Parameters

| in | p | pixel buffer, or NULL to allocate one |
| in | X, Y | position of top-left of image to read |
| in | W, H | width and height of image to read |
| in | alpha | alpha value for image (0 for none) |

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.

8.7.4.47 fl_rect()

void fl_rect (  
    int x,  
    int y,  
    int w,  
    int h )  [inline]

Draws a 1-pixel border 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.
8.7.4.48  fl_rectf()

FL_EXPORT void fl_rectf (  
    int x,  
    int y,  
    int w,  
    int h,  
    uchar r,  
    uchar g,  
    uchar b )

Colors 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.

8.7.4.49  fl_reset_spot()

FL_EXPORT void fl_reset_spot (  
    void )

Todo  provide user documentation for fl_reset_spot function

8.7.4.50  fl_rotate()

void fl_rotate (  
    double d ) [inline]

Concatenates rotation transformation onto the current one.

Parameters

| in   | \(d\) | rotation angle, counter-clockwise in degrees (not radians) |

8.7.4.51  fl_scale() [1/2]

void fl_scale (  
    double x ) [inline]

Concatenates scaling transformation onto the current one.

Parameters

| in   | \(x\) | scale factor in both \(x\)-direction and \(y\)-direction |

8.7.4.52  fl_scale() [2/2]

void fl_scale (  
    double x,  
    double y ) [inline]

Concatenates scaling transformation onto the current one.

Parameters

| in   | \(x,y\) | scale factors in \(x\)-direction and \(y\)-direction |

8.7.4.53  fl_scroll()
int \textit{X},
int \textit{Y},
int \textit{W},
int \textit{H},
int \textit{dx},
int \textit{dy},

\begin{verbatim}
void(*)(void *, int, int, int, int) \textit{draw_area},
void * \textit{data})
\end{verbatim}

Scroll a rectangle and draw the newly exposed portions.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{X}, \textit{Y}</td>
<td>position of top-left of rectangle</td>
</tr>
<tr>
<td>\textit{W}, \textit{H}</td>
<td>size of rectangle</td>
</tr>
<tr>
<td>\textit{dx}, \textit{dy}</td>
<td>pixel offsets for shifting rectangle</td>
</tr>
<tr>
<td>\textit{draw_area}</td>
<td>callback function to draw rectangular areas</td>
</tr>
<tr>
<td>\textit{data}</td>
<td>pointer to user data for callback</td>
</tr>
</tbody>
</table>

The contents of the rectangular area is first shifted by \textit{dx} and \textit{dy} pixels. The \textit{draw_area} callback is then called for every newly exposed rectangular area.

#### 8.7.4.54 \texttt{fl_set_spot()}

\begin{verbatim}
FL_EXPORT void fl_set_spot (  
    int \textit{font},  
    int \textit{size},  
    int \textit{X},  
    int \textit{Y},  
    int \textit{W},  
    int \textit{H},  
    Fl_Window * \textit{win} = 0 )
\end{verbatim}

Todo provide user documentation for \texttt{fl_set_spot} function

#### 8.7.4.55 \texttt{fl_set_status()}

\begin{verbatim}
FL_EXPORT void fl_set_status (  
    int \textit{X},  
    int \textit{Y},  
    int \textit{W},  
    int \textit{H}  
)
\end{verbatim}

Todo provide user documentation for \texttt{fl_set_status} function

#### 8.7.4.56 \texttt{fl_shortcut_label()}

\begin{verbatim}
FL_EXPORT const char * fl_shortcut_label (  
    unsigned int \textit{shortcut} )
\end{verbatim}

Get a human-readable string from a shortcut value.

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 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`

```c
fl_local_ctrl = "Strg"; // German for "Ctrl"
fl_local_shift = "Umschalt"; // German for "Shift"
```

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:
```c
Ctrl+Alt+Shift+Meta+F12
```

The default values for modifier key names are as given above for all platforms except Mac OS X. Mac OS X uses graphical characters that represent the typical OS X modifier names in menus, e.g. cloverleaf, saucepan, etc. You may, however, redefine Mac OS X modifier names as well.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>shortcut</th>
<th>the integer value containing the ascii character or extended keystroke plus modifiers</th>
</tr>
</thead>
</table>

Returns

a pointer to a static buffer containing human readable text for the shortcut

#### 8.7.4.57 flShortcutLabel() [2/2]

```c
FL_EXPORT const char * flShortcutLabel ( 
    unsigned int shortcut, 
    const char ** eom )
```

Get a human-readable string from a shortcut value.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>shortcut</th>
<th>the integer value containing the ascii character or extended keystroke plus modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>eom</td>
<td>if this pointer is set, it will receive a pointer to the end of the modifier text</td>
</tr>
</tbody>
</table>

Returns

a pointer to a static buffer containing human readable text for the shortcut

See also

`flShortcutLabel(unsigned int shortcut)`

#### 8.7.4.58 flTransform_dx()

```c
double flTransform_dx ( 
    double x, 
    double y ) [inline]
```

Transforms distance using current transformation matrix.
Parameters

\[ \text{in } x,y \text{ coordinate} \]

8.7.4.59 fl_transform_dy()

double fl_transform_dy ( 
    double x,
    double y ) [inline]
Transforms distance using current transformation matrix.

Parameters

\[ \text{in } x,y \text{ coordinate} \]

8.7.4.60 fl_transform_x()

double fl_transform_x ( 
    double x,
    double y ) [inline]
Transforms coordinate using the current transformation matrix.

Parameters

\[ \text{in } x,y \text{ coordinate} \]

8.7.4.61 fl_transform_y()

double fl_transform_y ( 
    double x,
    double y ) [inline]
Transforms coordinate using the current transformation matrix.

Parameters

\[ \text{in } x,y \text{ coordinate} \]

8.7.4.62 fl_transformed_vertex()

void fl_transformed_vertex ( 
    double xf,
    double yf ) [inline]
Adds coordinate pair to the vertex list without further transformations.

Parameters

\[ \text{in } xf,yf \text{ transformed coordinate} \]

8.7.4.63 fl_translate()

void fl_translate ( 
    double x,
    double y ) [inline]
Concatenates translation transformation onto the current one.

Parameters

| in | x, y | translation factor in x-direction and y-direction |

8.7.4.64 fl_vertex()

```c
void fl_vertex ( double x, double y ) [inline]
```

Adds a single vertex to the current path.

Parameters

| in | x, y | coordinate |

8.8 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.

8.8.1 Detailed Description

fl multithreading support functions declared in `<FL/Fl.H>`

8.8.2 Function Documentation

8.8.2.1 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

```c
Fl::awake(void *message=0)
```
8.8.2.2 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 redrags 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.

See also: Multithreading

8.8.2.3 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

8.8.2.4 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

8.8.2.5 unlock()

```c
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

8.9 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)`
  
  Releases a widget pointer from the watch list.
8.9 Safe widget deletion support functions

- static void Fl::watch_widget_pointer (Fl_Widget *w)

  Adds a widget pointer to the widget watch list.

8.9.1 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::watch_widget_pointer()

8.9.2 Function Documentation

8.9.2.1 clear_widget_pointer()

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
8.9.2.2 delete_widget()

```cpp
template
class Fl::Fl_Widget

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::~Fl_Widget()

8.9.2.3 do_widget_deletion()

```cpp
templatetd
class Fl::Fl_Widget

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)

8.9.2.4 release_widget_pointer()

```cpp
templatetd
class Fl::Fl_Widget

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()
8.9.2.5 watch_widget_pointer()

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):

```c
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

8.10 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 a pre-configured cairo fltk window.

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 cairo_t * Fl::cairo_make_current (Fl_Window *w)
  
  Provides a corresponding cairo context for window w.
8.10.1 Detailed Description

8.10.2 Function Documentation

8.10.2.1 cairo_autolink_context() [1/2]

static bool Fl::cairo_autolink_context ( ) [inline], [static]
Gets the current autolink mode for cairo support.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if no cairo context autolink is made for each window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if any fltk window is attached a cairo context when it is current.</td>
</tr>
</tbody>
</table>

See also

void cairo_autolink_context(bool alink)

Note

Only available when configure has the –enable-cairo option

8.10.2.2 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*). This will create a cairo context but 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

8.10.2.3 cairo_cc()  

static void Fl::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

8.10.2.4 cairo_make_current()

cairo_t * Fl::cairo_make_current (FL_Window * wi ) [static]
Provides a corresponding 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__context() 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.
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.

### 8.11 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`
- `#define ERRORS_TO_ISO8859_1 1`
- `#define NBC 0xFFFF + 1`
- `#define STRICT_RFC3629 0`

**Functions**

- `FL_EXPORT int fl_access (const char *f, int mode)`
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- `FL_EXPORT int fl_chmod (const char *f, int mode)`
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- `FL_EXPORT int fl_execvp (const char *file, char *const *argv)`
  Cross-platform function to open files with a UTF-8 encoded name.
- `FL_EXPORT char * fl_getcwd (char *b, int l)`
  Cross-platform function to get the current working directory as a UTF-8 encoded value.
- `FL_EXPORT char * fl_getenv (const char *v)`
  Cross-platform function to get environment variables with a UTF-8 encoded name or value.
- `FL_EXPORT char fl_make_path (const char *path)`
  Cross-platform function to recursively create a path in the file system.
- `FL_EXPORT void fl_make_path_for_file (const char *path)`
  Cross-platform function to create a path for the file in the file system.
- `FL_EXPORT int fl_mkdir (const char *f, int mode)`
  Cross-platform function to create a directory with a UTF-8 encoded name.
- `FL_EXPORT unsigned int fl_nonspacing (unsigned int ucs)`
  Returns true if the Unicode character `ucs` is non-spacing.
- `FL_EXPORT int fl_open (const char *f, int oflags,...)`
  Cross-platform function to open files with a UTF-8 encoded name.
- `FL_EXPORT int fl_rename (const char *f, const char *n)`
  Cross-platform function to rename a filesystem object using UTF-8 encoded names.
- `FL_EXPORT int fl_rmdir (const char *f)`
  Cross-platform function to remove a directory with a UTF-8 encoded name.
- `FL.Export int fl_stat (const char *f, struct stat *b)`
  Cross-platform function to stat() a file using a UTF-8 encoded name or value.
- `FL_EXPORT int fl_system (const char *cmd)`
  Cross-platform function to run a system command with a UTF-8 encoded string.
• FL_EXPORT int fl_tolower (unsigned int ucs)
  Returns the Unicode lower case value of ucs.

• FL_EXPORT int fl_toupper (unsigned int ucs)
  Returns the Unicode upper case value of ucs.

• FL_EXPORT unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)
  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

• FL_EXPORT char * fl_utf2mbcs (const char *s)
  Converts UTF-8 string s to a local multi-byte character string.

• FL_EXPORT int fl_unlink (const char *f)
  Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.

• FL_EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.

• FL EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Converts a UTF-8 string into a wide character string.

• FL_EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Returns the number of Unicode chars in the UTF-8 string.

• FL_EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Returns the number of Unicode chars in the UTF-8 string.

• FL_EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Returns the number of Unicode chars in the UTF-8 string.

• FL_EXPORT int fl_utf8locale (void)

• FL_EXPORT int fl_utf8locale1 (char c)
  Converts the string str to its lower case equivalent into buf.

• FL_EXPORT int fl_utf_toucher (const unsigned char *str, int len, char *buf)
  Converts the string str to its upper case equivalent into buf.

• FL_EXPORT int fl_wcwidth (const char *src)
  extended wrapper around fl_wcwidth_(unsigned int ucs) function.

• FL_EXPORT int fl_wcwidth_ (unsigned int ucs)
  wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK

8.11.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in <FL/fl_utf8.h>
8.11 Unicode and UTF-8 functions

8.11.2 Macro Definition Documentation

8.11.2.1 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.

8.11.2.2 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 RE-PLACEMENT 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.

8.11.2.3 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.

8.11.3 Function Documentation

8.11.3.1 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 under the MSWindows platform where the standard access() 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 test</td>
</tr>
</tbody>
</table>

Returns

the return value of _waccess() on Windows or access() on other platforms.

8.11.3.2 fl_chmod()

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 under the MSWindows 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>
Returns

the return value of _wchmod() on Windows or chmod() on other platforms.

8.11.3.3 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 under the MSWindows platform where the standard fopen() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</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().

8.11.3.4 fl_getcwd()

```c
char * fl_getcwd ( char * b, int l )
```

Cross-platform function to get the current working directory as a UTF-8 encoded value.
This function is especially useful under the MSWindows platform where the standard _wgetcwd() function returns UTF-16 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>b</th>
<th>the buffer to populate</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>the length of the buffer</td>
</tr>
</tbody>
</table>

Returns

the CWD encoded as UTF-8.

8.11.3.5 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 under the MSWindows 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 MSWindows this function calls getenv directly. The return value is returned as-is.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>v</th>
<th>the UTF-8 encoded environment variable</th>
</tr>
</thead>
</table>
Returns
the environment variable in UTF-8 encoding, or NULL in case of error.

8.11.3.6 fl_make_path()

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.

8.11.3.7 fl_make_path_for_file()

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.

8.11.3.8 fl_mkdir()

int fl_mkdir (const char * f, int mode )
Cross-platform function to create a directory with a UTF-8 encoded name.
This function is especially useful on the MSWindows platform where the standard _wmkdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in f</td>
<td>the UTF-8 encoded filename</td>
</tr>
<tr>
<td>in mode</td>
<td>the mode of the directory</td>
</tr>
</tbody>
</table>

Returns
the return value of _wmkdir() on Windows or mkdir() on other platforms.

8.11.3.9 fl_nonspacing()

unsigned int fl_nonspacing (unsigned int ucs )
Returns true if the Unicode character 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).

• http://unicode.org/glossary/#base_character
• http://unicode.org/glossary/#nonspacing_mark
• http://unicode.org/glossary/#combining_character

8.11.3.10 fl_open()

int fl_open (const char * f, int oflags, ...)
Cross-platform function to open files with a UTF-8 encoded name.
This function is especially useful under the MSWindows platform where the standard open() function fails with UTF-8 encoded non-ASCII filenames.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>the UTF-8 encoded filename</td>
</tr>
<tr>
<td>oflags</td>
<td>other arguments are as in the standard open() function</td>
</tr>
</tbody>
</table>

Returns

- a file descriptor upon successful completion, or -1 in case of error.

See also

fl_fopen().

8.11.3.11 fl_rename()

```c
int fl_rename (  
    const char * f,  
    const char * n )
```

Cross-platform function to rename a filesystem object using UTF-8 encoded names. This function is especially useful on the MSWindows platform where the standard _wrename() function expects UTF-16 encoded non-ASCII filenames.

Parameters

- \( f \) the UTF-8 encoded filename to change
- \( n \) the new UTF-8 encoded filename to set

Returns

- the return value of _wrename() on Windows or rename() on other platforms.

8.11.3.12 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 MSWindows platform where the standard _wrmdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

- \( f \) the UTF-8 encoded filename to remove

Returns

- the return value of _wrmdir() on Windows or rmdir() on other platforms.

8.11.3.13 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 under the MSWindows platform where the standard stat() 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></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.

8.11.3.14  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 under the MSWindows platform where non-ASCII program (file) names must be encoded as wide characters.
On platforms other than MSWindows this function calls system() directly.

Parameters

| in | cmd | the UTF-8 encoded command string |

Returns
the return value of _wsystem() on Windows or system() on other platforms.

8.11.3.15  fl_ucs_to_Utf16()

```c
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.

8.11.3.16  fl_unlink()

```c
int fl_unlink (const char *f)
```
Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
This function is especially useful under the MSWindows platform where the standard function expects UTF-16 encoded non-ASCII filenames.
Parameters

\[ f \] the filename to unlink

Returns

the return value of _wunlink() on Windows or unlink() on other platforms.

8.11.3.17 \texttt{fl_utf8back()}

\begin{verbatim}
const char* fl_utf8back {
    const char * p,
    const char* start,
    const char* end)
\end{verbatim}

Move \texttt{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. \texttt{start} is the start of the string and is used to limit the backwards search for the start of a UTF-8 character. \texttt{end} is the end of the string and is assumed to be a break between characters. It is assumed to be greater than \texttt{p}. If you wish to decrement a UTF-8 pointer, pass \texttt{p-1} to this.

8.11.3.18 \texttt{fl_utf8bytes()}

\begin{verbatim}
int fl_utf8bytes {
    unsigned ucs} \end{verbatim}

Return the number of bytes needed to encode the given UCS4 character in UTF-8.

Parameters

\begin{verbatim}
in | ucs | UCS4 encoded character
\end{verbatim}

Returns

number of bytes required

Returns number of bytes that utf8encode() will use to encode the character \texttt{ucs}.

8.11.3.19 \texttt{fl_utf8decode()}

\begin{verbatim}
unsigned fl_utf8decode {
    const char * p,
    const char* end,
    int * len}
\end{verbatim}

Decode a single UTF-8 encoded character starting at \texttt{p}. The resulting Unicode value (in the range 0-0x10ffff) is returned, and \texttt{len} is set to the number of bytes in the UTF-8 encoding (adding \texttt{len} to \texttt{p} will point at the next character). If \texttt{p} points at an illegal UTF-8 encoding, including one that would go past \texttt{end}, or where a code uses more bytes than necessary, then \texttt{*(unsigned char*)p} is translated as though it is in the Microsoft CP1252 character set and \texttt{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:

\begin{verbatim}
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;
    }
}\end{verbatim}

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.
8.11.3.20 \texttt{fl\_utf8encode()}

```c
int fl_utf8encode ( unsigned ucs,
                  char * buf )
```

Write the UTF-8 encoding of \texttt{ucs} into \texttt{buf} and return the number of bytes written. Up to 4 bytes may be written. If you know that \texttt{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 \texttt{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 \texttt{ucs} are illegal (in the range 0x8000 to 0xfffff, 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 0x10000.

8.11.3.21 \texttt{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 \texttt{dstlen} bytes are written to \texttt{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 \texttt{dstlen} then if you malloc a new array of size \texttt{n+1} you will have the space needed for the entire string. If \texttt{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 \texttt{fl\_utf8test()} returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.

8.11.3.22 \texttt{fl\_utf8froma()}

```c
unsigned fl_utf8froma ( char * dst,
                  unsigned dstlen,
                  const char * src,
                  unsigned srclen )
```

Convert an ISO-8859-1 (i.e. 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 \texttt{dstlen} bytes are written to \texttt{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 \texttt{dstlen} then if you malloc a new array of size \texttt{n+1} you will have the space needed for the entire string. If \texttt{dstlen} is zero then nothing is written and this call just measures the storage space needed.

\texttt{srclen} is the number of bytes in \texttt{src} to convert.

If the return value equals \texttt{srclen} then this indicates that no conversion is necessary, as only ASCII characters are in the string.

8.11.3.23 \texttt{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 \texttt{dstlen} bytes are written to \texttt{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 \texttt{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.

### 8.11.3.24 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 the pointer moving. Do not use this to scan strings, use fl_utf8decode() instead.

### 8.11.3.25 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

### 8.11.3.26 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

### 8.11.3.27 fl_utf8locale()

```c
int fl_utf8locale ( )
```

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.
8.11 Unicode and UTF-8 functions

8.11.3.28 fl_utf8test()

```c
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.

- 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 (ie no bytes have the high bit set). This is legal UTF-8, but also indicates plain ASCII. It also returns 1 if `srclen` 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.

8.11.3.29 fl_utf8to_mb()

```c
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). 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 `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.

If `fl_utf8locale()` returns true then this does not change the data.

8.11.3.30 fl_utf8toa()

```c
unsigned fl_utf8toa(
    const char * src,
    unsigned srclen,
    char * dst,
    unsigned dstlen
)
```
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_utf8decode()` does. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly (and possibly CP1252 on Windows).

`src` points at the UTF-8 sequence, and `srclen` is the number of bytes to convert.

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.

8.11.3.31 fl_utf8toUtf16()

```c
unsigned fl_utf8toUtf16(
    const char * src,
)
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).

### 8.11.3.32 \texttt{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.

### 8.11.3.33 \texttt{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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the strings are equal</td>
</tr>
<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>
8.11 Unicode and UTF-8 functions

8.11.3.34 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

<table>
<thead>
<tr>
<th>s1,s2</th>
<th>the UTF-8 strings to compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>the maximum number of UTF-8 characters to compare</td>
</tr>
</tbody>
</table>

Returns

result of comparison

Return values

| 0     | if the strings are equal                        |
| >0    | if s1 is greater than s2                        |
| <0    | if s1 is less than s2                           |

8.11.3.35 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]

8.11.3.36 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]

8.11.3.37 fl_wcwidth()

```c
int fl_wcwidth ( 
    const char * src )
```

extended wrapper around fl_wcwidth_(unsigned int ucs) function.

Parameters

| src | pointer to start of UTF-8 byte sequence |

Returns

width of character in columns

Depending on build options, this function may map C1 control characters (0x80 to 0x9f) to CP1252, and return the width of that character instead. This is not the same behaviour as fl_wcwidth_(unsigned int ucs).
Note that other control characters and DEL will still return -1, so if you want different behaviour, you need to test for those characters before calling \texttt{fl_wcwidth()}, and handle them separately.

### 8.11.3.38 \texttt{fl_wcwidth()}\footnote{wrapper to adapt Markus Kuhn's implementation of \texttt{wcwidth()} for FLTK}

\begin{verbatim}
int fl_wcwidth_ (unsigned int ucs )
\end{verbatim}

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\texttt{ucs} & Unicode character value \\
\hline
\end{tabular}

\textbf{Returns}

width of character in columns

See \url{http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c} for Markus Kuhn's original implementation of \texttt{wcwidth()} and \texttt{wcswidth()} (defined in IEEE Std 1002.1-2001) for Unicode.

\textbf{WARNING:} this function returns widths for "raw" Unicode characters. It does not even try to map C1 control characters (0x80 to 0x9F) to CP1252, and C0/C1 control characters and DEL will return -1. You are advised to use \texttt{fl_width(const char * src)} instead.

### 8.12 Mac OS X-specific symbols

Mac OS X-specific symbols declared in \texttt{<FL/x.H>} or \texttt{<FL/gl.h>}

\textbf{Classes}

- class \texttt{Fl_Mac_App_Menu}

  Mac OS-specific class allowing to customize and localize the application menu.

\textbf{Functions}

- void \texttt{fl_mac_set_about (Fl_Callback *cb, void *user_data, int shortcut=0)}

  \textit{Attaches a callback to the "About myprog" item of the system application menu.}

- void \texttt{fl_open_callback (void(*) (const char *) )}

  \textit{Register a function called for each file dropped onto an application icon.}

- void \texttt{gl_texture_pile_height (int max)}

  \textit{Changes the height of the pile of pre-computed string textures.}

- int \texttt{gl_texture_pile_height (void)}

  \textit{Returns the current height of the pile of pre-computed string textures.}

\textbf{Variables}

- int \texttt{fl_mac_os_version}

  \textit{The version number of the running Mac OS X (e.g., 100604 for 10.6.4)}

- int \texttt{fl_mac_quit_early}

  \textit{Determines whether cmd-Q or the "Quit xxx" item of application menu terminates the app or only the event loop.}

- class \texttt{Fl_Sys_Menu_Bar * fl_sys_menu_bar}

  \textit{The system menu bar.}
8.12 Mac OS X-specific symbols

8.12.1 Detailed Description
Mac OS X-specific symbols declared in `<FL/x.H>` or `<FL/gl.h>`

See also
The Apple OS X Interface

8.12.2 Function Documentation

8.12.2.1 fl_mac_set_about()

```c
void fl_mac_set_about ( Fl_Callback ∗ cb,
                      void ∗ user_data,
                      int shortcut = 0 ) [extern]
```

 Attaches a callback to the "About myprog" item of the system application menu.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cb</td>
<td>a callback that will be called by &quot;About myprog&quot; menu item with NULL 1st argument.</td>
</tr>
<tr>
<td>user_data</td>
<td>a pointer transmitted as 2nd argument to the callback.</td>
</tr>
<tr>
<td>shortcut</td>
<td>optional shortcut to attach to the &quot;About myprog&quot; menu item (e.g., FL_META+'a')</td>
</tr>
</tbody>
</table>

8.12.2.2 fl_open_callback()

```c
void fl_open_callback ( void(*)(const char ∗) cb ) [extern]
```

 Register a function called for each file dropped onto an application icon. `cb` will be called with a single Unix-style file name and path. If multiple files were dropped, `cb` will be called multiple times.

8.12.2.3 gl_texture_pile_height() [1/2]

```c
void gl_texture_pile_height ( int max )
```

 Changes the 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>Height of the texture pile</td>
</tr>
</tbody>
</table>

8.12.2.4 gl_texture_pile_height() [2/2]

```c
int gl_texture_pile_height ( void )
```

 Returns the current height of the pile of pre-computed string textures. The default value is 100.

8.12.3 Variable Documentation

8.12.3.1 fl_mac_quit_early

```c
int fl_mac_quit_early [extern]
```

 Determines whether cmd-Q or the "Quit xxx" item of application menu terminates the app or only the event loop. By default, `fl_mac_quit_early` = 1, and cmd-Q or "Quit xxx" terminate the app when all windows are closed without Fl::run() returning. If `fl_mac_quit_early` is set to 0, cmd-Q or "Quit xxx" terminate only the event loop when all
windows are closed, and Fl::run() returns.

Note

This OS-specific variable will not be part of the API of FLTK 1.4.

8.13 Common Dialogs classes and functions

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 message.
- int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style fmt message, this dialog features up to 3 customizable choice buttons.
- int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Like fl_choice() but with extended (negative) return values.
- 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 *name, 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.
- char * fl_input (const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message.
- void fl_message (const char *fmt,...)
  Shows an information message dialog box.
- void fl_message_hotspot (int enable)
  Sets whether or not to move the common 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 common 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_title (const char *title)
  Sets the title of the dialog window used in many common dialogs.
8.13 Common Dialogs classes and functions

- 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.

Variables

- static void( Fl::error )(const char *,...) = ::error

  FLTK calls Fl::error() to output a normal error message.

- static void( Fl::fatal )(const char *,...) = ::fatal

  FLTK 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

- 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 *,...) = ::warning

  FLTK calls Fl::warning() to output a warning message.

8.13.1 Detailed Description

8.13.2 Function Documentation

8.13.2.1 fl_alert()

void fl_alert ( 
    const char * fmt,
    ... )

Shows an alert message dialog box.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time.
Requests for additional dialog boxes are ignored.

#include <FL/fl_ask.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>can be used as an sprintf-like format and variables for the message text</td>
<td></td>
</tr>
</tbody>
</table>

8.13.2.2 fl_ask()

int fl_ask ( 
    const char * fmt,
    ... )

Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.
Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

Parameters

| in | fmt | can be used as an sprintf-like format and variables for the message text |

Return values

| 0 | if the no button is selected or another dialog box is still open |
| 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.

### 8.13.2.3 fl_beep()

```c
void fl_beep ( int type )
```

Emits a system beep message.

Parameters

| in | type | The beep type from the Fl_Beep enumeration. |

Note

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

### 8.13.2.4 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, this dialog features up to 3 customizable choice buttons.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

```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:
8.13 Common Dialogs classes and functions

Figure 8.2 fl_choice() three choices

```
switch ( fl_choice("How many musketeers?", "One", "Two", "Three") ) {
    case 0: .. // One
    case 1: .. // Two (default)
    case 2: .. // Three
}
```

Two choice example:

Figure 8.3 fl_choice() two choices

```
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
    case 0: .. // Yes
    case 1: .. // No (default)
}
```

One choice example:

Figure 8.4 fl_choice() one choice

```
fl_choice("All hope is lost.", "OK", 0, 0); // "OK" default
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>can be used as a sprintf-like format and variables for the message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b0</td>
<td>text label of button 0</td>
</tr>
<tr>
<td>in</td>
<td>b1</td>
<td>text label of button 1 (can be 0)</td>
</tr>
<tr>
<td>in</td>
<td>b2</td>
<td>text label of button 2 (can be 0)</td>
</tr>
</tbody>
</table>

Return values

| 0  | if the first button with b0 text is pushed or another dialog box is still open |
| 1  | if the second button with b1 text is pushed                                  |
| 2  | if the third button with b2 text is pushed                                  |

8.13.2.5 fl_choice_n()

```
int fl_choice_n (  
    const char * fmt,  
    const char * b0,  
    const char * b1,  
    const char * b2,  
    ... )
```

Like fl_choice() but with extended (negative) return values. This function can return negative values as described below whereas fl_choice() only returns "button values" (0, 1, 2).

With fl_choice_n() you can arrange the buttons in a way that any button can be the standard "cancel" button because Escape and closing the window with the close button can be distinguished from button return codes. Negative values are always "special" and should be considered like "cancel".

Generated by Doxygen
The special value \(-3\) means that the dialog was blocked (not executed).
Other than that both functions are the same.

See also

\begin{verbatim}
   fl_choice()
\end{verbatim}

Since

1.3.8

Parameters

\begin{verbatim}
\begin{tabular}{|c|l|}
\hline
   in & \textit{fmt} & can be used as an sprintf-like format and variables for the message text \\
   in & \textit{b0} & text label of button 0 \\
   in & \textit{b1} & text label of button 1 (can be 0) \\
   in & \textit{b2} & text label of button 2 (can be 0) \\
\hline
\end{tabular}
\end{verbatim}

Return values

\begin{verbatim}
\begin{tabular}{|c|l|}
\hline
   \textbf{-3} & if another dialog box is still open (the dialog was blocked) \\
   \textbf{-2} & if the dialog window was closed by clicking the close button \\
   \textbf{-1} & if the dialog was closed by hitting Escape \\
   \textbf{ 0} & if the first button with \textit{b0} text is pushed \\
   \textbf{ 1} & if the second button with \textit{b1} text is pushed \\
   \textbf{ 2} & if the third button with \textit{b2} text is pushed \\
\hline
\end{tabular}
\end{verbatim}

8.13.2.6 \texttt{fl\_color\_chooser()} [1/2]

\begin{verbatim}
int fl_color_chooser ( 
   const char * name, 
   double & r, 
   double & g, 
   double & b, 
   int cmode ) [related]
\end{verbatim}

Pops up a window to let the user pick an arbitrary RGB color.
8.13 Common Dialogs classes and functions

Note

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

![Figure 8.5 fl_color_chooser](image)

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>Title label for the window</td>
</tr>
<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 <code>mode(int)</code>. Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if user confirms the selection</td>
</tr>
<tr>
<td>0</td>
<td>if user cancels the dialog</td>
</tr>
</tbody>
</table>

**8.13.2.7 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.
```
Note

#include <FL/Fl_Color_Chooser.H>

Figure 8.6 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</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 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     |

8.13.2.8 fl_dir_chooser()

char * fl_dir_chooser (const char * message,
 const char * fname,
 int relative ) [related]

Shows a file chooser dialog and gets a directory.

Note

#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>
8.13 Common Dialogs classes and functions

Returns

the directory path string chosen by the user or NULL if user cancels

8.13.2.9 fl_file_chooser()

```c
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>
```

**Figure 8.7 Fl_File_Chooser**

![File Chooser](image)

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>message</td>
<td>text in title bar</td>
</tr>
<tr>
<td>in</td>
<td>pat</td>
<td>filename pattern filter</td>
</tr>
<tr>
<td>in</td>
<td>fname</td>
<td>initial/default filename selection</td>
</tr>
<tr>
<td>in</td>
<td>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

8.13.2.10 fl_file_chooser_callback()

```c
void fl_file_chooser_callback ( 
    void(*)(const char *) cb ) [related]
```
Set the file chooser callback.

Note

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

### 8.13.2.11 fl_file_chooser_ok_label()

```c
void fl_file_chooser_ok_label (const char * l) [related]
```

Set the "OK" button label.

Note

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

### 8.13.2.12 fl_input()

```c
const char * fl_input (const char * fmt, const char * defstr, ...)
```

Shows an input dialog displaying the `fmt` message.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>fmt</code></td>
<td>can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td>in <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, NULL if Cancel was pushed or another dialog box was still open

### 8.13.2.13 fl_message()

```c
void fl_message (const char * fmt, ...)
```

Shows an information message dialog box.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>fmt</code></td>
<td>can be used as an sprintf-like format and variables for the message text</td>
</tr>
</tbody>
</table>
8.13 Common Dialogs classes and functions

8.13.2.14 fl_message_hotspot() [1/2]

```c
void fl_message_hotspot ( int enable )
```

Sets whether or not to move the common 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.

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

Parameters:
- **in enable**
  - non-zero enables hotspot behavior, 0 disables hotspot

8.13.2.15 fl_message_hotspot() [2/2]

```c
int fl_message_hotspot ( void )
```

Gets whether or not to move the common 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.

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

Returns:
- 0 if disable, non-zero otherwise

See also:
- fl_message_hotspot(int)

8.13.2.16 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())

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

8.13.2.17 fl_message_title()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>title</th>
<th>window label, string copied internally</th>
</tr>
</thead>
</table>

8.13.2.18 fl_message_title_default()

```c
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.

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>title</th>
<th>default window label, string copied internally</th>
</tr>
</thead>
</table>

8.13.2.19 fl_password()

```c
const char * fl_password (const char *fmt, const char *defstr, ... )
```

Shows an input dialog displaying the fmt message. Like `fl_input()` except the input text is not shown, '*' characters are displayed instead.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>can be used as a sprintf-like format and variables for the message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>defstr</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, NULL if Cancel was pushed or another dialog box was still open

8.13.3 Variable Documentation

8.13.3.1 error

```c
void(* Fl::error)(const char *format,...) = ::error [static]
```

FLTK 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.
File names and URI utility functions

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

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

### 8.13.3.2 fatal

```c
void(* Fl::fatal)(const char *format,...) = ::fatal [static]
```

FLTK 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 FLTK 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 FLTK again. The default implementation exits with status 1 after displaying the message.

Note

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

### 8.13.3.3 warning

```c
void(* Fl::warning)(const char *format,...) = ::warning [static]
```

FLTK 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

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

### 8.14 File names and URI utility functions

File names and URI functions defined in `<FL/filename.H>`

#### Macros

- `#define fl_dirent_h_cyclic_include`
- `#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

- `FL_EXPORT void fl_decode_uri (char *uri)`

  *Decodes a URL-encoded string.*

- `FL_EXPORT int fl_filename_absolute (char *to, int tolen, const char *from)`
Makes a filename absolute from a relative filename.

- FL_EXPORT int fl_filename_expand (char *to, int tolen, const char *from)
  Expands a filename containing shell variables and tilde (~).
- FL_EXPORT const char * fl_filename_ext (const char *buf)
  Gets the extensions of a filename.
- FL_EXPORT void fl_filename_free_list (struct dirent **l, int n)
  Frees the list of filenames that is generated by fl_filename_list().
- FL_EXPORT int fl_filename_isdir (const char *name)
  Determines if a file exists and is a directory from its filename.
- FL_EXPORT int fl_filename_list (const char *d, struct dirent ***l, Fl_File_Sort_F *s=fl_numericsort)
  Portable and const-correct wrapper for the scandir() function.
- FL_EXPORT int fl_filename_match (const char *name, const char *pattern)
  Checks if a string s matches a pattern p.
- FL_EXPORT const char * fl_filename_name (const char *filename)
  Gets the file name from a path.
- FL_EXPORT int fl_filename_relative (char *to, int tolen, const char *from)
  Makes a filename relative to the current working directory.
- FL_EXPORT char * fl_filename_setext (char *to, int tolen, const char *ext)
  Replaces the extension in buf of max.
- FL_EXPORT int fl_open_uri (const char *uri, char *msg, int msglen)
  Opens the specified Uniform Resource Identifier (URI).

8.14.1 Detailed Description

File names and URI functions defined in `<FL/filename.H>`

8.14.2 Typedef Documentation

8.14.2.1 Fl_File_Sort_F

typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
File sorting function.

See also

fl_filename_list()

8.14.3 Function Documentation

8.14.3.1 fl_decode_uri()

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.

8.14.3.2 fl_filename_absolute()

FL_EXPORT int fl_filename_absolute (char * to, int tolen, const char * from)
Makes a filename absolute from a relative filename.

```
#include <FL/filename.H>
[..]
chdir("/var/tmp");
```
fl_filename_absolute(out, sizeof(out), "foo.txt"); // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), ".//foo.txt"); // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), "../log/messages"); // out="/var/log/messages"
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting absolute filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the absolute filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>relative filename</td>
</tr>
</tbody>
</table>

Returns

0 if no change, non zero otherwise

8.14.3.3 fl_filename_expand()

FL_EXPORT int fl_filename_expand (char * to,
int tolen,
const char * from )

Expands a filename containing shell variables and tilde (\~). Currently handles these variants:

- "~username" // if 'username' does not exist, result will be unchanged
- "~/file"
- "$VARNAME" // does NOT handle ${VARNAME}

Examples:

```c
#include <FL/filename.H>
[..]  
putenv("TMPDIR=/var/tmp");
fl_filename_expand(out, sizeof(out), "~-fred/.cshrc");  // out="/usr/fred/.cshrc"
fl_filename_expand(out, sizeof(out), "~/.cshrc");  // out="/usr/<yourname>/.cshrc"
fl_filename_expand(out, sizeof(out), "$TMPDIR/foo.txt");  // out="/var/tmp/foo.txt"
```

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting expanded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the expanded filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>filename containing shell variables</td>
</tr>
</tbody>
</table>

Returns

0 if no change, non zero otherwise

8.14.3.4 fl_filename_ext()

FL_EXPORT const char * fl_filename_ext (const char * buf )

Gets the extensions 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

| in | buf | the filename to be parsed |
8.14 File names and URI utility functions

Returns

a pointer to the extension (including '.') if any or NULL otherwise

8.14.3.5 fl_filename_free_list()

FL_EXPORT 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

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>list</td>
<td>table containing the resulting directory listing</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of entries in the list</td>
</tr>
</tbody>
</table>

8.14.3.6 fl_filename_isdir()

FL_EXPORT int fl_filename_isdir (  
    const char * n )

Determines if a file exists and is a directory from its filename.

#include <FL/filename.H>

fl_filename_isdir("/etc"); // returns non-zero
fl_filename_isdir("/etc/hosts"); // returns 0

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the filename to parse</td>
</tr>
</tbody>
</table>

Returns

non zero if file exists and is a directory, zero otherwise

8.14.3.7 fl_filename_list()

FL_EXPORT int fl_filename_list (  
    const char * d,  
    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 null-terminated file name. An 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 WIN32.

Include:

#include <FL/filename.H>

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>d</td>
<td>the name of the directory to list. It does not matter if it has a trailing slash.</td>
</tr>
<tr>
<td>out</td>
<td>list</td>
<td>table containing the resulting directory listing</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sort</th>
<th>sorting functor:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>fl_alphasort</code>: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared according to their ASCII ordering - uppercase before lowercase.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>fl_casealphasort</code>: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared equally; case is not significant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>fl_casenumeralsort</code>: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared equally; case is not significant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>fl_numericsort</code>: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared according to their ASCII ordering - uppercase before lowercase.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

the number of entries if no error, a negative value otherwise.

### 8.14.3.8 fl_filename_match()

FL_EXPORT 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.
- `{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 must be matched exactly.

Include:

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>the string to check for a match</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p</td>
<td>the string pattern</td>
</tr>
</tbody>
</table>

Returns

non zero if the string matches the pattern

### 8.14.3.9 fl_filename_name()

FL_EXPORT const char * fl_filename_name (  
    const char * filename )

Generated by Doxygen
8.14 File names and URI utility functions

Gets the file name from a path.
Similar to basename(3), exceptions shown below.

```c
#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("/usr");     // out="usr"
out = fl_filename_name("/.");      // out="." (basename(3) returns "/" instead)
out = fl_filename_name("..");      // out="..

Returns
a pointer to the char after the last slash, or to filename if there is none.

8.14.3.10 fl_filename_relative()

FL_EXPORT int fl_filename_relative {
    char * to,
    int tolen,
    const char * from
}

Makes a filename relative to the current working directory.

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

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 (no change)
fl_filename_relative(out, sizeof(out), ".foo.txt");                // out="../foo.txt", return=1 (no change)
fl_filename_relative(out, sizeof(out), "..oo.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.

8.14.3.11 fl_filename_setext()

FL_EXPORT 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's no '.' in buf, ext is appended.
If ext is NULL, behaves as if it were an empty string ("").

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.
8.14.3.12  fl_open_uri()

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
#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 9

Class Documentation

9.1 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

9.2 Fl Class Reference

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

#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_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 *data=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)
  See void add_fd(int fd, int when, Fl_FD_Handler cb, void*=0)
- 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 *data=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, 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 int args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)
  Parse command line switches using the cb argument handler.

- 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 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 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 cairo_t * cairo_make_current (Fl_Window *w)
  Provides a corresponding cairo context for window wi.
• 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 display to use for all windows.*

• static int dnd ()
   
   *Initiate a Drag And Drop operation.*

• static int dnd_text_ops ()
   
   *Gets or sets whether drag and drop text operations are supported.*

• static void dnd_text_ops (int v)
   
   *Gets or 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 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 window 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 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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int has_check (FL_Timeout_Handler, void *=0)</td>
<td>Returns 1 if the check exists and has not been called yet, 0 otherwise.</td>
</tr>
<tr>
<td>static int has_idle (FL_Idle_Handler cb, void *=data=0)</td>
<td>Returns true if the specified idle callback is currently installed.</td>
</tr>
<tr>
<td>static int has_timeout (FL_Timeout_Handler, void *=0)</td>
<td>Returns true if the timeout exists and has not been called yet.</td>
</tr>
<tr>
<td>static int is_scheme (const char *=name)</td>
<td>Returns whether the current scheme is the given name.</td>
</tr>
<tr>
<td>static int lock ()</td>
<td>The lock() method blocks the current thread until it can safely access FLTK widgets and data.</td>
</tr>
<tr>
<td>static FL_Window * modal ()</td>
<td>Returns the top-most modal() window currently shown.</td>
</tr>
<tr>
<td>static FL_Window * next_window (const FL_Window *=)</td>
<td>Returns the next top-level window in the list of shown() windows.</td>
</tr>
<tr>
<td>static bool option (FL_Option opt)</td>
<td>FLTK library options management.</td>
</tr>
<tr>
<td>static void option (FL_Option opt, bool val)</td>
<td>Override an option while the application is running.</td>
</tr>
<tr>
<td>static void own_colormap ()</td>
<td>Makes FLTK use its own colormap.</td>
</tr>
<tr>
<td>static void paste (FL_Widget &amp;receiver)</td>
<td>Backward compatibility only.</td>
</tr>
<tr>
<td>static void paste (FL_Widget &amp;receiver, int source, const char *=type=FL::clipboard_plain_text)</td>
<td>Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.</td>
</tr>
<tr>
<td>static FL_Widget * pushed ()</td>
<td>Gets the widget that is being pushed.</td>
</tr>
<tr>
<td>static void pushed (FL_Window *=)</td>
<td>Sets the widget that is being pushed.</td>
</tr>
<tr>
<td>static FL_Widget * readqueue ()</td>
<td>Reads the default callback queue and returns the first widget.</td>
</tr>
<tr>
<td>static int ready ()</td>
<td>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.</td>
</tr>
<tr>
<td>static void redraw ()</td>
<td>Redraws all widgets.</td>
</tr>
<tr>
<td>static void release ()</td>
<td>Releases the current grabbed window, equals grab(0).</td>
</tr>
<tr>
<td>static void release_widget_pointer (FL_Widget *&amp;w)</td>
<td>Releases a widget pointer from the watch list.</td>
</tr>
<tr>
<td>static int reload_scheme ()</td>
<td>Called by scheme according to scheme name.</td>
</tr>
<tr>
<td>static void remove_check (FL_Timeout_Handler, void *=0)</td>
<td>Removes a check callback.</td>
</tr>
<tr>
<td>static void remove_clipboard_notify (FL_Clipboard_Notify_Handler h)</td>
<td>Stop calling the specified callback when there are changes to the selection buffer or the clipboard.</td>
</tr>
<tr>
<td>static void remove_fd (int)</td>
<td>Removes a file descriptor handler.</td>
</tr>
<tr>
<td>static void remove_fd (int, int when)</td>
<td>Removes a file descriptor handler.</td>
</tr>
<tr>
<td>static void remove_handler (FL_Event_Handler h)</td>
<td>Removes a previously added event handler.</td>
</tr>
</tbody>
</table>
• 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**, void ∗data=0)
  Removes a timeout callback.

• static void **repeat_timeout** (double t, **Fl_Timeout_Handler**, void ∗data=0)
  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

• static int **run** ()
  As long as any windows are displayed this calls Fl::wait() repeatedly.

• 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 number 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 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 void **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 **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 void **selection** (**Fl_Widget** &owner, const char ∗, 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 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)

  Sets the function to call to draw a specific boxtype.

- static void set_boxtype (Fl_Boxtype, Fl_Boxtype from)

  Copies the from boxtype.

- static void set_color (Fl_Color i, 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_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 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 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)

  See int Fl::wait()

- 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 * ,...) = ::error
  FLTK calls Fl::error() to output a normal error message.
- static void(* fatal)(const char *,...) = ::fatal
  FLTK calls Fl::fatal() to output a fatal error message.
- static char const * 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 *,...) = ::warning
  FLTK calls Fl::warning() to output a warning message.

9.2.1 Detailed Description

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

9.2.2 Member Enumeration Documentation

9.2.2.1 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 in a text widget will change focus to the next text widget. (This is considered ‘old’ behavior) When switched off (default), the cursor will stop at the end of the text. Pressing Tab or Ctrl-Tab will advance the keyboard focus. 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 inside the widget that will receive the next keystroke. If switched off, no such 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 from any text widget. If disabled, no dragging is possible, however dropping 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 tooltip text will open a little tooltip window until the mouse leaves the widget. If disabled, no tooltip is shown.</td>
</tr>
<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_LAST</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>
9.2.3 Member Function Documentation

9.2.3.1 abi_check()

```cpp
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()

9.2.3.2 abi_version()

```cpp
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.

9.2.3.3 add_check()

```cpp
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:

```cpp
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();
}
```

9.2.3.4 add_fd()

```cpp
static void Fl::add_fd (int fd,
                        int when,
                        Fl_FD_Handler cb, 
                        void * argp = 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.

The second version takes a when bitfield, with the bits FL_READ, FL_WRITE, and FL_EXCEPT defined, to indicate when the callback should be done.

---

Generated by Doxygen
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 any file descriptor can be monitored (files, devices, pipes, sockets, etc.). Due to limitations in Microsoft Windows, WIN32 applications can only monitor sockets.

### 9.2.3.5 add_idle()

```cpp
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.

### 9.2.3.6 add_timeout()

```cpp
void Fl::add_timeout (  
    double t,  
    Fl_Timeout_Handler cb,  
    void ∗ argp = 0 ) [static]
```

Adds a one-shot timeout callback.

The function will be called by `Fl::wait()` at `t` seconds after this function is called. The optional void+ argument is passed to the callback.

You can have multiple timeout callbacks. To remove a timeout callback use `Fl::remove_timeout()`.

If you need more accurate, repeated timeouts, use `Fl::repeat_timeout()` to reschedule the subsequent timeouts.

The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```cpp
#include <stdio.h>  
#include <FL/Fl.H>  
#include <FL/Fl_Window.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();  
}
```

### 9.2.3.7 api_version()

```cpp
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.

### 9.2.3.8 arg()

```cpp
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 WIN32 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().

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.

9.2.3.9  args() [1/2]

```c
void Fl::args {
  int argc,
  char ** argv 1  [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.
```

9.2.3.10  args() [2/2]

```c
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 \texttt{cb} lets you define your own switches. It is called with the same \texttt{argc} and \texttt{argv}, and with \texttt{i} set to the index of the switch to be processed. The \texttt{cb} handler should return zero if the switch is unrecognized, and not change \texttt{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. \texttt{i} should be incremented by the same amount.

The \texttt{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 \texttt{Fl::arg()} for descriptions of the standard switches.

On return \texttt{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).
- \texttt{argc}

The return value is \texttt{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 \texttt{argc}.

A usage string is displayed if \texttt{Fl::args()} detects an invalid argument on the command-line. You can change the message by setting the \texttt{Fl::help} pointer.

A very simple command line parser can be found in \texttt{examples/howto-parse-args.cxx}

The simpler \texttt{Fl::args(int argc, char **argv)} form is useful if your program does not have command line switches of its own.

### 9.2.3.11 background()

```cpp
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.

### 9.2.3.12 background2()

```cpp
void Fl::background2 (uchar r, uchar g, uchar b) [static]
```

Changes the alternative background color.

This color is used as a background by \texttt{Fl_Input} and other text widgets.

This call may change fl_color(FL_FOREGROUND_COLOR) if it does not provide sufficient contrast to \texttt{FL_\rightarrow BACKGROUND2_COLOR}.

### 9.2.3.13 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 inactive_r() or \texttt{c} otherwise.

### 9.2.3.14 box_dh()

```cpp
int Fl::box_dh (Fl_Boxtype t) [static]
```

Returns the height offset for the given boxtype.
See also

```c
box_dy()
```

### 9.2.3.15 box_dw()

```c
int Fl::box_dw (Fl_Boxtype t) [static]
```

Returns the width offset for the given boxtype.

See also

```c
box_dy()
```

### 9.2.3.16 box_dx()

```c
int Fl::box_dx (Fl_Boxtype t) [static]
```

Returns the X offset for the given boxtype.

See also

```c
box_dy()
```

### 9.2.3.17 box_dy()

```c
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 overdrawing 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:

```c
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().

### 9.2.3.18 check()

```c
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:

```c
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).

### 9.2.3.19 display()

```c
void Fl::display (const char * d) [static]
```

Sets the X display to use for all windows.

Actually this just sets the environment variable $DISPLAY to the passed string, so this only works before you show() the first window or otherwise open the display, and does nothing useful under WIN32.
9.2.3.20  **dnd_text_ops()** [1/2]

```cpp
static int Fl::dnd_text_ops ( ) [inline], [static]
```

Gets or 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.

9.2.3.21  **dnd_text_ops()** [2/2]

```cpp
static void Fl::dnd_text_ops ( int v ) [inline], [static]
```

Gets or 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.

9.2.3.22  **draw_box_active()**

```cpp
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)

9.2.3.23  **flush()**

```cpp
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 trigger a redraw event, it should instead call Fl::awake() to get the main thread to process the event queue.

9.2.3.24  **get_system_colors()**

```cpp
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.

9.2.3.25  **gl_visual()**

```cpp
int Fl::gl_visual ( int mode, 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.

9.2.3.26  **is_scheme()**

```cpp
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)

9.2.3.27 option() [1/2]

bool Fl::option ( Fl_Option opt ) [static]

FLTK 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:

```c
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 |

Returns

true or false

See also

enum Fl::Fl_Option
Fl::option(Fl::Option, bool)
Since
FLTK 1.3.0

9.2.3.28  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)

9.2.3.29  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.

9.2.3.30  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()

9.2.3.31  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();
    }
}
308 Class Documentation

Fl::check();
if (user_hit_abort_button()) break;
}

9.2.3.32 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*)

9.2.3.33 reload_scheme()

int Fl::reload_scheme () [static]
Called by scheme according to scheme name.
Loads or reloads the current scheme selection. See void scheme(const char *name)

9.2.3.34 remove_check()

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.

9.2.3.35 remove_timeout()

void Fl::remove_timeout (
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
Removes a timeout callback.
It is harmless to remove a timeout callback that no longer exists.

Note

    This version removes all matching timeouts, not just the first one. This may change in the future.

9.2.3.36 repeat_timeout()

void Fl::repeat_timeout (
    double t,
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
You may only call this method inside a timeout callback.
The following code will print "TICK" each second on stdout with a fair degree of accuracy:

    void callback(void*)
    {
        puts("TICK");
        Fl::repeat_timeout(1.0, callback);
    }

int main()
{
    Fl::add_timeout(1.0, callback);
    return Fl::run();
}
9.2.3.37 run()

```cpp
int Fl::run ( ) [static]
```

As long as any windows are displayed this calls Fl::wait() repeatedly.
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();.

9.2.3.38 scheme()

```cpp
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 Mac OS X
- "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).

Uppercase scheme names are equivalent, but the stored scheme name will always be lowercase and Fl::scheme()
will return this lowercase name.
If the resulting scheme name is not defined, the default scheme will be used and Fl::scheme() will return NULL.

See also

Fl::is_scheme()

9.2.3.39 scrollbar_size() [1/2]

```cpp
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.

9.2.3.40 scrollbar_size() [2/2]

```cpp
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 wid-
gets.

Parameters

| in | W | The new default size for widget scrollbars, in pixels. |

9.2.3.41 set_box_color()

```cpp
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 \texttt{fl\_color()} depending on the widget's state to either \texttt{c} or \texttt{fl\_inactive(c)}.
It should be used whenever a box background is drawn in the box (type) drawing code instead of calling
\texttt{fl\_color(Fl\_Color bg)} with the background color \texttt{bg}, usually \texttt{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 inactive \texttt{r()} then the
internal variable is false (0), otherwise it is true (1). This is faster than calling \texttt{Fl\_Widget::active\_r()} because the
state is cached.

See also
\begin{itemize}
\item \texttt{Fl::draw\_box\_active()}
\item \texttt{Fl::box\_color(Fl\_Color)}
\end{itemize}

9.2.3.42 set_idle()

static void Fl::set_idle (  
    Fl\_Old\_Idle\_Handler cb ) [inline], [static]

Sets an idle callback.

\textbf{Deprecated} This method is obsolete - use the \texttt{add_idle()} method instead.

9.2.3.43 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

9.2.3.44 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

9.2.3.45 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.

\textbf{Deprecated} Use int Fl::api\_version() instead.

9.2.3.46 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.

9.2.3.47 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.
9.2.3.48 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:

- 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.

- Fl::visual(FL_DOUBLE|FL_INDEX)
  Hardware double buffering. Call this if you are going to use Fl_Double_Window.

- Fl::visual(FL_DOUBLE|FL_RGB)

- Fl::visual(FL_DOUBLE|FL_RGB8)
  Hardware double buffering and full color.

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).

9.2.3.49 wait()

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. The return value of Fl::wait() is non-zero if there are any visible windows - this may change in future versions of FLTK.

Fl::wait(time) waits a maximum of time seconds. It can return much sooner if something happens.

The return value is positive if an event or fd happens before the time elapsed. It is zero if nothing happens (on Win32 this will only return zero if time is zero). It is negative if an error occurs (this will happen on UNIX if a signal happens).

9.2.4 Member Data Documentation

9.2.4.1 help

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.

9.2.4.2 idle

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
- Fl_add_idle.cxx
- Fl_arg.cxx
- Fl_boxtype.cxx
- Fl_color.cxx
- Fl_color_mac.cxx
- Fl_color_win32.cxx
- Fl-compose.cxx
- Fl_display.cxx
- Fl_dnd_win32.cxx
- Fl_dnd_x.cxx
- Fl_get_key.cxx
- Fl_get_key_mac.cxx
- Fl_get_key_win32.cxx
- Fl_get_system_colors.cxx
- Fl_grab.cxx
- Fl_labeltype.cxx
- Fl_lock.cxx
- Fl_own_colormap.cxx
- Fl_set_font.cxx
- Fl_set_fonts_mac.cxx
- Fl_set_fonts_win32.cxx
- Fl_set_fonts_x.cxx
- Fl_set_fonts_xft.cxx
- FlShortcut.cxx
- Fl_visual.cxx
- Fl_Widget.cxx
- Fl_Window.cxx
- gl_start.cxx
- screen_xywh.cxx
- Fl_Cairo.cxx

9.3 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.

#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 ()
- 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 Fl_Group ∗ as_group ()

  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.
- 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, 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)

  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.

- **const Fl_Image *deimage () const**

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image *image () const**

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

- `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 (void *v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  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 () 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 ()
  Draws the widget.
• int handle (int)
  Handles the specified event.
• void value_damage ()
  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 ()
  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_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 ∗cb, void ∗d)**
  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**, **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_ACCECTS_MENU = 1<<19**,
  - **USERFLAG3 = 1<<29**, **USERFLAG2 = 1<<30**, **USERFLAG1 = 1<<31**}

  flags possible values enumeration.

### 9.3.1 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.
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.

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 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.

### 9.3.3 Member Function Documentation

#### 9.3.3.1 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements \Fl_Widget\.

#### 9.3.3.2 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> 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.

### Fl_Adjuster::soft()

#### 1/2

```cpp
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.

#### 2/2

```cpp
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.

### Fl_Adjuster::value_damage()

```cpp
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

### Fl_Bitmap Class Reference

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.

```cpp
#include <Fl_Bitmap.H>
```

Inheritance diagram for Fl_Bitmap:

```
Fl_Image
  ↓
Fl_Bitmap
  ↓
Fl_XBM_Image
```

#### Public Member Functions

- **Fl_Image** → **copy ()**
  - `virtual Fl_Image * copy (int W, int H)`
    - The copy() method creates a copy of the specified image.
  - `void draw (int X, int Y)
  - `virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)`
Draws the image with a bounding box.

- **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 W, int H)
  The constructors create a new bitmap from the specified bitmap data.

- virtual void label (Fl_Menu_Item ∗m)
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void label (Fl_Widget ∗w)
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void uncache ()
  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 void color_average (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the FLTK color value c.

- Fl_Image ∗ copy ()
  The copy() method creates a copy of the specified image.

- int count () const
  The count() method returns the number of data values associated with the image.

- int d () const
  Returns the current image depth.

- const char ∗const ∗ data () const
  Returns a pointer to the current image data array.

- virtual void desaturate ()
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)
  Draws the image.

- int fail ()
  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 height in pixels.

- 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.

- int w () const
  Returns the current image width in pixels.

- 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
9.4 Fl_Bitmap Class Reference

Friends

- class Fl_GDI_Graphics_Driver
- class Fl_GDI_Printer_Graphics_Driver
- class Fl_Quartz_Graphics_Driver
- class Fl_Xlib_Graphics_Driver

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- 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 Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_NO_IMAGE = -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 array 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.
- void h (int H)
  
  Sets the current image height in pixels.
- void ld (int LD)
  
  Sets the current line data size in bytes.
- void w (int W)
  
  Sets the current image width in pixels.

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)

9.4.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images. Images are drawn using the current color.

9.4.2 Member Function Documentation

9.4.2.1 copy()

FL_IMAGE * Fl_Bitmap::copy (  
  int W,  
  int H ) [virtual]

The copy() method creates a copy of the specified image. If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it. Reimplemented from Fl_Image.
9.4.2.2 draw()

```
void Fl_Bitmap::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image 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`.

9.4.2.3 label() [1/2]

```
void Fl_Bitmap::label (  
    Fl_Menu_Item * m ) [virtual]
```

The label() methods are an obsolete way to set the image attribute of a widget or menu item. Use the image() or deimage() methods of the `Fl_Widget` and `Fl_Menu_Item` classes instead.

Reimplemented from `Fl_Image`.

9.4.2.4 label() [2/2]

```
void Fl_Bitmap::label (  
    Fl_Widget * widget ) [virtual]
```

The label() methods are an obsolete way to set the image attribute of a widget or menu item. Use the image() or deimage() methods of the `Fl_Widget` and `Fl_Menu_Item` classes instead.

Reimplemented from `Fl_Image`.

9.4.2.5 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`

9.5 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
   |
   v
Fl_RGB_Image
   |
   v
Fl_BMP_Image
```

Generated by Doxygen
Public Member Functions

- `Fl_BMP_Image` (const char *filename)
  The constructor loads the named BMP image from the given bmp filename.

Public Member Functions inherited from `Fl_RGB_Image`

- virtual void color_average (FL_Color c, float i)
  The color_average() method averages the colors in the image with the FLTK color value c.

- `Fl_Image * copy ()
  The copy() method creates a copy of the specified image.

- virtual void desaturate ()
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)
  The draw() method draws the image 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 W, int H, int D=3, int LD=0)
  The constructor creates a new image from the specified data.

- virtual void label (Fl_Menu_Item *m)
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void label (Fl_Widget *w)
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void uncache ()
  If the image has been cached for display, delete the cache data.

- virtual ~Fl_RGB_Image ()
  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from `Fl_Image`

- `Fl_Image * copy ()
  The copy() method creates a copy of the specified image.

- int count () const
  The count() method returns the number of data values associated with the image.

- int d () const
  Returns the current image depth.

- const char *const *data () const
  Returns a pointer to the current image data array.

- void draw (int X, int Y)
  Draws the image.

- int fail ()
  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 height in pixels.

- 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.

- `int w () const`
  Returns the current image width in pixels.
- `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_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).

### 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_NO_IMAGE = -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 array 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.
- `void h (int H)`
  Sets the current image height in pixels.
- `void ld (int LD)`
  Sets the current line data size in bytes.
- `void w (int W)`
  Sets the current image width in pixels.

### 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)`
9.5.1 Detailed Description

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 Fl_BMP_Image()

Fl_BMP_Image::Fl_BMP_Image (const char * bmp)

The 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.
The documentation for this class was generated from the following files:

- Fl_BMP_Image.H
- Fl_BMP_Image.cxx

9.6 Fl_Box Class Reference

This widget simply draws its box, and possibly its label.
#include <Fl_Box.H>

Inheritance diagram for Fl_Box:

```
Fl_Widget
        Fl_Box
```

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)
- virtual int handle (int)

  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

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 Fl_Gl_Window * as_gl_window ()
  
  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_Window * as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- 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, 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)
  
  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.
• const Fl_Image * deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image * image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.

- **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.

- **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 (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  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 () 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 ()
  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 of 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 ()
  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_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 ∗cb, void ∗d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.6.1 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.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 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.

9.6.3 Member Function Documentation

9.6.3.1 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

9.6.3.2 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.

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_Box.H`
- `Fl_Box.cxx`

# 9.7 Fl_Browser Class Reference

The `Fl_Browser` widget displays a scrolling list of text lines, and manages all the storage for the text.

```c
#include <Fl_Browser.H>
```

Inheritance diagram for `Fl_Browser`:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Browser_
  ↓
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 char)
  Gets the current column separator character.

- **void column_char**(char c)
  Sets the column separator to c.

- **const int *column_widths**(const int *)
  Gets the current column width array.

- **void column_widths**(const int *, int line) const
  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 *)
  The constructor makes an empty browser.

- **char format_char**(const char)
  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**()
  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 *, int val=1)
  For back compatibility.

- **int load**(const char *)
  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)
  Generated by Doxygen
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 ()**
  - 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)
  - 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).
9.7 Fl_Browser Class Reference

- `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 position () const`
  Gets the vertical scroll position of the list as a pixel position `pos`.

- `void position (int pos)`
  Sets the vertical scroll position of the list to pixel position `pos`.

- `void resize (int X, int Y, int W, int H)`
  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`
  This method has been deprecated, existing for backwards compatibility only.

- `void scrollbar_width (int width)`
  This method has been deprecated, existing for backwards compatibility only.

- `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`.

**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.

• virtual **Fl_Group** + as_group ()
  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 *)**

• void **forms_end ()**
  This is for forms compatibility only.

• int **handle (int)**
  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
  See void **Fl_Group::resizable(Fl_Widget *box)**

• void **resizable (Fl_Widget &o)**
  See void **Fl_Group::resizable(Fl_Widget *box)**

• void **resizable (Fl_Widget *)**
  The resizable widget defines the resizing box for the group.

• void **resize (int, int, int, int)**
  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 Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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)
  
  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.
- const Fl_Image *deimage () const
  void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
- void do_callback ()
  Calls the widget callback.
- void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
- void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
- 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.
- const Fl_Image *image () const
- void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
- 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.

• 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 ()

Generated by Doxygen
Enables keyboard focus navigation with this widget.

- 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 **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 ∗ **topWindow** () const
  
  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ **topWindow_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 ∗ **userData** () const
  
  Gets the user data for this widget.

- void **userData** (void ∗ v)
  
  Sets the user data for this widget.

- unsigned int **visible** () const
  
  Returns whether a widget is visible.

- unsigned int **visiblefocus** ()
  
  Checks whether this widget has a visible focus.

- void **visiblefocus** (int v)
  
  Modifies keyboard focus navigation.

- int **visibler** () 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
  The height of the entire list of all visible() items in pixels.
- int incr_height () const
  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
  Return the item at specified line.
- void item_draw (void ∗item, int X, int Y, int W, int H) const
  Draws item at the position specified by XYWH.
- void ∗item_first () const
  Returns the very first item in the list.
- int item_height (void ∗item) const
  Returns height of item in pixels.
- void ∗item_last () const
  Returns the very last item in the list.
- void ∗item_next (void ∗item) const
  Returns the next item after item.
- void ∗item_prev (void ∗item) const
  Returns the previous item before item.
- void item_select (void ∗item, int val)
  Change the selection state of item to the value val.
- int item_selected (void ∗item) const
  See if item is selected.
- void item_swap (void ∗a, void ∗b)
  Swap the items a and b.
- const char ∗item_text (void ∗item) const
  Returns the label text for item.
- int item_width (void ∗item) const
  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 ()
  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

- void draw ()
  
  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
**draw_focus** ()

Draws a focus rectangle around the widget.

**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.

**draw_label** () const

Draws the widget's label at the defined label position.

**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.

**flags** () const

Gets the widget flags mask.

**h**(int v)

Internal use only.

**set_flag**(unsigned int c)

Sets a flag in the flags mask.

**w**(int v)

Internal use only.

**x**(int v)

Internal use only.

**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 ∗cb, void ∗d)

  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, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
};
```

flags possible values enumeration.

9.7.1 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. 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).

9.7.2 Constructor & Destructor Documentation

9.7.2.1 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>
9.7.3 Member Function Documentation

9.7.3.1 _remove()

```cpp
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()

9.7.3.2 add()

```cpp
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()

9.7.3.3 bottomline()

```cpp
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()

9.7.3.4 clear()

```cpp
void Fl_Browser::clear ( )
```

Generated by Doxygen
Removes all the lines in the browser.

See also

add(), insert(), remove(), swap(int,int), clear()

9.7.3.5 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()

9.7.3.6 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()

9.7.3.7 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:

```
Fl_Browser *b = new Fl_Browser(...);
static int widths[] = { 50, 50, 50, 70, 70, 48, 40, 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\tCPU\tMEM\tVSZ\tRSS\tTTY\tSTAT\tSTART\tTIME\tCOMMAND");
b->add("root\t2888\t0.0\t0.0\t1352\t0\t0.0\troot/sbin/mingetty tty3");
b->add("root\t13115\t0.0\t0.0\t1352\t0\t0.0\troot/sbin/mingetty tty2");
```

See also

column_char(), column_widths()

9.7.3.8 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()
9.7.3.9 data() [1/2]

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)

9.7.3.10 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) |

9.7.3.11 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()

9.7.3.12 displayed()

int Fl_Browser::displayed (  
    int line  
)

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
topline(), middleline(), bottomline(), displayed(), lineposition()

### 9.7.3.13 find_line()

```cpp
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**

| item | that was found. |
| NULL | if line is out of range. |

See also
item_at(), find_line(), lineno()

### 9.7.3.14 format_char() [1/2]

```cpp
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
- '@@' Print rest of line starting with '@'
- '@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
- '@B0', '@B1', ... '@B255' Fill the backgound 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

Generated by Doxygen
• '@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.

9.7.3.15 format_char()  [2/2]

```cpp
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

9.7.3.16 full_height()

```cpp
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_`.

9.7.3.17 hide()  [1/2]

```cpp
void Fl_Browser::hide ( ) [inline], [virtual]
```

Hides the entire `Fl_Browser` widget — opposite of `show()`.
Reimplemented from `Fl_Widget`.

9.7.3.18 hide()  [2/2]

```cpp
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()`
9.7.3.19 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.

9.7.3.20 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. |

9.7.3.21 incr_height()

```
int Fl_Browser::incr_height ( ) const [protected], [virtual]
```

The default 'average' item height (including inter-item spacing) in pixels. This currently returns textsize() + 2.

**Returns**

The value in pixels.

See also

item_height(), item_width(), incr_height(), full_height()

Reimplemented from Fl_Browser_.

9.7.3.22 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.
### 9.7.3.23 insert() [2/2]

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>line</code></td>
<td>The new line will be inserted above this line (1 based).</td>
</tr>
<tr>
<td><code>item</code></td>
<td>The item to be added.</td>
</tr>
</tbody>
</table>

### 9.7.3.24 item_at()

```cpp
void * Fl_Browser::item_at (  
    int line ) const [inline], [protected], [virtual]
```

Return the item at specified line.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>line</code></td>
<td>The line of the item to return. (1 based)</td>
</tr>
</tbody>
</table>

**Returns**

The item, or NULL if line out of range.

**See also**

- `item_at()`, `find_line()`, `lineno()`

Reimplemented from `Fl_Browser_`.

### 9.7.3.25 item_draw()

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item to be drawn</td>
</tr>
<tr>
<td><code>X, Y, W, H</code></td>
<td>position and size.</td>
</tr>
</tbody>
</table>
9.7 Fl_Browser Class Reference

Implements Fl_Browser_.

9.7.3.26 item_first()

```cpp
def Fl_Browser::item_first() const [protected], [virtual]
```

Returns the very first item in the list.

Example of use:
```
// 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_.

9.7.3.27 item_height()

```cpp
def 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_.

9.7.3.28 item_last()

```cpp
def Fl_Browser::item_last() const [protected], [virtual]
```

Returns the very last item in the list.

Example of use:
```
// 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
9.7.3.29  item_next()

```cpp
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_`.

9.7.3.30  item_prev()

```cpp
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_`.

9.7.3.31  item_select()

```cpp
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
\begin{verbatim}
select(), selected(), value(), item_select(), item_selected()
\end{verbatim}
Reimplemented from Fl_Browser_.

9.7.3.32 item_selected()

\begin{verbatim}
int Fl_Browser::item_selected ( void ∗ item) const [protected], [virtual]
\end{verbatim}
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
\begin{verbatim}
select(), selected(), value(), item_select(), item_selected()
\end{verbatim}
Reimplemented from Fl_Browser_.

9.7.3.33 item_swap()

\begin{verbatim}
void Fl_Browser::item_swap ( void ∗ a,
    void ∗ b ) [inline], [protected], [virtual]
\end{verbatim}
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
\begin{verbatim}
swap(int,int), item_swap()
\end{verbatim}
Reimplemented from Fl_Browser_.

9.7.3.34 item_text()

\begin{verbatim}
const char ∗ Fl_Browser::item_text ( void ∗ item) const [protected], [virtual]
\end{verbatim}
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
9.7.3.35  item_width()

```cpp
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*.

9.7.3.36  lineno()

```cpp
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()*

9.7.3.37  lineposition()

```cpp
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()*
9.7.3.38  load()

```cpp
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**

- `in filename` The filename to load

**Returns**

1 if OK, 0 on error (errno has reason)

**See also**

- `add()`

9.7.3.39  make_visible()

```cpp
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**

- `in line` The line to be made visible.

**See also**

- `show(int), hide(int), display(), visible(), make_visible()`

9.7.3.40  middleline()

```cpp
void Fl_Browser::middleline (  
    int line  
) [inline]
```

Scrolls the browser so the middle item in the browser is showing the specified line.

**Parameters**

- `in line` The line to be displayed in the middle.

**See also**

- `topline(), middleline(), bottomline(), displayed(), lineposition()`

9.7.3.41  move()

```cpp
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>

9.7.3.42 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()`

9.7.3.43 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. |

9.7.3.44 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()`
9.7.3.45 selected()

int Fl_Browser::selected (  
    int line ) const  
Returns 1 if specified line is selected, 0 if not.

Parameters

| in | line | The line being checked (1 based) |

Returns

1 if item selected, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

9.7.3.46 show() [1/2]

void Fl_Browser::show ( ) [inline], [virtual]  
Shows the entire Fl_Browser widget – opposite of hide().  
Reimplemented from Fl_Widget.

9.7.3.47 show() [2/2]

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

| in | line | The line to be shown. (1 based) |

See also

show(int), hide(int), display(), visible(), make_visible()

9.7.3.48 size()

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.

9.7.3.49 swap() [1/2]

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

| in | a, b | The two items to be swapped. |
See also
**swap(int,int), item_swap()**

### 9.7.3.50 swap() [2/2]

```cpp
void Fl_Browser::swap (  
    int a,  
    int b  
)
```

Swaps two browser lines a and b. You must call **redraw()** to make any changes visible.

**Parameters**

- **in** `a,b` The two lines to be swapped. (both 1 based)

See also
**swap(int,int), item_swap()**

### 9.7.3.51 text() [1/2]

```cpp
const char * Fl_Browser::text (  
    int line ) const
```

Returns the label text for the specified line. Return value can be NULL if line is out of range or unset. The parameter line is 1 based.

**Parameters**

- **in** `line` The line number of the item whose text is returned. (1 based)

**Returns**

The text string (can be NULL)

### 9.7.3.52 text() [2/2]

```cpp
void Fl_Browser::text (  
    int line,  
    const char * newtext  
)
```

Sets the text for the specified line to newtext. 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. Does nothing if line is out of range.

**Parameters**

- **in** `line` The line of the item whose text will be changed. (1 based)
- **in** `newtext` The new string to be assigned to the item.

### 9.7.3.53 textsize()

```cpp
void Fl_Browser::textsize (  
    Fl_Fontsize newSize  
)
```

Sets the default text size (in pixels) for the lines in the browser to 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.

9.7.3.54 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.

9.7.3.55 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

| in line | The line to be displayed at the top. |

See also
topline(), middleline(), bottomline(), displayed(), lineposition()

9.7.3.56 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()

9.7.3.57 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()

9.7.3.58 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

| in | line | The line in the browser to be tested. (1 based) |

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

9.8 Fl_Browser_ Class Reference

This is the base class for browsers.

### Inheritance diagram for Fl_Browser_:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Browser_
    ↓
Fl_Browser Fl_Check_Browser
    ↓
Fl_File_Browser Fl_Hold_Browser Fl_Multi_Browser Fl_Select_Browser
```

### Public Types

- enum {
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7
}

Values for hasScrollbar().

### 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)
  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 position () const`
  
  Gets the vertical scroll position of the list as a pixel position `pos`.

- `void position (int pos)`
  
  Sets the vertical scroll position of the list to pixel position `pos`.

- `void resize (int X, int Y, int W, int H)`
  
  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`
  
  This method has been deprecated, existing for backwards compatibility only.

- `void scrollbar_width (int width)`
  
  This method has been deprecated, existing for backwards compatibility only.

- `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 `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`.

### 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.

- `virtual Fl_Group ∗as_group ()`
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 *)

- **void** forms_end ()
  This is for forms compatibility only.

- **int** handle (int)
  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 *)
  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
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.

- **void** resize (int, int, int, int)
  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 Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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)
  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
  Generated by Doxygen
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.

- const Fl_Image *deimage () const

- void deimage (Fl_Image &img)

- void deimage (Fl_Image *img)

- void do_callback ()
  
  Calls the widget callback.

- void do_callback (Fl.Widget *o, long arg)
  
  Calls the widget callback.

- void do_callback (Fl.Widget *o, void *arg=0)
  
  Calls the widget callback.

- 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.

- const Fl_Image *image () const

- void image (Fl_Image &img)

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label.
• 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.
• 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.
- 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 (void *v)
  Sets the user data for this widget.
- unsigned int visible () const
  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 () 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 ()
  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.

- **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

- **void draw ()**
  
  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 ()**
  
  draws a focus rectangle around the widget
• void **draw_focus**(FL_Boxtypet, intx, int y, int w, int h) 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 ∗cb, void ∗d)
  
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

  flags possible values enumeration.
9.8.1 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.

9.8.1.1 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.

9.8.2 Member Enumeration Documentation

9.8.2.1 anonymous enum

```
nanonymous 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
```

<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</td>
</tr>
<tr>
<td></td>
<td>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>

9.8.3 Constructor & Destructor Documentation

9.8.3.1 Fl_Browser_()

```
Fl_Browser_::Fl_Browser_ (int X, int Y,
```
The constructor makes an empty browser.

Parameters

| in | X, Y, W, H | position and size. |
| in | L          | The label string, may be NULL. |

### 9.8.4 Member Function Documentation

#### 9.8.4.1 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

| out | X, Y, W, H | The returned bounding box. |
|     |           | (The original contents of these parameters are overwritten) |

#### 9.8.4.2 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

| in | item | The item being deleted. |

#### 9.8.4.3 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

| in | docallbacks | If 1, invokes widget callback if item changed. |
|    |            | If 0, doesn't do callback (default). |

#### 9.8.4.4 display()

```cpp
void Fl_Browser_::display {
    void * item )
```
Displays the item, scrolling the list as necessary.

Parameters

| in | item | The item to be displayed. |

See also

display(), displayed()

### 9.8.4.5 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

| in | item | The item to check |

Returns

1 if visible, 0 if not visible.

See also

display(), displayed()

### 9.8.4.6 draw()

```cpp
void Fl_Browser_::draw (void ) [protected], [virtual]
```

Draws the list within the normal widget bounding box. Implements Fl_Widget.

### 9.8.4.7 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

| in | ypos | The y position (eg. Fl::event_y()) to find an item under. |

Returns

The item, or NULL if not found

### 9.8.4.8 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.

### 9.8.4.9 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.

### 9.8.4.10 handle()

```cpp
int Fl_Browser_::handle ( int event ) [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_Widget.
Reimplemented in Fl_Check_Browser.

### 9.8.4.11 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.
9.8.4.12 hposition() [1/2]

```cpp
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

- `position()`, `hposition()`

9.8.4.13 hposition() [2/2]

```cpp
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

| in     | `pos` | The horizontal position (in pixels) to scroll the browser to. |

See also

- `position()`, `hposition()`

9.8.4.14 incr_height()

```cpp
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.

9.8.4.15 inserting()

```cpp
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

| in     | `a`  | The starting item position |
|        | `b`  | The new item being inserted |

9.8.4.16 item_at()

```cpp
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`. 

Generated by Doxygen
Parameters

\textbf{in} \hspace{0.5cm} \textit{index} \hspace{0.5cm} The index of the item to be returned

Returns

The item at the specified index.

Reimplemented in \texttt{Fl\_Browser}.

\textbf{9.8.4.17} \textbf{item\_draw()}

\begin{verbatim}
virtual void Fl_Browser_::item_draw ( 
    void * item, 
    int \textit{X}, 
    int \textit{Y}, 
    int \textit{W}, 
    int \textit{H} ) const [protected], [pure virtual]
\end{verbatim}

This method must be provided by the subclass to draw the item in the area indicated by \textit{X}, \textit{Y}, \textit{W}, \textit{H}.

Implemented in \texttt{Fl\_Browser}.

\textbf{9.8.4.18} \textbf{item\_first()}

\begin{verbatim}
virtual void * Fl_Browser_::item_first ( ) const [protected], [pure virtual]
\end{verbatim}

This method must be provided by the subclass to return the first item in the list.

See also

\texttt{item\_first()}, \texttt{item\_next()}, \texttt{item\_last()}, \texttt{item\_prev()}

Implemented in \texttt{Fl\_Browser}.

\textbf{9.8.4.19} \textbf{item\_height()}

\begin{verbatim}
virtual int Fl_Browser_::item_height ( 
    void * \textit{item} ) const [protected], [pure virtual]
\end{verbatim}

This method must be provided by the subclass to return the height of \textit{item} in pixels.

Allow for two additional pixels for the list selection box.

Parameters

\textbf{in} \hspace{0.5cm} \textit{item} \hspace{0.5cm} The item whose height is returned.

Returns

The height of the specified \textit{item} in pixels.

See also

\texttt{item\_height()}, \texttt{item\_width()}, \texttt{item\_quick\_height()}

Implemented in \texttt{Fl\_Browser}.

\textbf{9.8.4.20} \textbf{item\_last()}

\begin{verbatim}
virtual void * Fl_Browser_::item_last ( ) const [inline], [protected], [virtual]
\end{verbatim}

This method must be provided by the subclass to return the last item in the list.

See also

\texttt{item\_first()}, \texttt{item\_next()}, \texttt{item\_last()}, \texttt{item\_prev()}

Reimplemented in \texttt{Fl\_Browser}.
9.8.4.21  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_Browser.

9.8.4.22  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_Browser.

9.8.4.23  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.

9.8.4.24  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_Browser.
9.8.4.25 item_selected()

```c
int Fl_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 item.
The method should return 1 if item is selected, or 0 otherwise.

**Parameters**

```c
ing item The item to test.
```

Reimplemented in Fl_Browser.

9.8.4.26 item_swap()

```c
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**

```c
ing a,b The two items to be swapped.
```

Reimplemented in Fl_Browser.

9.8.4.27 item_text()

```c
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**

```c
ing item The item whose label text is returned.
```

**Returns**

The item's text label. (Can be NULL if blank)

Reimplemented in Fl_Browser.

9.8.4.28 item_width()

```c
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**

```c
ing item The item whose width is returned.
```

**Returns**

The width of the item in pixels.

Implemented in Fl_Browser.
9.8.4.29 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()`

9.8.4.30 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.

9.8.4.31 position()[1/2]

`int Fl_Browser_::position ( ) const [inline]`

Gets the vertical scroll position of the list as a pixel position `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()`

9.8.4.32 position()[2/2]

`void Fl_Browser_::position ( int pos )`

Sets the vertical scroll position of the list to pixel position `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  | pos | The vertical position (in pixels) to scroll the browser to. |

See also

`position(), hposition()`

9.8.4.33 redraw_line()

`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()

9.8.4.34 redraw_lines()

void Fl_Browser_::redraw_lines ( ) [inline], [protected]
This method will cause the entire list to be redrawn.
See also

redraw_lines(), redraw_line()

9.8.4.35 replacing()

void Fl_Browser_::replacing ( 
    void * a,
    void * b ) [protected]
This method should be used when item \texttt{a} is being replaced by item \texttt{b}.
It allows the \texttt{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.

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{\texttt{a}} & Item being replaced \\
\hline
\textbf{in} & \textbf{\texttt{b}} & Item to replace \texttt{\texttt{a}} \\
\hline
\end{tabular}

Parameters

9.8.4.36 resize()

void Fl_Browser_::resize ( 
    int \texttt{X},
    int \texttt{Y},
    int \texttt{W},
    int \texttt{H} ) [virtual]
Repositions and/or resizes the browser.

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{in} & \texttt{X, Y, W, H} & The new position and size for the browser, in pixels. \\
\hline
\end{tabular}

Reimplemented from \texttt{Fl_Widget}.

9.8.4.37 scrollbar_left()

void Fl_Browser_::scrollbar_left ( ) [inline]
Moves the vertical scrollbar to the lefthand side of the list.
For back compatibility.

9.8.4.38 scrollbar_right()

void Fl_Browser_::scrollbar_right ( ) [inline]
Moves the vertical scrollbar to the righthand side of the list.
For back compatibility.

9.8.4.39 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)

9.8.4.40 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

in newSize  Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

9.8.4.41 scrollbar_width() [1/2]

int Fl_Browser_::scrollbar_width ( ) const [inline]

This method has been deprecated, existing for backwards compatibility only. Use scrollbar_size() instead. This method always returns the global value Fl::scrollbar_size().

Returns

Always returns the global value Fl::scrollbar_size().

Todo  This method should eventually be removed in 1.4+

9.8.4.42 scrollbar_width() [2/2]

void Fl_Browser_::scrollbar_width ( int width ) [inline]

This method has been deprecated, existing for backwards compatibility only. Use scrollbar_size(int) instead. This method sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

Todo  This method should eventually be removed in 1.4+

9.8.4.43 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 1, 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.

9.8.44 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 1, invokes widget callback if item changed. If 0, doesn’t do callback (default).</td>
</tr>
</tbody>
</table>

9.8.45 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.

9.8.46 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*, 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></td>
</tr>
<tr>
<td></td>
<td>Values other than the above will cause undefined behavior</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other flags may appear in the future.</td>
<td></td>
</tr>
</tbody>
</table>

Todo Add a flag to ignore case
9.8.4.47 swapping()

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.

9.8.4.48 textfont()

Fl_Font Fl_Browser_::textfont ( ) const [inline]

Gets the default text font for the lines in the browser.

See also

textfont(), textsize(), texcolor()

9.8.5 Member Data Documentation

9.8.5.1 hscrollbar

Fl_Scrollbar Fl_Browser_::hscrollbar

Horizontal scrollbar. Public, so that it can be accessed directly.

9.8.5.2 scrollbar

Fl_Scrollbar Fl_Browser_::scrollbar

Vertical scrollbar. Public, so that it can be accessed directly.

The documentation for this class was generated from the following files:

- Fl_Browser_.H
- Fl_Browser_.cxx

9.9 Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

#include <Fl_Button.H>

Inheritance diagram for Fl_Button:
Public Member Functions

- `int clear ()`
  Same as `value(0)`.
- `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.
- `virtual int handle (int)`
  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 Fl_Group ∗ as_group ()`
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.

- 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, 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)
  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)
  Copies 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.
• const Fl_Image * **deimage** () const
  Sets the image to use as part of the widget label.
• void **deimage** (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void **deimage** (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void **do_callback** ()
  Calls the widget callback.
  Calls the widget callback.
  Calls the widget callback.
• 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.
• const Fl_Image * **image** () const
  Sets the image to use as part of the widget label.
• void **image** (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void **image** (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
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.
• 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.
• 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 (void ∗v)
  Sets the user data for this widget.

• **unsigned int** visible () const
  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 () 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 **void** draw ()
  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 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 ()`
  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_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 ∗ cb, void ∗ d)`
  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 }`
9.9.1 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()`. 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.

**Todo** Refactor the doxygen comments for `Fl_Button type()` documentation.

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.

**Todo** Refactor the doxygen comments for `Fl_Button when()` documentation.

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.
- `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).

9.9.2 Constructor & Destructor Documentation

9.9.2.1 `Fl_Button()`

```cpp
default public:
  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. 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.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>X,Y,W,H</code></td>
</tr>
<tr>
<td>in</td>
<td><code>L</code></td>
</tr>
</tbody>
</table>
9.9 Fl_Button Class Reference

9.9.3 Member Function Documentation

9.9.3.1 clear()

```cpp
int Fl_Button::clear ( ) [inline]
```

Same as `value(0)`.

See also

```
value(int v)
```

9.9.3.2 down_box() [1/2]

```cpp
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
```

9.9.3.3 down_box() [2/2]

```cpp
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
```

9.9.3.4 draw()

```cpp
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:

```
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw();            // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

Reimplemented in Fl_Light_Button, and Fl_Return_Button.

9.9.3.5 handle()

```cpp
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.
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

Fi_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Light_Button, Fl_Repeat_Button, and Fl_Return_Button.

9.9.3.6 set()

```cpp
template <class T> int Fl_Button::set ( ) const [inline]
```

Same as `value(1)`.

See also

value(int v)

9.9.3.7 shortcut() [1/2]

```cpp
template <class T> int Fl_Button::shortcut ( ) const [inline]
```

Returns the current shortcut key for the button.

Return values

| int |

9.9.3.8 shortcut() [2/2]

```cpp
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 |

9.9.3.9 value()

```cpp
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

\[ \text{in} \ v \ \text{button value.} \]

See also

set(), clear()

The documentation for this class was generated from the following files:

- Fl_Button.H
- Fl_Button.cxx

9.10 Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.
```
#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 \(w\) to keep track on.

9.10.1 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 –enable-cairo option
9.10.2 Member Function Documentation

9.10.2.1 cc()

```cpp
void Fl_Cairo_State::cc (cairo_t * c,
    bool own = true) [inline]
```

Sets the current cairo context.

- `own == true` (the default) indicates that the cairo context `c` will be deleted by FLTK internally when another cc is set later.
- `own == false` indicates cc deletion is handled externally by the user program.

The documentation for this class was generated from the following files:

- Fl_Cairo.H
- Fl_Cairo.cxx

9.11 Fl_Cairo_Window Class Reference

This defines a pre-configured cairo fltk window.

```cpp
#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)`
  
  This defines the cairo draw callback prototype that you must further.

Public Member Functions

- `Fl_Cairo_Window (int w, int h)`
- `void set_draw_cb (cairo_draw_cb cb)`

You must provide a draw callback which will implement your cairo rendering.

Public Member Functions inherited from Fl_Double_Window

- `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 ()`
  
  Forces the window to be redrawn.
- `void hide ()`
  
  Removes the window from the screen.
• void resize (int, int, int, int)
  Changes the size and position of the window.

• void show ()
  Puts the window on the screen.

• void show (int a, char **b)

• ~Fl_Double_Window ()
  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

• virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• unsigned int border () const
  See void Fl_Window::border(int)

• 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.

• 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 ()
  Returns the window height including any window title bar and any frame added by the window manager.

• int decorated_w ()
  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.

• Fl_Window (int w, int h, const char *title=0)
  Creates a window from the given size and title.

• Fl_Window (int x, int y, int w, int h, const char *title=0)
  Creates a window from the given position, size 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.

• virtual int handle (int)
  
  Handles the specified event.

• 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)

  Sets the current icon window target dependent data.

• 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.

• 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.

• 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.

• unsigned int override () const

  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

• 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, FL_OVERRIDE.

- void set_tooltip_window()
  Marks the window as a tooltip window.

- 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 (int argc, char **argv)
  Puts the window on the screen 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 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.

- virtual Fl_Group *as_group ()
  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
  
  See void Fl_Group::resizable(Fl_Widget *box).

• void **resizable** (Fl_Widget &o)
  
  See void Fl_Group::resizable(Fl_Widget *box).

• void **resizable** (Fl_Widget *o)
  
  The resizable widget defines the resizing box for the group.

• 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.

• 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, void *p)
  Sets the current callback function 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.

- **const Fl_Image** ∗ deimage () const

- **void** deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- **void** deimage (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label.

- **void** do_callback ()
  
  Calls the widget callback.

- **void** do_callback (Fl_Widget ∗ o, long arg)
  
  Calls the widget callback.

- **void** do_callback (Fl_Widget ∗ o, void ∗ arg=0)
  
  Calls the widget callback.

- **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.

- **const Fl_Image** ∗ image () const

- **void** image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- **void** image (Fl_Image ∗ img)
  
  Sets the image to use as part of the widget label.

- **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.
• 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.
• 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.
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 (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 ()
  Overloaded to provide cairo callback support.

Protected Member Functions inherited from Fl_Double_Window

- void flush (int eraseoverlay)
  Forces the window to be redrawn.

Protected Member Functions inherited from Fl_Window

- 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.
Protected Member Functions inherited from Fl_Group

- `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 ()`  
  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_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_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 const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Attributes inherited from Fl_Double_Window

- char force_doublebuffering_
  Force double buffering, even if the OS already buffers windows (overlays need that on MacOS and Windows2000)
9.12 Fl_Cairo_Window Class Reference

Protected Attributes inherited from Fl_Window

• shape_data_type * shape_data_
  
  non-null means the window has a non-rectangular shape

Static Protected Attributes inherited from Fl_Window

• static Fl_Window * current_
  
  Stores the last window that was made current.

9.11.1 Detailed Description

This defines a pre-configured cairo fltk window. 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.

Note

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 it only when your window is the current window.

See also

Fl_Window::current()

9.11.2 Member Function Documentation

9.11.2.1 draw()

void Fl_Cairo_Window::draw (  
  void ) [inline], [protected], [virtual]
  
Overloaded to provide cairo callback support. Reimplemented from Fl_Window.

9.11.2.2 set_draw_cb()

void Fl_Cairo_Window::set_draw_cb (  
  cairo_draw_cb cb ) [inline]
  
You must provide a draw callback which will implement your cairo rendering. This method will permit you to set your cairo callback to cb. The documentation for this class was generated from the following file:

• Fl_Cairo_Window.H

9.12 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)
  Add the data value val with optional label str and color col to the chart.
- uchar autosize () const
  Get whether the chart will automatically adjust the bounds of the chart.
- void autosize (uchar n)
  Set 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)
  Set the maximum number of data values for a chart.
- void replace (int ind, double val, const char ∗str=0, unsigned col=0)
  Replace a data value val at the given position ind.
- int size () const
  Returns the number of data values in the chart.
- void size (int W, int H)
- Fl_Color textcolor () const
  Gets the chart's text color.
- void textcolor (Fl_Color n)
  Gets 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)
  Gets 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 Fl_Group ∗ as_group ()
  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.
• 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, 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)
  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 color 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.

- **const Fl_Image * deimage () const**
  void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image * image () const**
  void image (Fl_Image &img)
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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.
• 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.
• 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 (void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  Returns whether a widget is visible.

- **unsigned int visible_focus ()**
  
  Checks whether this widget has a visible focus.

- **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** ()
  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** ()
  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_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` ∗ cb, void ∗ d)
  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

```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, 
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
```

flags possible values enumeration.

9.12.1 Detailed Description

Fl_Chart displays simple charts.
It is provided for Forms compatibility.

```
Figure 9.2 Fl_Chart
```

Todo Refactor Fl_Chart::type() information.

The type of an Fl_Chart object can be set using type(uchar t) to:

- **FL_BAR_CHART**: Each sample value is drawn as a vertical bar.
- **FL_FILLED_CHART**: The chart is filled from the bottom of the graph to the sample values.
- **FL_HORBAR_CHART**: Each sample value is drawn as a horizontal bar.
- **FL_LINE_CHART**: The chart is drawn as a polyline with vertices at each sample value.
- **FL_PIE_CHART**: A pie chart is drawn with each sample value being drawn as a proportionate slice in the circle.
- **FL_SPECIALPIE_CHART**: Like FL_PIE_CHART, but the first slice is separated from the pie.
- **FL_SPIKE_CHART**: Each sample value is drawn as a vertical line.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 Fl_Chart()

Fl_Chart::Fl_Chart ( 
    int X, 
    int Y, 
    int W, 
```
Create a new Fl_Chart widget using the given position, size and label string. The default boxstyle is FL_NO_BOX.

Parameters

- `X, Y, W, H`: position and size of the widget
- `L`: widget label, default is no label

### 9.12.3 Member Function Documentation

#### 9.12.3.1 add()

```cpp
void Fl_Chart::add ( double val, const char *str = 0, unsigned col = 0 )
```

Add the data value `val` with optional label `str` and color `col` to the chart.

Parameters

- `val`: data value
- `str`: optional data label
- `col`: optional data color

#### 9.12.3.2 autosize() [1/2]

```cpp
uchar Fl_Chart::autosize ( ) const [inline]
```

Get whether the chart will automatically adjust the bounds of the chart.

Returns

- non-zero if auto-sizing is enabled and zero if disabled.

#### 9.12.3.3 autosize() [2/2]

```cpp
void Fl_Chart::autosize ( uchar n ) [inline]
```

Set whether the chart will automatically adjust the bounds of the chart.

Parameters

- `n`: non-zero to enable automatic resizing, zero to disable.

#### 9.12.3.4 bounds() [1/2]

```cpp
void Fl_Chart::bounds ( double *a, double *b ) const [inline]
```

Gets the lower and upper bounds of the chart values.

Parameters

- `a, b`: are set to lower, upper
9.12.3.5 bounds() [2/2]

```cpp
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

9.12.3.6 draw()

```cpp
void Fl_Chart::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 = &scroll; // scroll is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

9.12.3.7 insert()

```cpp
void Fl_Chart::insert (  
  int ind,  
  double val,  
  const char * str = 0,  
  unsigned col = 0  
)
```

Inserts a data value `val` at the given position `ind`.

Position 1 is the first data value.

Parameters

- `in ind` insertion position
- `in val` data value
- `in str` optional data label
- `in col` optional data color

9.12.3.8 maxsize()

```cpp
void Fl_Chart::maxsize (  
  int m  
)
```

Set 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 m` maximum number of data values allowed.

9.12.3.9 replace()

```cpp
void Fl_Chart::replace (  
  int ind,  
```

Generated by Doxygen
Replace a data value `val` at the given position `ind`.
Position 1 is the first data value.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ind</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>data value</td>
</tr>
<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>

The documentation for this class was generated from the following files:

- Fl_Chart.H
- Fl_Chart.cxx

### 9.13 FL_CHART_ENTRY Struct Reference

For internal use only.

```cpp
#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.

#### 9.13.1 Detailed Description

For internal use only.

The documentation for this struct was generated from the following file:

- Fl_Chart.H

### 9.14 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.

```cpp
#include <Fl_Check_Browser.H>
```

Inheritance diagram for Fl_Check_Browser:

```
Fl_Widget

Fl_Group

Fl_Browser_

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.
- 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 position () const
Gets the vertical scroll position of the list as a pixel position \texttt{pos}.

- \texttt{void position (int pos)}
  
  Sets the vertical scroll position of the list to pixel position \texttt{pos}.

- \texttt{void resize (int X, int Y, int W, int H)}
  
  Repositions and/or resizes the browser.

- \texttt{void scrollbar_left ()}
  
  Moves the vertical scrollbar to the lefthand side of the list.

- \texttt{void scrollbar_right ()}
  
  Moves the vertical scrollbar to the righthand side of the list.

- \texttt{int scrollbar_size () const}
  
  Gets the current size of the scrollbars' troughs, in pixels.

- \texttt{void scrollbar_size (int newSize)}
  
  Sets the pixel size of the scrollbars' troughs to \texttt{newSize}, in pixels.

- \texttt{int scrollbar_width () const}
  
  This method has been deprecated, existing for backwards compatibility only.

- \texttt{void scrollbar_width (int width)}
  
  This method has been deprecated, existing for backwards compatibility only.

- \texttt{int select (void ∗item, int val=1, int docallbacks=0)}
  
  Sets the selection state of \texttt{item} to \texttt{val}, and returns 1 if the state changed or 0 if it did not.

- \texttt{int select_only (void ∗item, int docallbacks=0)}
  
  Selects \texttt{item} and returns 1 if the state changed or 0 if it did not.

- \texttt{void sort (int flags=0)}
  
  Sort the items in the browser based on \texttt{flags}.

- \texttt{FL_Color textcolor () const}
  
  Gets the default text color for the lines in the browser.

- \texttt{void textcolor (FL_Color col)}
  
  Sets the default text color for the lines in the browser to \texttt{color col}.

- \texttt{FL_Font textfont () const}
  
  Gets the default text font for the lines in the browser.

- \texttt{void textfont (FL_Font font)}
  
  Sets the default text font for the lines in the browser to \texttt{font}.

- \texttt{FL_Fontsize textsize () const}
  
  Gets the default text size (in pixels) for the lines in the browser.

- \texttt{void textsize (FL_Fontsize newSize)}
  
  Sets the default text size (in pixels) for the lines in the browser to \texttt{size}.

\section*{Public Member Functions inherited from \texttt{FL_Group}}

- \texttt{FL_Widget ∗& _ddfdesign_kludge ()}
  
  This is for forms compatibility only.

- \texttt{void add (FL_Widget ∗)}
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- \texttt{void add (FL_Widget ∗)}
  
  \texttt{See void FL_Group::add(FL_Widget ∗w)}

- \texttt{void add_resizable (FL_Widget ∗)}
  
  Adds a widget to the group and makes it the resizable widget.

- \texttt{FL_Widget ∗const ∗ array () const}
  
  Returns a pointer to the array of children.

- \texttt{virtual FL_Group ∗ as_group ()}
  
  Returns an \texttt{FL_Group} pointer if this widget is an \texttt{FL_Group}.

- \texttt{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 *)**

- **void forms_end ()**

  This is for forms compatibility only.

- **int handle (int)**

  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**

  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget &o)**

  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget *o)**

  The resizable widget defines the resizing box for the group.

- **void resize (int, int, int, int)**

  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 Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_WINDOW.
- 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, 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)
  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.

- const Fl_Image *deimage () const
  void deimage (Fl_Image &img)
    Sets the image to use as part of the widget label.

- void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.

- void do_callback ()
  Calls the widget callback.

- void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.

- void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.

- 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.

- const Fl_Image *image () const
  void image (Fl_Image &img)
    Sets the image to use as part of the widget label.

- void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
- `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.
- `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.

- 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 (void ∗v)`
  Sets the user data for this widget.

- unsigned int `visible () const`
  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 () 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)**
  
  Handles the event within the normal widget bounding box.

### 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 ()**
  
  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_last () const**
  
  This method must be provided by the subclass to return the last item in the list.

- **virtual const char *item_text (void *item) const**
  
  This optional method returns a string (label) that may be used for sorting.

- **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

- `void draw ()`
  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 ()`
  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_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_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 *cb, void *d) 
  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, 
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 
} 
  flags possible values enumeration.

9.14.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.
9.14.2 Member Function Documentation

9.14.2.1 add() [1/2]

```cpp
int Fl_Check_Browser::add (char * s)
```

Add a new unchecked line to the end of the browser.

See also

add(char *s, int b)

9.14.2.2 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.

9.14.2.3 handle()

```cpp
int Fl_Check_Browser::handle (int event)
```

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_.

9.14.2.4 nchecked()

```cpp
int Fl_Check_Browser::nchecked () const
```

Returns how many items are currently checked.

9.14.2.5 nitems()

```cpp
int Fl_Check_Browser::nitems () const
```

Returns how many lines are in the browser. The last line number is equal to this.

9.14.2.6 remove()

```cpp
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.
9.14.2.7 set_checked()

```cpp
void Fl_Check_Browser::set_checked (int item) [inline]
```
Equivalent to Fl_Check_Browser::checked(item, 1).

The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

9.15 Fl_Check_Button Class Reference

A button with a "checkmark" to show its status.

Inheritance diagram for Fl_Check_Button:

```
Fl_Widget
   /
Fl_Button
   |
Fl_Light_Button
   |
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.
- `virtual int handle (int)`
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- `int clear ()`
  Same as value(0).
- `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 `Fl_Group ∗as_group()`
  
  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`.

- 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, 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)`
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.

- `const Fl_Image *deimage () const`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  Calls the widget callback.

- `void do_callback (Fl_Widget *o, long arg)`
Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image ∗image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

- 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 (void ∗v)**
  Sets the user data for this widget.

- unsigned int **visible () const**
  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 () 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 *cb, void *d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
  }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

• virtual void draw ()
  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
  Draws the widget box according its box style.

- void draw_box() const
  Draws a box of type t, of color c at the widget's position and size.

- void draw_box(Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void set_flag(unsigned int c)
  Sets a flag in the flags mask.

- void draw_focus()
  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_label()
  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.

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

9.15.1 Detailed Description

A button with a "checkmark" to show its status.

![Fl_Check_Button](image)

Figure 9.3 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.
9.15.2 Constructor & Destructor Documentation

9.15.2.1 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

| 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_Check_Button.H
- Fl_Check_Button.cxx

9.16 Fl_Choice Class Reference

A button that is used to pop up a menu.
#include <Fl_Choice.H>

Inheritance diagram for Fl_Choice:

```
Fl_Widget
  Fl_Menu_
    Fl_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)
  
  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.

- **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)**
  See Fl_Boxtype Fl_Menu_::down_box() const
- **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.
- **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 * 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.
• 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 Fl_Group * as_group ()**
  
  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.

- **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, void *p)**
  
  Sets the current callback function 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.*

• **const FL_Image *deimage () const**

• **void deimage (FL_Image &img)**
  
  *Sets the image to use as part of the widget label.*

• **void deimage (FL_Image *img)**
  
  *Sets the image to use as part of the widget label.*

• **void do_callback ()**
  
  *Calls the widget callback.*

• **void do_callback (FL_Widget *o, long arg)**
  
  *Calls the widget callback.*

• **void do_callback (FL_Widget *o, void *arg=0)**
  
  *Calls the widget callback.*

• **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.*

• **const FL_Image *image () const**

• **void image (FL_Image &img)**
  
  *Sets the image to use as part of the widget label.*

• **void image (FL_Image *img)**
  
  *Sets the image to use as part of the widget label.*

• **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.

- 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.

- 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 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 (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  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 () 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 ()
  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 ()
  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_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 *cb, void *d)
  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

\[
\text{enum} \{\ 
\text{INACTIVE} = 1<<0, \text{INVISIBLE} = 1<<1, \text{OUTPUT} = 1<<2, \text{NOBORDER} = 1<<3, \\
\text{FORCE\_POSITION} = 1<<4, \text{NON\_MODAL} = 1<<5, \text{SHORTCUT\_LABEL} = 1<<6, \text{CHANGED} = 1<<7, \\
\text{OVERRI}DE = 1<<8, \text{VISIBLE\_FOCUS} = 1<<9, \text{COPYED\_LABEL} = 1<<10, \text{CLIP\_CHILDREN} = 1<<11, \\
\text{MENU\_WINDOW} = 1<<12, \text{TOOLTIP\_WINDOW} = 1<<13, \text{MODAL} = 1<<14, \text{NO\_OVERLAY} = 1<<15, \\
\text{GROUP\_RELATIVE} = 1<<16, \text{COPYED\_TOOLTIP} = 1<<17, \text{FULLSCREEN} = 1<<18, \text{MAC\_USE\_ACCENTS\_MENU} = 1<<19, \\
\text{USER\_FLAG3} = 1<<29, \text{USER\_FLAG2} = 1<<30, \text{USER\_FLAG1} = 1<<31 \} \\
\]

flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar alloc
- uchar down_box_
- Fl_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

9.16.1 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.
Todo  Refactor the doxygen comments for Fl_Choice changed() documentation.

- int Fl_Widget::changed() const  This value is true 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.

- Fl_Boxtype Fl_Choice::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_Choice::down_box(Fl_Boxtype b)  Sets the current down box type to b.

### 9.16.2 Constructor & Destructor Documentation

#### 9.16.2.1 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

| in | X, Y, W, H | position and size of the widget |
| in | L | widget label, default is no label |

### 9.16.3 Member Function Documentation

#### 9.16.3.1 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:

```c
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
9.16.3.2 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.

**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`.

9.16.3.3 value() [1/3]

```cpp
int Fl_Choice::value ( ) const [inline]
```

Gets the index of the last item chosen by the user.
The index is zero initially.

9.16.3.4 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.

9.16.3.5 value() [3/3]

```cpp
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**

- `in v 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

### 9.17 Fl_Clock Class Reference

This widget provides a round analog clock display.

```cpp
#include <Fl_Clock.H>
```

Inheritance diagram for Fl_Clock:

```
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 boxtype, position, size, and label string.
- **int handle (int)**
  - 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).
- **int minute () const**
  - Returns the displayed minute (0 to 59).
- **int second () const**
  - Returns the displayed second (0 to 60, 60=leap second).
- **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 Fl_Group * as_group ()
  - 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.
- 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, 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)
  - 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 ()

Generated by Doxygen
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.

- `const FL_Image ∗deimage () const`

- `void deimage (FL_Image &img)`
  Sets the image to use as part of the widget label.

- `void deimage (FL_Image ∗img)`
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  Calls the widget callback.

- `void do_callback (FL_Widget ∗o, long arg)`
  Calls the widget callback.

- `void do_callback (FL_Widget ∗o, void ∗arg=0)`
  Calls the widget callback.

- `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.

- `const FL_Image ∗image () const`

- `void image (FL_Image &img)`
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

- 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 (void * v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 ∗cb, void ∗d)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Clock_Output

• void draw ()
  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 ()
  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_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.*

### 9.17.1 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()**, see below. br

![FL_SQUARE_CLOCK type](image1)

**Figure 9.5 FL_SQUARE_CLOCK type**

br

![FL_ROUND_CLOCK type](image2)

**Figure 9.6 FL_ROUND_CLOCK type**

Generated by Doxygen
9.17.2 Constructor & Destructor Documentation

9.17.2.1 Fl_Clock() [1/2]

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 boxtype is FL_NO_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>

9.17.2.2 Fl_Clock() [2/2]

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 boxtype, position, size, and label string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>boxtype</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>

9.17.3 Member Function Documentation

9.17.3.1 handle()

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.

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_Clock.H
- Fl_Clock.cxx

9.18 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).

- 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.
- 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

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 Fl_Group * as_group ()
  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.

• 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, 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)
  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.

• const Fl_Image *deimage () const

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.

• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.

• void do_callback ()
  Calls the widget callback.

• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.

• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.

• 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.

• const Fl_Image *image () const

• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.

• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.

• int inside (const Fl_Widget *wgt) const
  Checks if this widget is a child of \( \text{wgt} \).

• 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.

• 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.

• 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 **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** (void **∗** v)
  Sets the user data for this widget.

• unsigned int **visible** () const
  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** () 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 ()**
  
  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 ()**

  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_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 ∗cb, void ∗d)**

  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.
9.18 Fl_Clock_Output Class Reference

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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.18.1 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.

9.18.2 Constructor & Destructor Documentation

9.18.2.1 Fl_Clock_Output()

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. The default boxtype is FL_NO_BOX.
### Member Function Documentation

#### 9.18.3.1 draw() [1/2]

```cpp
draw ( ) [protected], [virtual]
```

Draw clock with current position and size. Implements Fl_Widget.

#### 9.18.3.2 draw() [2/2]

```cpp
draw ( int X, int Y, int W, int H ) [protected]
```

Draw clock with the given position and size.

#### 9.18.3.3 hour()

```cpp
hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).

See also

- value(), minute(), second()

#### 9.18.3.4 minute()

```cpp
minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).

See also

- value(), hour(), second()

#### 9.18.3.5 second()

```cpp
second ( ) const [inline]
```

Returns the displayed second (0 to 60, 60=leap second).

See also

- value(), hour(), minute()
9.18.3.6 value() [1/3]
ulong Fl_Clock_Output::value ( ) const [inline]
Returns the displayed time.
Returns the time in seconds since the UNIX epoch (January 1, 1970).
See also
    value(ulong)

9.18.3.7 value() [2/3]
void Fl_Clock_Output::value (  
    int H,  
    int m,  
    int s )
Set the displayed time.
Set the time in hours, minutes, and seconds.
Parameters
    
      in  H,m,s  displayed time

See also
    hour(), minute(), second()

9.18.3.8 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

9.19 Fl_Color_Chooser Class Reference

The Fl_Color_Chooser widget provides a standard RGB color chooser.
#include <Fl_Color_Chooser.H>
Inheritance diagram for 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 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 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.

- **virtual Fl_Group ∗as_group ()**
  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.

- int handle(int)
  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
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable(Fl_Widget &)
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable(Fl_Widget *o)
  The resizable widget defines the resizing box for the group.

- void resize(int, int, int, int)
  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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• 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, 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)
  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.

- \texttt{int \ contains \ (const \ Fl\_Widget \ *w) \ const}
  - Checks if \texttt{w} is a child of this widget.
- \texttt{void \ copy\_label \ (const \ char \ *new\_label)}
  - Sets the current label.
- \texttt{void \ copy\_tooltip \ (const \ char \ *text)}
  - Sets the current tooltip text.
- \texttt{uchar \ damage \ () \ const}
  - Returns non-zero if \texttt{draw()} needs to be called.
- \texttt{void \ damage \ (uchar \ c)}
  - Sets the damage bits for the widget.
- \texttt{void \ damage \ (uchar \ c, \ int \ x, \ int \ y, \ int \ w, \ int \ h)}
  - Sets the damage bits for an area inside the widget.
- \texttt{int \ damage\_resize \ (int, \ int, \ int, \ int)}
  - Internal use only.
- \texttt{void \ deactivate \ ()}
  - Deactivates the widget.
- \texttt{Fl\_Image \ * \ deimage \ ()}
  - Gets the image that is used as part of the widget label.
- \texttt{const \ Fl\_Image \ * \ deimage \ () \ const}
- \texttt{void \ deimage \ (Fl\_Image \ &img)}
  - Sets the image to use as part of the widget label.
- \texttt{void \ deimage \ (Fl\_Image \ *img)}
  - Sets the image to use as part of the widget label.
- \texttt{void \ do\_callback \ ()}
  - Calls the widget callback.
- \texttt{void \ do\_callback \ (Fl\_Widget \ *o, \ long \ arg)}
  - Calls the widget callback.
- \texttt{void \ do\_callback \ (Fl\_Widget \ *o, \ void \ *arg=0)}
  - Calls the widget callback.
- \texttt{void \ draw\_label \ (int, \ int, \ int, \ int, \ Fl\_Align) \ const}
  - Draws the label in an arbitrary bounding box with an arbitrary alignment.
- \texttt{int \ h \ () \ const}
  - Gets the widget height.
- \texttt{virtual \ void \ hide \ ()}
  - Makes a widget invisible.
- \texttt{Fl\_Image \ * \ image \ ()}
  - Gets the image that is used as part of the widget label.
- \texttt{const \ Fl\_Image \ * \ image \ () \ const}
- \texttt{void \ image \ (Fl\_Image \ &img)}
  - Sets the image to use as part of the widget label.
- \texttt{void \ image \ (Fl\_Image \ *img)}
  - Sets the image to use as part of the widget label.
- \texttt{int \ inside \ (const \ Fl\_Widget \ *wgt) \ const}
  - Checks if this widget is a child of \texttt{wgt}.
- \texttt{int \ is\_label\_copied \ () \ const}
  - Returns whether the current label was assigned with \texttt{copy\_label()}.
- \texttt{const \ char \ * \ label \ () \ const}
  - Gets the current label text.
- \texttt{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.

• 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.

• 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
  Sets 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 (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 *cb, void *d)
  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.

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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

- void draw ()
  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` ()
  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_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.

9.19.1 Detailed Description

The Fl_Color_Chooser widget provides a standard RGB color chooser.

![Figure 9.9 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).

`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.

### 9.19.2 Constructor & Destructor Documentation

#### 9.19.2.1 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**

- **in X, Y, W, H** position and size of the widget
- **in L** widget label, default is no label

### 9.19.3 Member Function Documentation

#### 9.19.3.1 b()

```cpp
double Fl_Color_Chooser::b ( ) const [inline]
```

Returns the current blue value.

\[ 0 \leq b \leq 1. \]

#### 9.19.3.2 g()

```cpp
double Fl_Color_Chooser::g ( ) const [inline]
```

Returns the current green value.

\[ 0 \leq g \leq 1. \]

#### 9.19.3.3 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.
9.19 Fl_Color_Chooser Class Reference

Returns

1 if a new hsv value was set, 0 if the hsv value was the previous one.

9.19.3.4 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

<table>
<thead>
<tr>
<th>in</th>
<th>H,S,V</th>
<th>color components</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>R,G,B</td>
<td>color components</td>
</tr>
</tbody>
</table>

9.19.3.5 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.

9.19.3.6 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)

9.19.3.7 mode() [2/2]

```cpp
void Fl_Color_Chooser::mode ( 
    int newMode)
```

Set which *Fl_Color_Chooser* variant is currently active.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>newMode</th>
<th>color modes are rgb(0), byte(1), hex(2), or hsv(3)</th>
</tr>
</thead>
</table>

9.19.3.8 r()

```cpp
double Fl_Color_Chooser::r ( ) const [inline]
```

Returns the current red value.

0 <= r <= 1.

9.19.3.9 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

| in   | R,G,B | color components |

Returns

1 if a new rgb value was set, 0 if the rgb value was the previous one.

9.19.3.10 rgb2hsv()

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

| in   | R,G,B | color components |

| out  | H,S,V | color components |

9.19.3.11 saturation()

double Fl_Color_Chooser::saturation() const [inline]

Returns the saturation.
0 \leq saturation \leq 1.

9.19.3.12 value()

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

9.20 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

#include <Fl_Copy_Surface.H>

Inheritance diagram for Fl_Copy_Surface:
9.20 Fl_Copy_Surface Class Reference

Public Member Functions

- const char ∗ class_name ()
  Returns the name of the class of this object.
- void draw (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)
  Copies a widget in the clipboard.
- void draw_decorated_window (Fl_Window ∗win, int delta_x=0, int delta_y=0)
  Copies a window and its borders and title bar to the clipboard.
- Fl_Copy_Surface (int w, int h)
  Constructor.
- int h ()
  Returns the pixel height of the copy surface.
- void set_current ()
  Make this surface the current drawing surface.
- int w ()
  Returns the pixel width of the copy surface.
- ∼Fl_Copy_Surface ()
  Destructor.

Public Member Functions inherited from Fl_Surface_Device

- const char ∗ class_name ()
  Returns the name of the class of this object.
- Fl_Graphics_Driver ∗ driver ()
  Returns the graphics driver of this drawing surface.
- void driver (Fl_Graphics_Driver ∗graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual ∼Fl_Surface_Device ()
  The destructor.

Public Member Functions inherited from Fl_Device

- virtual ∼Fl_Device ()
  Virtual destructor.

Static Public Attributes

- static const char ∗ class_id = “Fl_Copy_Surface”

Static Public Attributes inherited from Fl_Surface_Device

- static const char ∗ class_id = “Fl_Surface_Device”
Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * surface()
  The current drawing surface.

Protected Member Functions inherited from Fl_Surface_Device

- Fl_Surface_Device (Fl_Graphics_Driver * graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.

9.20.1 Detailed Description

Supports copying of graphical data to the clipboard.

After creation of an Fl_Copy_Surface object, call set_current() on it, 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
copy_surf->set_current(); // direct graphics requests to the clipboard
fl_color(FL_WHITE); fl_rectf(0, 0, g->w(), g->h()); // draw a white background
delete copy_surf; // after this, the clipboard is loaded
Fl_Display_Device::display_device()->set_current(); // direct graphics requests back to the display
```

Platform details:

- MSWindows: Transparent RGB images copy without transparency. The graphical data are copied to the clipboard as an 'enhanced metafile'.

- 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: the graphical data are copied to the clipboard as an image in BMP format.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 Fl_Copy_Surface()

Fl_Copy_Surface::Fl_Copy_Surface (  
    int w,  
    int h )

Constructor.

Parameters

<table>
<thead>
<tr>
<th>w</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

are the width and height of the clipboard surface in pixels where drawing will occur.
9.20.3 Member Function Documentation

9.20.3.1 class_name()

const char * Fl_Copy_Surface::class_name ( ) [inline], [virtual]

Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:

```cpp
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```

Reimplemented from Fl_Device.

9.20.3.2 draw()

```cpp
void Fl_Copy_Surface::draw ( Fl_Widget * widget, int delta_x = 0, int delta_y = 0 )
```

Copies a widget in the clipboard.

Parameters

| widget | any FLTK widget (e.g., standard, custom, window, GL view) to copy |
| delta-- | and |
| _x | |
| delta-- | give the position in the clipboard of the top-left corner of the widget |
| _y | |

9.20.3.3 draw_decorated_window()

```cpp
void Fl_Copy_Surface::draw_decorated_window ( Fl_Window * win, int delta_x = 0, int delta_y = 0 )
```

Copies a window and its borders and title bar to the clipboard.

Parameters

| win | an FLTK window to copy |
| delta-- | and |
| _x | |
| delta-- | give the position in the clipboard of the top-left corner of the window's title bar |
| _y | |

9.20.3.4 set_current()

```cpp
void Fl_Copy_Surface::set_current ( void ) [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Reimplemented from Fl_Surface_Device.
The documentation for this class was generated from the following files:

- Fl_Copy_Surface.H
- Fl_Copy_Surface.cxx
9.21 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.

```c
#include <Fl_Counter.H>
```

Inheritance diagram for Fl_Counter:

```
Fl_Widget
   +------------------ Fl_Valuator
   |                    +------------------ Fl_Counter
   +------------------ Fl_Simple_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)
  
  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 Fl_Group * as_group ()
  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.

• 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, void *p)
  Sets 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, 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)
9.21 Fl.Counter Class Reference

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.

- const Fl_Image * deimage () const
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void do_callback ()
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, long arg)
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, void *arg=0)
  
  Calls the widget callback.

- 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.

- const Fl_Image * image () const
  
  Sets the image to use as part of the widget label.

- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- 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.

• `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.

• `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)`
9.21 Fl.Counter Class Reference

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 (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 use_accents_menu ()
  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 ()
  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()`
  
  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_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 *cb, void *d)
  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,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
flags possible values enumeration.

9.21.1 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.

Todo  Refactor the doxygen comments for Fl_Counter type() documentation.

The type of an Fl_Counter object can be set using type(uchar t) to:

- FL_NORMAL_COUNTER: Displays a counter with 4 arrow buttons.
- FL_SIMPLE_COUNTER: Displays a counter with only 2 arrow buttons.

9.21.2 Constructor & Destructor Documentation

9.21.2.1 Fl_Counter()

Fl_Counter::Fl_Counter {
    int X,
    int Y,
int \texttt{W},
int \texttt{H},
\texttt{const char \ast L = 0 })

Creates a new Fl.Counter widget using the given position, size, and label string.
The default type is FL_NORMAL_COUNTER.

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>

9.21.3 Member Function Documentation

9.21.3.1 draw()

void 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 \texttt{redraw()} instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method \textit{from within your own draw() method}, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

\begin{verbatim}
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
\end{verbatim}

Implements Fl_Widget.

9.21.3.2 handle()

int Fl.Counter::handle (int \texttt{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.

Parameters

| in  | \texttt{event} | the kind of event received |

Return values

\begin{verbatim}
| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |
\end{verbatim}

See also

\texttt{Fl_Event}

Reimplemented from Fl_Widget.

9.21.3.3 lstep()

void Fl.Counter::lstep (double \texttt{s}) [inline]

Sets the increment for the large step buttons.
The default value is 1.0.
Parameters

| in a | large step increment. |

### 9.21.3.4 `step()` [1/2]

```cpp
void Fl_Counter::step (double a) [inline]
```

Sets the increment for the normal step buttons.

Parameters

| in a | normal step increment. |

### 9.21.3.5 `step()` [2/2]

```cpp
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

### 9.22 Fl_Device Class Reference

All graphical output devices and all graphics systems.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Device:

```
Fl_Device
  
  Fl_Graphics_Driver
    Fl_GDI_Graphics_Driver
    Fl_PostScript_Graphics_Driver
    Fl_Quartz_Graphics_Driver
    Fl_Xlib_Graphics_Driver
  
  Fl_Surface_Device
    Fl_Copy_Surface
  
  Fl_Display_Device
    Fl_Image_Surface
  
  Fl_Paged_Device
```

Public Member Functions

- virtual `const char * class_name ()`
  
  Returns the name of the class of this object.

- virtual `~Fl_Device ()`
  
  Virtual destructor.
Static Public Attributes

- static const char * class_id = "Fl_Device"
  
  A string that identifies each subclass of Fl_Device.

9.22.1 Detailed Description

All graphical output devices and all graphics systems. This class supports a rudimentary system of run-time type information.

9.22.2 Constructor & Destructor Documentation

9.22.2.1 ~Fl_Device()

virtual Fl_Device::~Fl_Device () [inline], [virtual]

Virtual destructor.

The destructor of Fl_Device must be virtual to make the destructors of derived classes being called correctly on destruction.

9.22.3 Member Function Documentation

9.22.3.1 class_name()

virtual const char * Fl_Device::class_name () [inline], [virtual]

Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:

```cpp
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```


9.22.4 Member Data Documentation

9.22.4.1 class_id

const char * Fl_Device::class_id = "Fl_Device" [static]

A string that identifies each subclass of Fl_Device.
Function class_name() applied to a device of this class returns this string.
The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

9.23 Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Device_Plugin:

```
Fl_Plugin

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
</tr>
</tbody>
</table>
```

Generated by Doxygen
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, int x, int y, int height)=0**
  Prints a widget.

- **Fl_RGB_Image * rectangle_capture (Fl_Widget *widget, int x, int y, int w, int h)**
  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.

9.23.1 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.

9.23.2 Member Function Documentation

9.23.2.1 print()

    virtual int Fl_Device_Plugin::print ( 
        Fl_Widget * w, 
        int x, 
        int y, 
        int height ) [pure virtual]

Prints a widget.

Parameters

<table>
<thead>
<tr>
<th>w</th>
<th>the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>offsets where to print relatively to coordinates origin</td>
</tr>
<tr>
<td>height</td>
<td>height of the current drawing area</td>
</tr>
</tbody>
</table>

9.23.2.2 rectangle_capture()

    Fl_RGB_Image * Fl_Device_Plugin::rectangle_capture ( 
        Fl_Widget * widget, 
        int x, 
        int y, 
        int w, 
        int h ) [inline]

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 file:

- Fl_Device.H

### 9.24 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
   |
   V
Fl_Valuator
   |
   V
Fl_Dial
   |
   V
Fl_Fill_Dial Fl_Line_Dial
```

**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)
  
  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)
  
  Rounds 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 Fl_Group ∗ as_group ()
  
  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.

- 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, 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)
  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 ()
9.24 Fl_Dial Class Reference

Deactivates the widget.

- **Fl_Image** ∗ deimage ()
  Gets the image that is used as part of the widget label.

- const Fl_Image ∗ deimage () const
- void deimage (Fl_Image img)
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image img)
  Sets the image to use as part of the widget label.

- void do_callback ()
  Calls the widget callback.

- void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.

- void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.

- 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.

- const Fl_Image ∗ image () const
- void image (Fl_Image img)
  Sets the image to use as part of the widget label.

- void image (Fl_Image img)
  Sets the image to use as part of the widget label.

- 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.*

• 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.*

• 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
  
  *Generated by Doxygen*
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 (void *v)
  
  Sets the user data for this widget.

- unsigned int visible () const
  
  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 () 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 ()
  
  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()`
  
  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_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 *cb, void *d)`
  
  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, USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31
}
flags possible values enumeration.

9.24.1 Detailed Description

The Fl_Dial widget provides a circular dial to control a single floating point value.

![Figure 9.11 Fl_Dial](image)

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.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 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.

9.24.3 Member Function Documentation

9.24.3.1 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.

9.24.3.2 draw() [1/2]

void Fl_Dial::draw (void ) [protected], [virtual]

Draws dial at current position and size. Implements Fl_Widget.
9.24.3.3 draw() [2/2]

```cpp
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 |

9.24.3.4 handle() [1/2]

```cpp
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. |

9.24.3.5 handle() [2/2]

```cpp
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

9.25 Fl_Display_Device Class Reference

A display to which the computer can draw.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl(Display_Device):

```
Fl_Device
  ↓
Fl_Surface_Device
  ↓
Fl_Display_Device
```

Public Member Functions

- const char * class_name ()
Returns the name of the class of this object.

- **Fl_Display_Device** (`Fl/Graphics_Driver + graphics_driver`)
  
  A constructor that sets the graphics driver used by the display.

### Public Member Functions inherited from Fl_Surface_Device

- **const char* class_name()**
  
  Returns the name of the class of this object.

- **Fl(Graphics_Driver)* driver()**
  
  Returns the graphics driver of this drawing surface.

- **void driver(Fl/Graphics_Driver + graphics_driver)**
  
  Sets the graphics driver of this drawing surface.

- **virtual void set_current(void)**
  
  Makes this surface the current drawing surface.

- **virtual ~Fl_Surface_Device()**
  
  The destructor.

### Public Member Functions inherited from Fl_Device

- **virtual ~Fl_Device()**
  
  Virtual destructor.

### Static Public Member Functions

- **static Fl_Display_Device* display_device()**
  
  Returns the platform display device.

### Static Public Member Functions inherited from Fl_Surface_Device

- **static Fl_Surface_Device* surface()**
  
  The current drawing surface.

### Static Public Attributes

- **static const char* class_id = "Fl_Display_Device"**

### Static Public Attributes inherited from Fl_Surface_Device

- **static const char* class_id = "Fl_Surface_Device"**

### Static Public Attributes inherited from Fl_Device

- **static const char* class_id = "Fl_Device"**
  
  A string that identifies each subclass of Fl_Device.

### Additional Inherited Members

### Protected Member Functions inherited from Fl_Surface_Device

- **Fl_Surface_Device(Fl/Graphics_Driver + graphics_driver)**
  
  Constructor that sets the graphics driver to use for the created surface.

### 9.25.1 Detailed Description

A display to which the computer can draw.

When the program begins running, an Fl_Display_Device instance has been created and made the current drawing surface. There is no need to create any other object of this class.
9.25.2 Member Function Documentation

9.25.2.1 class_name()

const char * Fl_Display_Device::class_name () [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
Reimplemented from Fl_Device.
The documentation for this class was generated from the following files:
- Fl_Device.H
- Fl_Device.cxx

9.26 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:

Public Member Functions

- 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 ()
  Forces the window to be redrawn.
- void hide ()
  Removes the window from the screen.
- void resize (int, int, int, int)
  Changes the size and position of the window.
- void show ()
  Puts the window on the screen.
- void show (int a, char **b)
- ~Fl_Double_Window ()
  The destructor also deletes all the children.
Public Member Functions inherited from Fl_Window

- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int border () const
  See void Fl_Window::border(int)
- 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 (Fl_RGB_Image *, int, int)
  Changes the cursor for this window.
- void cursor (Fl_Cursor, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
- void cursor (Fl_Cursor)
  Changes the cursor for this window.
- int decorated_h ()
  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w ()
  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.
- Fl_Window (int w, int h, const char *title=0)
  Creates a window from the given size and title.
- Fl_Window (int x, int y, int w, int h, const char *title=0)
  Creates a window from the given position, size 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.
- virtual int handle (int)
  Handles the specified event.
- 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 *)
  
  Sets the current icon window target dependent data.

• void **iconize** ()
  
  Iconifies the window.

• const char **iconlabel**(const char *) 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.

• const char **label**(const char *) const
  
  See void Fl_Window::label(const char *)

• void **label**(const char *)
  
  Sets the window title bar label.

• void **label**(const char *, const char *)
  
  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.

• unsigned int **menu_window**(const char *) const
  
  Returns true if this window is a menu window.

• unsigned int **modal**(const char *) const
  
  Returns true if this window is modal.

• unsigned int **non_modal**(const char *) const
  
  Returns true if this window is modal or non-modal.

• unsigned int **override**(const char *) const
  
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

• 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|FL_OVERRIDE.

• void **set_tooltip_window** ()
  
  Marks the window as a tooltip window.

• void **shape**(const Fl_Image &)
  
  Set the window’s shape with an Fl_Image.

• void **shape**(const Fl_Image *)
  
  Assigns a non-rectangular shape to the window.

• void **show**(int argc, char **argv)
  
  Show the Window.
Puts the window on the screen 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 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.
- virtual Fl_Group ∗ as_group ()
  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()
  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**

  See void Fl_Group::resizable(Fl_Widget ∗box)

- **void** resizable (Fl_Widget ∗o)

  See void Fl_Group::resizable(Fl_Widget ∗box)

  The resizable widget defines the resizing box for the group.

- **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.

- **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, 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)**
  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.

- `const Fl_Image * deimage () const`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image &img)`

- `void deimage (Fl_Image * img)`
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  Calls the widget callback.

- `void do_callback (Fl_Widget * o, long arg)`
  Calls the widget callback.

- `void do_callback (Fl_Widget * o, void * arg=0)`
  Calls the widget callback.

- `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.

- `const Fl_Image * image () const`
  Sets the image to use as part of the widget label.

- `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label.

- `void image (Fl_Image * img)`
  Sets the image to use as part of the widget label.

- `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.
• 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.
• 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 (void *v)
Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 flush (int eraseoverlay)
  Forces the window to be redrew.

Protected Member Functions inherited from Fl_Window

- virtual void draw ()
  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.

Protected Member Functions inherited from Fl_Group

- 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** ()
  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_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

- char **force_doublebuffering**_
  Force double buffering, even if the OS already buffers windows (overlays need that on MacOS and Windows2000)

Protected Attributes inherited from Fl_Window

- shape_data_type ∗ **shape_data**
  non-null means the window has a non-rectangular shape

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 ∗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 const char * **default_xclass** ()
  
  Returns the default xclass.

• static void **default_xclass** (const char *)
  
  Sets the default window xclass.

### 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 *cb, void *d)
  
  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, 
  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.

### 9.26.1 Detailed Description

The **Fl_Double_Window** provides a double-buffered window. If possible this will use the X double buffering extension (Xdbe). If not, it will draw the window data into an off-screen pixmap, and then copy it to the on-screen window.

It is highly recommended that you put the following code before the first show() of any window in your program:

```
Fl::visual(FL_DOUBLE|FL_INDEX)
```

This makes sure you can use Xdbe on servers where double buffering does not exist for every visual.
9.26.2 Constructor & Destructor Documentation

9.26.2.1 \texttt{\~Fl\_Double\_Window()}

\texttt{Fl\_Double\_Window::\~Fl\_Double\_Window ( )}

The destructor \textit{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.

9.26.3 Member Function Documentation

9.26.3.1 \texttt{flush() [1/2]}

\texttt{void Fl\_Double\_Window::flush ( ) [virtual]}
Forces the window to be redrawn.
Reimplemented from \texttt{Fl\_Window}.
Reimplemented in \texttt{Fl\_Overlay\_Window}.

9.26.3.2 \texttt{flush() [2/2]}

\texttt{void Fl\_Double\_Window::flush (int eraseoverlay) [protected]}
Forces the window to be redrawn.

\textbf{Parameters}

\begin{verbatim}
   in  eraseoverlay  non-zero to erase overlay, zero to ignore
\end{verbatim}

\texttt{Fl\_Overlay\_Window} relies on \texttt{flush(1)} copying the back buffer to the front everywhere, even if \texttt{damage() == 0}, thus erasing the overlay, and leaving the clip region set to the entire window.

9.26.3.3 \texttt{hide()}

\texttt{void Fl\_Double\_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 \texttt{Fl\_Window}.
Reimplemented in \texttt{Fl\_Overlay\_Window}.

9.26.3.4 \texttt{resize()}

\texttt{void Fl\_Double\_Window::resize (int X, int Y, int W, int H) [virtual]}
Changes the size and position of the window.
If \texttt{shown()} is true, these changes are communicated to the window server (which may refuse that size and cause a further resize). If \texttt{shown()} is false, the size and position are used when \texttt{show()} is called. See \texttt{Fl\_Group} for the effect of resizing on the child widgets.
You can also call the \texttt{Fl\_Widget} methods \texttt{size(x,y)} and \texttt{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 \texttt{Fl\_Window}.
Reimplemented in \texttt{Fl\_Overlay\_Window}.

9.26.3.5 \texttt{show()}

\texttt{void Fl\_Double\_Window::show ( ) [virtual]}

Generated by Doxygen
Puts the window on the screen.
Usually (on X) this has the side effect of opening the display.
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_Window.
Reimplemented in Fl_Overlay_Window.
The documentation for this class was generated from the following files:
• Fl_Double_Window.H
• Fl_Double_Window.cxx

9.27 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()

9.27.1 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::MyClass() :
    group(10,10,100,100),
    button_in_group(20,20,60,30),
    end(),
    button_outside_group(10,120,60,30)
{}
The documentation for this class was generated from the following file:
• Fl_Group.H

9.28 Fl_File_Browser Class Reference
The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.
#include <Fl_File_Browser.H>
Inheritance diagram for Fl_File_Browser:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Browser
  ↓
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

- 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 *=NULL)
  
  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` ()
  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)
  Scrolls 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 ()**
  
  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)
  
  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 position () const
  
  Gets the vertical scroll position of the list as a pixel position pos.

• void position (int pos)
  
  Sets the vertical scroll position of the list to pixel position pos.

• void resize (int X, int Y, int W, int H)
  
  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
  
  This method has been deprecated, existing for backwards compatibility only.

• void scrollbar_width (int width)
  
  This method has been deprecated, existing for backwards compatibility only.

• 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.

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.

• virtual **Fl_Group** + as_group ()
  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.

• int handle (int)
  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
  See void **Fl_Group**::resizable(**Fl_Widget** +box)

• void resizable (**Fl_Widget** &o)
  See void **Fl_Group**::resizable(**Fl_Widget** +box)

• void resizable (**Fl_Widget** +o)
  The resizable widget defines the resizing box for the group.

• void resize (int, int, int, int)
  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 Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• 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, 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)
  
  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.

- **const Fl_Image *deimage () const**

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const Fl_Image *image () const**

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **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.
• 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 ()
  Generated by Doxygen
Enables keyboard focus navigation with this widget.

- 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** (void ∗v)
  Sets the user data for this widget.

- unsigned int **visible** () const
  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** () 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 *cb, void *d)
  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 ,
  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.
- void insert (int line, FL_BLINE *item)
  Insert specified item above line.
- void * item_at (int line) const
  Return the item at specified line.
- void * item_first () const
  Returns the very first item in the list.
- void * item_last () const
  Returns the very last item in the list.
- void * item_next (void *item) const
  Generated by Doxygen
- Returns the next item after `item`.
- `void *item_prev (void *item) const`
  Returns the previous item before `item`.
- `void item_select (void *item, int val)`
  Change the selection state of `item` to the value `val`.
- `int item_selected (void *item) const`
  See if `item` is selected.
- `void item_swap (void *a, void *b)`
  Swap the items `a` and `b`.
- `const char *item_text (void *item) const`
  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 ()`
  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.

Generated by Doxygen
Protected Member Functions inherited from Fl_Group

- void draw ()
  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 ()
  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_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.

9.28.1 Detailed Description

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.
9.28 Fl_File_Browser Class Reference

9.28.2 Constructor & Destructor Documentation

9.28.2.1 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.

9.28.3 Member Function Documentation

9.28.3.1 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.

9.28.3.2 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.

9.28.3.3 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.

9.28.3.4 filter() [2/2]

```cpp
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.

9.28.3.5 iconsize() [1/2]

```cpp
uchar Fl_File_Browser::iconsize ( ) const [inline]
```
Sets or gets the size of the icons.
The default size is 20 pixels.

9.28.3.6 iconsize() [2/2]

```cpp
void Fl_File_Browser::iconsize ( 
    uchar s ) [inline]
```
Sets or gets the size of the icons.
The default size is 20 pixels.

9.28.3.7 load()

```cpp
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. The sort argument specifies a sort function to be used with `fl_filename_list()`. The documentation for this class was generated from the following files:

- Fl_File_Browser.H
- Fl_File_Browser.cxx

### 9.29 Fl_File_Chooser Class Reference

The `Fl_File_Chooser` widget displays a standard file selection dialog that supports various selection modes.

#### Public Types

- `enum { SINGLE = 0, MULTI = 1, CREATE = 2, DIRECTORY = 4 }`

#### Public Member Functions

- `Fl_Widget * add_extra (Fl_Widget *gr)`
  Adds extra widget at the bottom of `Fl_File_Chooser` window.
- `void callback (void (*cb)(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 ()`
  See `void filter(const char *pattern)`
- `void filter (const char *p)`
  Sets or gets 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 *d, const char *p, int t, const char *title)`
  The constructor creates the `Fl_File_Chooser` dialog shown.
- `void hide ()`
  Hides the `Fl_File_Chooser` window.
- `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 ()`
9.29 Fl_File_Chooser Class Reference

- 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 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) see Fl_Window::shown()

- 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 i=1)
    - Gets the current value of the selected file(s).
  - int visible ()
    - Returns 1 if the Fl_File_Chooser window is visible.
  - ~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 = "File Systems"
  [standard text may be customized at run-time]
- static const char * hidden_label = "Show hidden files"
  [standard text may be customized at run-time]
- static const char * manage_favorites_label = "Manage Favorites"
  [standard text may be customized at run-time]
- static const char * new_directory_label = "New Directory?"
  [standard text may be customized at run-time]
- static const char * new_directory_tooltip = "Create a new directory."
  [standard text may be customized at run-time]
- static const char * preview_label = "Preview"
  [standard text may be customized at run-time]
- static const char * save_label = "Save"
  [standard text may be customized at run-time]
- static const char * show_label = "Show:"
  [standard text may be customized at run-time]
- static Fl_File_Sort_F * sort = fl_numericsort
  the sort function that is used when loading the contents of a directory.

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(*)(const char *))
  Set the file chooser callback.
- void fl_file_chooser_ok_label (const char *)
  Set the "OK" button label.
9.29 Fl_File_Chooser Class Reference

9.29.1 Detailed Description

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

The Fl_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 MSWindows 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; (WIN32) &quot;File Systems&quot; (all others)</td>
</tr>
<tr>
<td>hidden_label</td>
<td>&quot;Show hidden files:&quot;</td>
</tr>
<tr>
<td>manageFavorites_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>showing_label</td>
<td>&quot;Show:&quot;</td>
</tr>
<tr>
<td>sort</td>
<td>fl_numericsort</td>
</tr>
</tbody>
</table>

Figure 9.12 Fl_File_Chooser
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. For more complex customization, consider copying the FLTK file chooser code and changing it accordingly.

### 9.29.2 Constructor & Destructor Documentation

#### 9.29.2.1 Fl_File_Chooser()

```cpp
Fl_File_Chooser::Fl_File_Chooser (
    const char ∗ pathname,
    const char ∗ pattern,
    int type,
    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` argument can be one of the following:

- SINGLE - allows the user to select a single, existing file.
- MULTI - allows the user to select one or more existing files.
- CREATE - allows the user to select a single, existing file or specify a new filename.
- DIRECTORY - allows the user to select a single, existing directory.

The `title` argument is used to set the title bar text for the `Fl_File_Chooser` window.

### 9.29.3 Member Function Documentation

#### 9.29.3.1 add_extra()

```cpp
Fl_Widget ∗ Fl_File_Chooser::add_extra ( 
    Fl_Widget ∗ gr )
```

Adds extra widget at the bottom of `Fl_File_Chooser` window.

Returns pointer for previous extra widget or NULL if not set previously. If argument is NULL only remove previous extra widget.

Note

`Fl_File_Chooser` does not delete extra widget in destructor! To prevent memory leakage, don't forget to delete unused extra widgets.

#### 9.29.3.2 filter()

```cpp
void Fl_File_Chooser::filter ( 
    const char ∗ pattern )
```

Sets or gets 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\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.
9.29.3.3 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().

9.29.3.4 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().

9.29.3.5 preview()

void Fl_File_Chooser::preview ( int e )

Enable or disable the preview tile.
1 = enable preview, 0 = disable preview.

9.29.3.6 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);
    ...
}

9.29.4 Member Data Documentation

9.29.4.1 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

9.30 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
}
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)
  Loads 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.

9.30.1 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.
9.30.2 Constructor & Destructor Documentation

9.30.2.1 Fl_File_Icon()

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>filename pattern</td>
</tr>
<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>

9.30.3 Member Function Documentation

9.30.3.1 add()

short * Fl_File_Icon::add ( 
    short d )

Adds a keyword value to the icon array, returning a pointer to it.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>d</td>
</tr>
</tbody>
</table>

9.30.3.2 add_color()

short * Fl_File_Icon::add_color ( 
    Fl_Color c ) [inline]

Adds a color value to the icon array, returning a pointer to it.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>c</td>
</tr>
</tbody>
</table>

9.30.3.3 add_vertex() [1/2]

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
</tr>
</tbody>
</table>

9.30.3.4 add_vertex() [2/2]

short * Fl_File_Icon::add_vertex ( 
    float x, 
    float y, 
    short d, 
    short * nd = 0 ) [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

<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>d</td>
</tr>
<tr>
<td>in</td>
<td>nd</td>
</tr>
</tbody>
</table>

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>vertex coordinates</th>
</tr>
</thead>
</table>

9.30.3.5 draw()

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

<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>ic</td>
<td>icon color</td>
</tr>
<tr>
<td>in</td>
<td>active</td>
<td>status, default is active [non-zero]</td>
</tr>
</tbody>
</table>

9.30.3.6 find()

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

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>name of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>filetype</td>
<td>enumerated file type</td>
</tr>
</tbody>
</table>

Returns

  matching file icon or NULL

9.30.3.7 label()

void Fl_File_Icon::label (  
  Fl_Widget * w )

Applies the icon to the widget, registering the Fl_File_Icon label type as needed.

Parameters

| in | w       | widget for which this icon will become the label |
### 9.30.3.8 labeltype()

```cpp
class Fl_File_Icon {
public:

    // labeltype()
    static void labeltype(
        const Fl_Label * o,
        int x, int y, int w, int h,
        Fl_Align a
    )
```

Draw the icon label.

**Parameters**

- `o` label data
- `x, y, w, h` position and size of label
- `a` label alignment [not used]

### 9.30.3.9 load()

```cpp
class Fl_File_Icon {
public:

    // load()
    void load(
        const char * f
    )
```

Loads the specified icon image. The format is deduced from the filename.

**Parameters**

- `f` filename

### 9.30.3.10 load_fti()

```cpp
class Fl_File_Icon {
public:

    // load_fti()
    int load_fti(
        const char * fti
    )
```

Loads an SGI icon file.

**Parameters**

- `fti` icon filename

**Returns**

0 on success, non-zero on error

### 9.30.3.11 load_image()

```cpp
class Fl_File_Icon {
public:

    // load_image()
    int load_image(
        const char * ifile
    )
```

Load an image icon file from an image filename.

**Parameters**

- `ifile` image filename
Returns

0 on success, non-zero on error

9.30.3.12 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:

```cpp
Fl_File_Icon::load_system_icons();
```

9.30.3.13 next()

```cpp
Fl_File_Icon * Fl_File_Icon::next ( ) [inline]
```

Returns next file icon object.
See `Fl_File_Icon::first()`

9.30.3.14 type()

```cpp
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_Icon.H`
- `Fl_File_Icon.cxx`
- `Fl_File_Icon2.cxx`

9.31 Fl_File_Input Class Reference

This widget displays a pathname in a text input field.

```cpp
#include <Fl_File_Input.H>
```

Inheritance diagram for Fl_File_Input:

```
Fl_Widget
|
V
Fl_Input
|
V
Fl_Input_
|
V
Fl_File_Input
```

Generated by Doxygen
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.**

- virtual int **handle (int event)**
  
  **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)**
  
  **Handles the specified event.**

Public Member Functions inherited from Fl_Input_

- 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.**

- **Fl_Input_ (int, int, int, int, const char ∗=0)**
  
  **Creates a new Fl_Input_ widget.**

- **Fl_Char 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 **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 position of the text cursor.

• int **position** (int p)
  * Sets the cursor position and mark.

• int **position** (int p, int m)
  * Sets the index for the cursor and mark.

• 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 **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)
  * 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()**
  Returns the text displayed in the widget.

- **int value(const char *)**
  Changes the widget text.

- **int value(const char *, int)**
 变化 the widget text.

- **int wrap()**
  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()**
  Activates the widget.

- **void _set_fullscreen()**
  Activates the widget.

- **unsigned int active()**
  Returns whether the widget is active.

- **int active_r()**
  Returns whether the widget and all of its parents are active.

- **Fl_Align align()**
  Gets the label alignment.

- **void align(Fl_Align alignment)**
  Sets the label alignment.

- **long argument()**
  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_Group * as_group()**
  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.

- **Fl_Boxtype box()**
  Gets the box type of the widget.

- **void box(Fl_Boxtype new_box)**
  Sets the box type for the widget.

- **Fl_Callback_p callback()**
  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, 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)
  
  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.

- const Fl_Image *deimage () const
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label.

- void do_callback ()
Calls the widget callback.

- void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.
- void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
- 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.
- const Fl_Image ∗image () const
  Sets the image to use as part of the widget label.
- void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
- 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.
- 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.

- 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** (void ∗v)
  Sets the user data for this widget.

- unsigned int **visible** () const
  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 () 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 void draw ()
  
  Draws the file input widget.

Protected Member Functions inherited from Fl_Input

• void draw ()
  
  Draws the widget.

Protected Member Functions inherited from Fl_Input_

• void drawtext (int, int, int, int)
  
  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 ()

• 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 xsroll () 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_box ()
  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 *cb, void *d)
  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, 
  FULLSCREEN = 1<<17, MAC_USE_ACCENTS_MENU = 1<<18,
  COPYFLAG3 = 1<<29, COPYFLAG2 = 1<<30, COPYFLAG1 = 1<<31 
}

flags possible values enumeration.

9.31.1 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 9.13 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.

9.31.2 Constructor & Destructor Documentation

9.31.2.1 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>

9.31.3 Member Function Documentation

9.31.3.1 down_box

void Fl_File_Input::down_box ( 


Sets the box type to use for the navigation bar.

### 9.31.3.2 draw()

```cpp
void Fl_File_Input::draw ( void ) [protected], [virtual]
```

Draws the file input widget.
Implements Fl_Widget.

### 9.31.3.3 errorcolor()

```cpp
Fl_Color Fl_File_Input::errorcolor ( ) const [inline]
```

Gets the current error color.

**Todo** Better docs for Fl_File_Input::errorcolor() - is it even used?

### 9.31.3.4 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.

### 9.31.3.5 value() [1/2]

```cpp
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 |

### 9.31.3.6 value() [2/2]

```cpp
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:
9.32 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)
  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.

- void 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 Fl_Group * as_group ()

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.
• `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, 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)`
  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)`
• void deactivate()
  Deactivates the widget.
• Fl_Image *deimage()
  Gets the image that is used as part of the widget label.
• const Fl_Image *deimage() const
• void deimage(Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage(Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback()
  Calls the widget callback.
• void do_callback(Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback(Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image() const
• void image(Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image(Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• `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.
• `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 (void *v)**
  Sets the user data for this widget.

- **unsigned int visible () const**
  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 () const**
  Returns whether a widget and all its parents are visible.

- **int w () const**
  Gets the widget width.

- **Fi_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 *cb, void *d)**
  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_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

• void draw ()
  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 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.

### 9.32.1 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`

### 9.33 Fl_Fill_Slider Class Reference

Widget that draws a filled horizontal slider, useful as a progress or value meter.

```cpp
#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=0L)`
  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)
  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 ()
  
  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 Fl_Group** * as_group ()
  
  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.

• **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, 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)
  
  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.

- **const Fl_Image ∗deimage () const**

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image ∗img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗o, void ∗arg=0)**
  Calls the widget callback.

- **void draw_label (int, int, int, Fl_Align) const**
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- **int h () const**

Generated by Doxygen
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.

- const Fl_Image * image () const
- void image (Fl_Image &img)
  Sets the image to use as part of the widget label.

- void image (Fl_Image *img)
  Sets the image to use as part of the widget label.

- 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.

- 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.

- `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 (void * v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  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 () 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 ∗ cb, void ∗ d)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

• void draw ()
  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** ()
  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_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.
9.33.1 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

9.34 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)**
  
  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- **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.

- **Fl_Input_ (int, int, int, int, const char ∗=0)**


---

Generated by Doxygen
Creates a new Fl_Input_ widget.

- Fl_Char 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 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 position of the text cursor.
- int position (int p)
  Sets the cursor position and mark.
- int position (int p, int m)
  Sets the index for the cursor and mark.
- 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 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)
  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 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 Fl_Group ∗ as_group ()`
  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.

• `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, 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)
  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.

• const Fl_image * deimage () const
  
Sets the image to use as part of the widget label.

• void deimage (Fl_image &img)
  
Sets the image to use as part of the widget label.

• void deimage (Fl_image *img)
  
Sets the image to use as part of the widget label.

• void do_callback ()
  
Calls the widget callback.

• void do_callback (Fl_Widget *o, long arg)
  
Calls the widget callback.

• void do_callback (Fl_Widget *o, void *arg=0)
  
Calls the widget callback.

• 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.

• const Fl_image * image () const
  
Sets the image to use as part of the widget label.

• void image (Fl_image &img)
  
Sets the image to use as part of the widget label.

• void image (Fl_image *img)
  
Sets the image to use as part of the widget label.

• 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.

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.

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(void *v)
  Sets the user data for this widget.
- unsigned int visible() const
  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() 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 *cb, void *d)
  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,
      USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
    }
    flags possible values enumeration.
Protected Member Functions inherited from Fl_Input

- void draw ()
  Draws the widget.

Protected Member Functions inherited from Fl_Input_

- void drawtext (int, int, int, int)
  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 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 ()
- 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
  Draws the widget box according its box style.
- void draw_box () const
  Draws the widget box at the widget's position and size.
- 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 ()
  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_label () const
  Draws the widget's label at the given position and size.
- 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.

### 9.34.1 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).

### 9.34.2 Constructor & Destructor Documentation

#### 9.34.2.1 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

### 9.35 Fl_FLTK_File_Chooser Class Reference

Inheritance diagram for Fl_FLTK_File_Chooser:

```
Fl_FLTK/File_Chooser
    |_________________________
    |                        |
    | Fl_FLTK/File_Chooser   |
    |_________________________
    |                        |
    | Fl_GTK/File_Chooser    |
```

Protected Member Functions

- virtual int count () const
- const char * directory () const
- void directory (const char *val)
- const char * errmsg () const
- void errmsg (const char *msg)
- int exist_dialog ()
- virtual const char * filename () const
- virtual const char * filename (int i) const
The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser.H
- Fl_Native_File_Chooser_FLTK.cxx

This a structure for an actual system font, with junk to help choose it and info on character sizes.

#include <Fl_Font.H>

Public Attributes

- Fl_Font_Descriptor * next
- Fl_Fontsize size
9.36.1 Detailed Description

This a structure 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.

The documentation for this class was generated from the following file:

- Fl_Font.H

9.37 Fl_Fontdesc Struct Reference

Public Attributes

- Fl_Font_Descriptor * first
- char fontname[128]
- int n
- const char * name
- char ** xlist

The documentation for this struct was generated from the following file:

- Fl_Font.H

9.38 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 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 `Fl_Group * as_group()`  
  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`.

- `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, 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)`  
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.

- **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.
• 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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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()`
  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()`
  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_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 *cb, void *d)`
  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,
  USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31 } 

flags possible values enumeration.

9.38.1 Detailed Description

Forms compatibility Bitmap Image Widget.

9.38.2 Member Function Documentation

9.38.2.1 draw()

void Fl_FormsBitmap::draw ()
  [protected], [virtual]

Draws the bitmap and its associated box.
Implements Fl_Widget.

9.38.2.2 set()

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

9.39 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**
  
  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_Group ∗ as_group ()**
  
  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.

- **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, 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)**
  
  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.

- `const Fl_Image ∗deimage () const`
  
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image &img)`
  
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image ∗img)`
  
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, long arg)`
  
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, void ∗arg=0)`
  
  Calls the widget callback.

- `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)`

  Generated by Doxygen
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.

- const Fl_Image *image() const
  Sets the image to use as part of the widget label.

- Fl_Image *image(Fl_Image &img)
  Sets the image to use as part of the widget label.

- void image(Fl_Image *img)
  Sets the image to use as part of the widget label.

- 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.

- 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.

- **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 *userData() const**
  Gets the user data for this widget.

- **void userData(void *v)**
  Sets the user data for this widget.

- **unsigned int visible() const**
  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() 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 ()
  
  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 ()
  
  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_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 ∗cb, void ∗d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
  }
  flags possible values enumeration.

9.39.1 Detailed Description

Forms pixmap drawing routines.

9.39.2 Constructor & Destructor Documentation

9.39.2.1 Fl_FormsPixmap()

Fl_FormsPixmap::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.

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>

9.39.3 Member Function Documentation

9.39.3.1 draw()

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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

### 9.39.3.2 Pixmap()

```c
void Fl_FormsPixmap::Pixmap ( Fl_Pixmap * B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

**Parameters**

- **in** \( B \) existing pixmap

### 9.39.3.3 set()

```c
void Fl_FormsPixmap::set ( char *const * bits )
```

Set/create the internal pixmap using raw data.

**Parameters**

- **in** \( bits \) raw data

The documentation for this class was generated from the following files:

- Fl_FormsPixmap.H
- forms_pixmap.cxx

### 9.40 Fl_FormsText Class Reference

Inheritance diagram for Fl_FormsText:

```
Fl_Widget

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 Fl_Group * as_group ()
  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.

• Fl_BoxLayout box () const
  Gets the box type of the widget.

• void box (Fl_BoxLayout 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, 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)
  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.

- **const Fl_Image * deimage () const**

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image * image () const**

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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.
• 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.
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` (void ∗v)
  Sets the user data for this widget.
- unsigned int `visible` () const
  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` () 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** ()
  
  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** ()
  
  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_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 ∗cb, void ∗d)
  
  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, COPYED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCESTS_MENU
    = 1<<19,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.40.1 Member Function Documentation

9.40.1.1 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 = &scroll; // scroll 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

9.41 Fl_Free Class Reference

Emulation of the Forms "free" widget.
#include <Fl_Free.H>

Inheritance diagram for Fl_Free:

```
   Fl_Widget
      |
      v
   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)
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 ()

Generated by Doxygen
• 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 `Fl_Group` * `as_group()`
  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`.
• `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, 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)
  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.

- `const Fl_Image ∗deimage () const`

- `void deimage (Fl_Image &img)`

- `void deimage (Fl_Image ∗img)`

- `void do_callback ()`
  
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, long arg)`
  
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, void ∗arg=0)`
  
  Calls the widget callback.

- `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.

- `const Fl_Image ∗image () const`

- `void image (Fl_Image &img)`

- `void image (Fl_Image ∗img)`

- `void 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.

• 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.

- 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 `takevents()` 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(void * v)`
  Sets the user data for this widget.

- unsigned int `visible()` const
  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()` 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 ()
  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 ()
  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_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 ∗ cb, void ∗ d)
  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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }

flags possible values enumeration.

9.41.1 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.

9.41.2 Constructor & Destructor Documentation

9.41.2.1 Fl_Free()

Fl_Free::Fl_Free ( uchar t,
  int X, Y, W, 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</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X, Y, W, H</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
</tr>
<tr>
<td>in</td>
<td>hdl</td>
</tr>
</tbody>
</table>

The constructor takes both the type and the handle function. The handle function should be declared as follows:

```
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.

Generated by Doxygen
The event argument contains the event type:
// 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

9.41.3 Member Function Documentation

9.41.3.1 draw()

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:
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Implements Fl_Widget.

9.41.3.2 handle()

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.

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

9.42 Fl_GDI_Graphics_Driver Class Reference

The MSWindows-specific graphics class.
#include <Fl_Device.H>
Inheritance diagram for Fl_GDI_Graphics_Driver:
Fl_GDI_Graphics_Driver
Fl_Graphics_Driver
Fl_Device
Fl_GDI_Printer_Graphics_Driver

Public Member Functions

- const char * class_name ()
  Returns the name of the class of this object.
- void color (Fl_Color c)
  see fl_color(Fl_Color c).
- void color (uchar r, uchar g, uchar b)
  see fl_color(uchar r, uchar g, uchar b).
- void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  see fl_copy_offscreen()
- int descent ()
  see fl_descent().
- void draw (const char *str, int n, int x, int y)
  see fl_draw(const char *str, int n, int x, int y).
- void draw (Fl_Bitmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  see fl_width(unsigned int n).
- void draw (Fl_Bitmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Bitmap object to the device.
- void draw (Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Pixmap object to the device.
- void draw (Fl_RGB_Image *img, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_RGB_Image object to the device.
- void draw (int angle, const char *str, int n, int x, int y)
  see fl_draw(int angle, const char *str, int n, int x, int y).
- void draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  see fl_draw_image(const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0).
- void draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  see fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
- void draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  see fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0).
- void draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D=3).
- void font (Fl_Font face, Fl_Fontsize size)
  see fl_font(Fl_Font face, Fl_Fontsize size).
- int height ()
  see fl_height().
- void rtl_draw (const char *str, int n, int x, int y)
  see fl_rtl_draw(const char *str, int n, int x, int y).
- void text_extents (const char *, int n, int &dx, int &dy, int &w, int &h)
  see fl_text_extents(const char *, int n, int &dx, int &dy, int &w, int &h).
- double width (const char *str, int n)
  see fl_width(const char *str, int n).
- double width (unsigned int n)
  see fl_width(unsigned int n).
Public Member Functions inherited from Fl_Graphics_Driver

- \texttt{Fl\_Color color ()}
  
  see \texttt{fl\_color(void)}.

- \texttt{virtual int draw\_scaled (Fl\_Image *img, int X, int Y, int W, int H)}
  
  Draws an \texttt{Fl\_Image} scaled to width \(W\) & height \(H\) with top-left corner at \(X,Y\).

- \texttt{Fl\_Font font ()}
  
  see \texttt{fl\_font(void)}.

- \texttt{Fl\_Font\_Descriptor * font\_descriptor ()}
  
  Returns a pointer to the current \texttt{Fl\_Font\_Descriptor} for the graphics driver.

- \texttt{void font\_descriptor (Fl\_Font\_Descriptor *d)}
  
  Sets the current \texttt{Fl\_Font\_Descriptor} for the graphics driver.

- \texttt{Fl\_Fontsize size ()}
  
  see \texttt{fl\_size()}

- \texttt{virtual \~Fl\_Graphics\_Driver ()}
  
  The destructor.

Public Member Functions inherited from Fl\_Device

- \texttt{virtual \~Fl\_Device ()}
  
  Virtual destructor.

Static Public Attributes

- \texttt{static const char * class\_id = "Fl\_GDI\_Graphics\_Driver"}

Static Public Attributes inherited from Fl\_Graphics\_Driver

- \texttt{static const char * class\_id = "Fl\_Graphics\_Driver"}

Static Public Attributes inherited from Fl\_Device

- \texttt{static const char * class\_id = "Fl\_Device"}
  
  A string that identifies each subclass of \texttt{Fl\_Device}.

Additional Inherited Members

Protected Member Functions inherited from Fl\_Graphics\_Driver

- \texttt{virtual void arc (double x, double y, double r, double start, double end)}
  
  see \texttt{fl\_arc(double x, double y, double r, double start, double end)}.

- \texttt{virtual void arc (int x, int y, int w, int h, double a1, double a2)}
  
  see \texttt{fl\_arc(int x, int y, int w, int h, double a1, double a2)}.

- \texttt{virtual void begin\_complex\_polygon ()}
  
  see \texttt{fl\_begin\_complex\_polygon()}. 

- \texttt{virtual void begin\_line ()}
  
  see \texttt{fl\_begin\_line()}. 

- \texttt{virtual void begin\_loop ()}
  
  see \texttt{fl\_begin\_loop()}. 

- \texttt{virtual void begin\_points ()}
  
  see \texttt{fl\_begin\_points()}. 

- \texttt{virtual void begin\_polygon ()}
  
  see \texttt{fl\_begin\_polygon()}. 

- \texttt{virtual void circle (double x, double y, double r)}
9.42 Fl_GDI_Graphics_Driver Class Reference

- `virtual int clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)`
  - `see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).`

- `Fl_Graphics_Driver`()
  - The constructor.

- `virtual int not_clipped (int x, int y, int w, int h)`
  - `see fl_not_clipped(int x, int y, int w, int h).`

- `Fl_Region clip_region ()`
  - `see fl_clip_region().`

- `void clip_region (Fl_Region r)`
  - `see fl_clip_region(Fl_Region r).`

- `virtual void curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)`
  - `see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).`

- `virtual void end_complex_polygon ()`
  - `see fl_end_complex_polygon().`

- `virtual void end_line ()`
  - `see fl_end_line().`

- `virtual void end_loop ()`
  - `see fl_end_loop().`

- `virtual void end_points ()`
  - `see fl_end_points().`

- `virtual void end_polygon ()`
  - `see fl_end_polygon().`

- `void mult_matrix (double a, double b, double c, double d, double x, double y)`
  - `see fl_mult_matrix(double a, double b, double c, double d, double x, double y).`

- `virtual int not_clipped (int x, int y, int w, int h)`
  - `see fl_not_clipped(int x, int y, int w, int h).`

- `void push_matrix ()`
  - `see fl_push_matrix().`

- `virtual void push_clip (int x, int y, int w, int h)`
  - `see fl_push_clip(int x, int y, int w, int h).`
• void push_matrix()
  see fl_push_matrix().
• virtual void push_no_clip()
  see fl_push_no_clip().
• virtual void rect (int x, int y, int w, int h)
  see fl_rect(int x, int y, int w, int h).
• virtual void rectf (int x, int y, int w, int h)
  see fl_rectf(int x, int y, int w, int h).
• void restore_clip()
  see fl_restore_clip().
• void rotate (double d)
  see fl_rotate(double d).
• void scale (double x)
  see fl_scale(double x).
• void scale (double x, double y)
  see fl_scale(double x, double y).
• double transform_dx (double x, double y)
  see fl_transform_dx(double x, double y).
• double transform_dy (double x, double y)
  see fl_transform_dy(double x, double y).
• double transform_x (double x, double y)
  see fl_transform_x(double x, double y).
• double transform_y (double x, double y)
  see fl_transform_y(double x, double y).
• virtual void transformed_vertex (double xf, double yf)
  see fl_transformed_vertex(double xf, double yf).
• void translate (double x, double y)
  see fl_translate(double x, double y).
• virtual void vertex (double x, double y)
  see fl_vertex(double x, double y).
• virtual void xyline (int x, int y, int x1)
  see fl_xyline(int x, int y, int x1).
• virtual void xyline (int x, int y, int x1, int y2)
  see fl_xyline(int x, int y, int x1, int y2).
• virtual void xyline (int x, int y, int x1, int y2, int x3)
  see fl_xyline(int x, int y, int x1, int y2, int x3).
• virtual void yxline (int x, int y, int y1)
  see fl_yxline(int x, int y, int y1).
• virtual void yxline (int x, int y, int y1, int x2)
  see fl_yxline(int x, int y, int y1, int x2).
• virtual void yxline (int x, int y, int y1, int x2, int y3)
  see fl_yxline(int x, int y, int y1, int x2, int y3).

Protected Attributes inherited from Fl_Graphics_Driver

• matrix * fl_matrix
  Points to the current coordinate transformation matrix.

9.42.1 Detailed Description

The MSWindows-specific graphics class.
This class is implemented only on the MSWindows platform.
9.42.2 Member Function Documentation

9.42.2.1 class_name()

const char * Fl_GDI_Graphics_Driver::class_name ( )  [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:

```
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```
Reimplemented from Fl_Graphics_Driver.
Reimplemented in Fl_GDI_Printer_Graphics_Driver.

9.42.2.2 color() [1/2]

void Fl_GDI_Graphics_Driver::color ( Fl_Color c )  [virtual]
see fl_color(Fl_Color c).
Reimplemented from Fl_Graphics_Driver.

9.42.2.3 color() [2/2]

void Fl_GDI_Graphics_Driver::color ( uchar r, uchar g, uchar b )  [virtual]
see fl_color(uchar r, uchar g, uchar b).
Reimplemented from Fl_Graphics_Driver.

9.42.2.4 copy_offscreen()

void Fl_GDI_Graphics_Driver::copy_offscreen ( int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy )  [virtual]
see fl_copy_offscreen()
Reimplemented from Fl_Graphics_Driver.

9.42.2.5 descent()

int Fl_GDI_Graphics_Driver::descent ( )  [virtual]
see fl_descent()
Reimplemented from Fl_Graphics_Driver.

9.42.2.6 draw() [1/5]

void Fl_GDI_Graphics_Driver::draw ( const char * str, int n, int x, int y )  [virtual]
see fl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.42.2.7 draw() [2/5]

void Fl_GDI_Graphics_Driver::draw ( Fl_Bitmap * bm,
Draws an Fl_Bitmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.
Reimplemented in Fl_GDI_Printer_Graphics_Driver.

9.42.2.8 draw() [3/5]

```c
void Fl_GDI_Graphics_Driver::draw (Fl_Pixmap ∗pxm, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
```

Draws an Fl_Pixmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.
Reimplemented in Fl_GDI_Printer_Graphics_Driver.

9.42.2.9 draw() [4/5]

```c
void Fl_GDI_Graphics_Driver::draw (Fl_RGB_Image ∗rgb, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
```

Draws an Fl_RGB_Image object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.42.2.10 draw() [5/5]

```c
void Fl_GDI_Graphics_Driver::draw (int angle, const char ∗str, int n, int x, int y) [virtual]
```

see Fl_draw(int angle, const char ∗str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.42.2.11 draw_image() [1/2]

```c
void Fl_GDI_Graphics_Driver::draw_image (const uchar ∗buf, int X, int Y)
```
int Y,
int W,
int H,
int D = 3,
int L = 0) [virtual]
see fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.42.2.12 draw_image() [2/2]

void Fl_GDI_Graphics_Driver::draw_image(
    Fl_Draw_Image_Cb cb,
    void * data,
    int X,
    int Y,
    int W,
    int H,
    int D = 3) [virtual]
see fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
Reimplemented from Fl_Graphics_Driver.

9.42.2.13 draw_image_mono() [1/2]

void Fl_GDI_Graphics_Driver::draw_image_mono(
    const uchar * buf,
    int X,
    int Y,
    int W,
    int H,
    int D = 1,
    int L = 0) [virtual]
see fl_draw_imageMono(const uchar * buf, int X, int Y, int W, int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.42.2.14 draw_image_mono() [2/2]

void Fl_GDI_Graphics_Driver::draw_image_mono(
    Fl_Draw_Image_Cb cb,
    void * data,
    int X,
    int Y,
    int W,
    int H,
    int D = 1) [virtual]
see fl_draw_imageMono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
Reimplemented from Fl_Graphics_Driver.

9.42.2.15 font()

void Fl_GDI_Graphics_Driver::font(
    Fl_Font face,
    Fl_Fontsize fsize) [virtual]
see fl_font(Fl_Font face, Fl_Fontsize size).
Reimplemented from Fl_Graphics_Driver.

9.42.2.16 height()

int Fl_GDI_Graphics_Driver::height() [virtual]
see fl_height().
Reimplemented from Fl_Graphics_Driver.

9.42.2.17 rtl_draw()

void Fl_GDI_Graphics_Driver::rtl_draw (  
    const char ∗str,  
    int n,  
    int x,  
    int y ) [virtual]

see fl_rtl_draw(const char ∗str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.42.2.18 text_extents()

void Fl_GDI_Graphics_Driver::text_extents (  
    const char ∗t,  
    int n,  
    int &dx,  
    int &dy,  
    int &w,  
    int &h ) [virtual]

see fl_text_extents(const char∗, int n, int& dx, int& dy, int& w, int& h).
Reimplemented from Fl_Graphics_Driver.

9.42.2.19 width() [1/2]

double Fl_GDI_Graphics_Driver::width (  
    const char ∗str,  
    int n ) [virtual]

see fl_width(const char ∗str, int n).
Reimplemented from Fl_Graphics_Driver.

9.42.2.20 width() [2/2]

double Fl_GDI_Graphics_Driver::width (  
    unsigned int c ) [virtual]

see fl_width(unsigned int n).
Reimplemented from Fl_Graphics_Driver.

The documentation for this class was generated from the following files:

- Fl_Device.H
- fl_color_win32.cxx
- Fl_Device.cxx
- fl_draw_image_win32.cxx

9.43 Fl_GDI_Printer_Graphics_Driver Class Reference

The graphics driver used when printing on MSWindows.
#include <Fl_Device.H>
Inheritance diagram for Fl_GDI_Printer_Graphics_Driver:
Public Member Functions

- const char * class_name ()
  Returns the name of the class of this object.
- void draw (Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Bitmap object to the device.
- void draw (Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Pixmap object to the device.
- int draw_scaled (Fl_Image *img, int XP, int YP, int WP, int HP)
  Draws an Fl_Image scaled to width \( W \) & height \( H \) with top-left corner at \( X,Y \).

Public Member Functions inherited from Fl_GDI_Graphics_Driver

- void color (Fl_Color c)
  see fl_color(Fl_Color c).
- void color (uchar r, uchar g, uchar b)
  see fl_color(uchar r, uchar g, uchar b).
- void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  see fl_copy_offscreen()
- int descent ()
  see fl_descent().
- void draw (const char *str, int n, int x, int y)
  see fl_draw(const char *str, int n, int x, int y).
- void draw (Fl_RGB_Image *img, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_RGB_Image object to the device.
- void draw (int angle, const char *str, int n, int x, int y)
  see fl_draw(int angle, const char *str, int n, int x, int y).
- void draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  see fl_draw_image(const uchar* buf, int X, int Y, int W, int H, int D, int L).
- void draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  see fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
- void draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)
  see fl_draw_image_mono(const uchar* buf, int X, int Y, int W, int H, int D, int L).
- void draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)
  see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
- void font (Fl_Font face, Fl_Fontsize size)
  see fl_font(Fl_Font face, Fl_Fontsize size).
- int height ()
  see fl_height().
- void rtl_draw (const char *str, int n, int x, int y)
  see fl_rtl_draw(const char *str, int n, int x, int y).
• void text_extents (const char *, int n, int &dx, int &dy, int &w, int &h)
  
  see fl_text_extents(const char*, int n, int& dx, int& dy, int& w, int& h).

• double width (const char *str, int n)
  
  see fl_width(const char *str, int n).

• double width (unsigned int c)
  
  see fl_width(unsigned int n).

Public Member Functions inherited from Fl_Graphics_Driver

  • Fl_Color color ()
    
    see fl_color(void).

  • Fl_Font font ()
    
    see fl_font(void).

  • Fl_Font_Descriptor *font_descriptor ()
    Returns a pointer to the current Fl_Font_Descriptor for the graphics driver.

  • void font_descriptor (Fl_Font_Descriptor *d)
    
    Sets the current Fl_Font_Descriptor for the graphics driver.

  • Fl_Fontsize size ()
    
    see fl_size().

  • virtual ~Fl_Graphics_Driver ()
    
    The destructor.

Public Member Functions inherited from Fl_Device

  • virtual ~Fl_Device ()
    
    Virtual destructor.

Static Public Attributes

  • static const char *class_id = "Fl_GDI_Printer_Graphics_Driver"

Static Public Attributes inherited from Fl_GDI_Graphics_Driver

  • static const char *class_id = "Fl_GDI_Graphics_Driver"

Static Public Attributes inherited from Fl_Graphics_Driver

  • static const char *class_id = "Fl_Graphics_Driver"

Static Public Attributes inherited from Fl_Device

  • static const char *class_id = "Fl_Device"
    
    A string that identifies each subclass of Fl_Device.

Additional Inherited Members

Protected Member Functions inherited from Fl_Graphics_Driver

  • virtual void arc (double x, double y, double r, double start, double end)
    
    see fl_arc(double x, double y, double r, double start, double end).

  • virtual void arc (int x, int y, int w, int h, double a1, double a2)
    
    see fl_arc(int x, int y, int w, int h, double a1, double a2).

  • virtual void begin_complex_polygon ()
    
    see fl_begin_complex_polygon().

  • virtual void begin_line ()
9.43 Fl_GDI_Printer_Graphics_Driver Class Reference

- virtual void `begin_loop()`
  - see `fl_begin_loop()`.
- virtual void `begin_points()`
  - see `fl_begin_points()`.
- virtual void `begin_polygon()`
  - see `fl_begin_polygon()`.
- virtual void `circle(double x, double y, double r)`
  - see `fl_circle(double x, double y, double r)`.
- virtual int `clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H)`
  - see `fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H)`.
- `Fl_Region clip_region()`
  - see `fl_clip_region()`.
- void `clip_region(Fl_Region r)`
  - see `fl_clip_region(Fl_Region r)`.
- virtual void `curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)`
  - see `fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)`.
- virtual void `end_complex_polygon()`
  - see `fl_end_complex_polygon()`.
- virtual void `end_line()`
  - see `fl_end_line()`.
- virtual void `end_loop()`
  - see `fl_end_loop()`.
- virtual void `end_points()`
  - see `fl_end_points()`.
- virtual void `end_polygon()`
  - see `fl_end_polygon()`.
- `Fl_Graphics_Driver()`
  - The constructor.
- virtual void `gap()`
  - see `fl_gap()`.
- virtual void `line(int x, int y, int x1, int y1)`
  - see `fl_line(int x, int y, int x1, int y1)`.
- virtual void `line(int x, int y, int x1, int y1, int x2, int y2)`
  - see `fl_line(int x, int y, int x1, int y1, int x2, int y2)`.
- virtual void `line_style(int style, int width=0, char *dashes=0)`
  - see `fl_line_style(int style, int width, char *dashes)`.
- virtual void `loop(int x0, int y0, int x1, int y1, int x2, int y2)`
  - see `fl_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)`
  - see `fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`.
- void `mult_matrix(double a, double b, double c, double d, double x, double y)`
  - see `fl_mult_matrix(double a, double b, double c, double d, double x, double y)`.
- virtual int `not_clipped(int x, int y, int w, int h)`
  - see `fl_not_clipped(int x, int y, int w, int h)`.
- virtual void `pie(int x, int y, int w, int h, double a1, double a2)`
  - see `fl_pie(int x, int y, int w, int h, double a1, double a2)`.
- virtual void `point(int x, int y)`
  - see `fl_point(int x, int y)`.
- virtual void `polygon(int x0, int y0, int x1, int y1, int x2, int y2)`
  - see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2)`.

Generated by Doxygen
• virtual void polygon (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  
  see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).

• virtual void pop_clip ()
  
  see fl_pop_clip().

• void pop_matrix ()
  
  see fl_pop_matrix().

• virtual void push_clip (int x, int y, int w, int h)
  
  see fl_push_clip(int x, int y, int w, int h).

• void push_matrix ()
  
  see fl_push_matrix().

• virtual void push_no_clip ()
  
  see fl_push_no_clip().

• virtual void rect (int x, int y, int w, int h)
  
  see fl_rect(int x, int y, int w, int h).

• virtual void rectf (int x, int y, int w, int h)
  
  see fl_rectf(int x, int y, int w, int h).

• void restore_clip ()
  
  see fl_restore_clip().

• void rotate (double d)
  
  see fl_rotate(double d).

• void scale (double x)
  
  see fl_scale(double x).

• void scale (double x, double y)
  
  see fl_scale(double x, double y).

• double transform_dx (double x, double y)
  
  see fl_transform_dx(double x, double y).

• double transform_dy (double x, double y)
  
  see fl_transform_dy(double x, double y).

• double transform_x (double x, double y)
  
  see fl_transform_x(double x, double y).

• double transform_y (double x, double y)
  
  see fl_transform_y(double x, double y).

• virtual void transformed_vertex (double xf, double yf)
  
  see fl_transformed_vertex(double xf, double yf).

• void translate (double x, double y)
  
  see fl_translate(double x, double y).

• virtual void vertex (double x, double y)
  
  see fl_vertex(double x, double y).

• virtual void xyline (int x, int y, int x1)
  
  see fl_xyline(int x, int y, int x1).

• virtual void xyline (int x, int y, int x1, int y1)
  
  see fl_xyline(int x, int y, int x1, int y1).

• virtual void xyline (int x, int y, int x1, int y2)
  
  see fl_xyline(int x, int y, int x1, int y2).

• virtual void xyline (int x, int y, int x1, int y2, int x3)
  
  see fl_xyline(int x, int y, int x1, int y2, int x3).

• virtual void yxline (int x, int y, int y1)
  
  see fl_yxline(int x, int y, int y1).

• virtual void yxline (int x, int y, int y1, int x2)
  
  see fl_yxline(int x, int y, int y1, int x2).

• virtual void yxline (int x, int y, int y1, int x2, int y3)
  
  see fl_yxline(int x, int y, int y1, int x2, int y3).
9.43 Fl_GDI_Printer_Graphics_Driver Class Reference

Protected Attributes inherited from Fl_Graphics_Driver

• matrix * fl_matrix
  
  Points to the current coordinate transformation matrix.

9.43.1 Detailed Description

The graphics driver used when printing on MSWindows.
This class is implemented only on the MSWindows platform. It's extremely similar to Fl_GDI_Graphics_Driver.

9.43.2 Member Function Documentation

9.43.2.1 class_name()

const char * Fl_GDI_Printer_Graphics_Driver::class_name ( ) [inline], [virtual]

Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
  if ( instance->class_name() == Fl_Printer::class_name() ) { ... }
Reimplemented from Fl_GDI_Graphics_Driver.

9.43.2.2 draw() [1/2]

void Fl_GDI_Printer_Graphics_Driver::draw (  
    Fl_Bitmap * bm,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy ) [virtual]

Draws an Fl_Bitmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_GDI_Graphics_Driver.

9.43.2.3 draw() [2/2]

void Fl_GDI_Printer_Graphics_Driver::draw (  
    Fl_Pixmap * pxm,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy ) [virtual]

Draws an Fl_Pixmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_GDI_Graphics_Driver.

9.43.2.4 draw_scaled()

int Fl_GDI_Printer_Graphics_Driver::draw_scaled (  
    Fl_Image * img,  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]

Draws an Fl_Image scaled to width W & height H with top-left corner at X.Y.
Returns

zero when the graphics driver doesn't implement scaled drawing, non-zero if it does implement it.

Reimplemented from Fl_Graphics_Driver.

The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

9.44 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_Pixmap
  Fl_GIF_Image
```

**Public Member Functions**

- `Fl_GIF_Image (const char ∗filename)`
  
The constructor loads the named GIF image.

**Public Member Functions inherited from Fl_Pixmap**

- virtual `void color_average (Fl_Color c, float i)`
  
The color_average() method averages the colors in the image with the FLTK color value c.

- virtual `Fl_Image ∗copy ()`

- virtual `Fl_Image ∗copy (int W, int H)`

- `void draw (int X, int Y)`

- `void draw (int X, int Y, int W, int H, int cx=0, int cy=0)`
  
  Draws the image 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)`

- `Fl_Pixmap (uchar ∗const ∗D)`

- virtual `void label (Fl_Menu_Item ∗m)`

- virtual `void label (Fl_Widget ∗w)`

  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual `∼Fl_Pixmap ()`

  The destructor frees all memory and server resources that are used by the pixmap.

- virtual `void uncache ()`
  
  If the image has been cached for display, delete the cache data.

- virtual `∼Fl_Pixmap ()`
Public Member Functions inherited from Fl_Image

• Fl_Image * copy ()
  The copy() method creates a copy of the specified image.

• int count () const
  The count() method returns the number of data values associated with the image.

• int d () const
  Returns the current image depth.

• const char * const * data () const
  Returns a pointer to the current image data array.

• void draw (int X, int Y)
  Draws the image.

• int fail ()
  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 height in pixels.

• void inactive ()
  The inactive() method calls color_average(FLBACKGROUND_COLOR, 0.33f) to produce an image that appears
grayed out.

• int ld () const
  Returns the current line data size in bytes.

• int w () const
  Returns the current image width in pixels.

• 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_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).

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_NO_IMAGE = -1

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 array 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.
- void h (int H)
  Sets the current image height in pixels.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the current image width in pixels.

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)

9.44.1 Detailed Description

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF^SM^ images. The class loads the first image and supports transparency.

9.44.2 Constructor & Destructor Documentation

9.44.2.1 Fl_GIF_Image()

Fl_GIF_Image::Fl_GIF_Image (const char * infname )
const char * infname )
The constructor loads the named GIF image.
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.
The documentation for this class was generated from the following files:

- Fl_GIF_Image.H
- Fl_GIF_Image.cxx

9.45 Fl_Gl_Choice Class Reference

Static Public Member Functions

- static Fl_Gl_Choice * find (int mode, const int *)

Public Attributes

- GLXFBConfig best_fb
- Colormap colormap
- XVisualInfo * vis

The documentation for this class was generated from the following files:

- Fl_Gl_Choice.H
- Fl_Gl_Choice.cxx
9.46 Fl_Gl_Window Class Reference

The Fl_Gl_Window widget sets things up so OpenGL works.

```
#include <Fl_Gl_Window.H>
```

Inheritance diagram for Fl_Gl_Window:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Window
    ↓
Fl_Gl_Window
```

Public Member Functions

- **virtual Fl_Gl_Window * as_gl_window ()**
  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.

- **void * context () const**
  Returns a pointer to the GLContext that this window is using.

- **void context (void *, 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 ()**
  Forces the window to be drawn, this window is also made current and calls draw().

- **int handle (int)**
  Handle some FLTK events as needed.

- **void hide ()**
  Hides the window and destroys the OpenGL context.

- **void hide_overlay ()**
  Hides the window if it is not this window, does nothing in WIN32.

- **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 ()**
  The make_overlay_current() method 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 ()**
  This method causes draw_overlay() to be called at a later time.

• **void resize (int, int, int, int)**
  Changes the size and position of the window.

• **void show ()**
  Puts the window on the screen.

• **void show (int a, char **b)**

• **void swap_buffers ()**
  The swap_buffers() method swaps the back and front 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 Fl_Window * as_window ()**
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• **unsigned int border () const**
  See void Fl_Window::border(int)

• **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.

• **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 \texttt{decorated\_h} ()
  Returns the window height including any window title bar and any frame added by the window manager.

• int \texttt{decorated\_w} ()
  Returns the window width including any frame added by the window manager.

• void \texttt{default\_cursor} (\texttt{Fl\_Cursor} \texttt{c}, \texttt{Fl\_Color}, \texttt{Fl\_Color}=\texttt{FL\_WHITE})
  For back compatibility only.

• void \texttt{default\_cursor} (\texttt{Fl\_Cursor})
  Sets the default window cursor.

• \texttt{Fl\_Window} (int \texttt{w}, int \texttt{h}, const char \texttt{∗} \texttt{title}=0)
  Creates a window from the given size and title.

• \texttt{Fl\_Window} (int \texttt{x}, int \texttt{y}, int \texttt{w}, int \texttt{h}, const char \texttt{∗} \texttt{title}=0)
  Creates a window from the given position, size and title.

• void \texttt{free\_position} ()
  Undoes the effect of a previous \texttt{resize()} or \texttt{show()} so that the next time \texttt{show()} is called the window manager is free to position the window.

• void \texttt{fullscreen} ()
  Makes the window completely fill one or more screens, without any window manager border visible.

• unsigned int \texttt{fullscreen\_active} () const
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.

• void \texttt{fullscreen\_off} ()
  Turns off any side effects of \texttt{fullscreen()}

• void \texttt{fullscreen\_off} (int \texttt{X}, int \texttt{Y}, int \texttt{W}, int \texttt{H})
  Turns off any side effects of \texttt{fullscreen()} and does \texttt{resize(x,y,w,h)}.

• void \texttt{fullscreen\_screens} (int \texttt{top}, int \texttt{bottom}, int \texttt{left}, int \texttt{right})
  Sets which screens should be used when this window is in fullscreen mode.

• void \texttt{hotspot} (const \texttt{Fl\_Widget \&p}, int \texttt{offscreen}=0)
  See void \texttt{Fl\_Window::hotspot(int \texttt{x}, int \texttt{y}, int \texttt{offscreen} = 0)}

• void \texttt{hotspot} (const \texttt{Fl\_Widget \*}, int \texttt{offscreen}=0)
  See void \texttt{Fl\_Window::hotspot(int \texttt{x}, int \texttt{y}, int \texttt{offscreen} = 0)}

• void \texttt{hotspot} (int \texttt{x}, int \texttt{y}, int \texttt{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 \texttt{∗} \texttt{icon} () const
  Gets the current icon window target dependent data.

• void \texttt{icon} (const \texttt{Fl\_RGB\_Image \*})
  Sets or resets a single window icon.

• void \texttt{icon} (const void \texttt{∗}{\texttt{ic}})
  Sets the current icon window target dependent data.

• void \texttt{iconize} ()
  Iconifies the window.

• const char \texttt{∗} \texttt{iconlabel} () const
  See void \texttt{Fl\_Window::iconlabel(const char\∗)}

• void \texttt{iconlabel} (const char \texttt{∗})
  Sets the icon label.

• void \texttt{icons} (const \texttt{Fl\_RGB\_Image \*}[\texttt{}], int)
  Sets the window icons.

• const char \texttt{∗} \texttt{label} () const
  See void \texttt{Fl\_Window::label(const char\∗)}

• void \texttt{label} (const char \texttt{∗})
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.
- 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.
- unsigned int override () const
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.
- 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|FL_OVERRIDE.
- void set_tooltip_window ()
  Marks the window as a tooltip window.
- 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 (int argc, char **argv)
  Puts the window on the screen 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 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.
- `virtual Fl_Group * as_group ()`
  
  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 &, 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`
  
  See void `Fl_Group::resizable(Fl_Widget *box)`
• **void resizable** (Fl_Widget &o)
  
  See void Fl_Group::resizable(Fl_Widget *box)

• **void resizable** (Fl_Widget *o)
  
  The resizable widget defines the resizing box for the group.

• **virtual ~Fl_Group** ()
  
  The destructor also deletes all the children.

### 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.
- **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, 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)**
  
  Sets the current callback function for the widget.
- **unsigned int changed** () const
  
  Checks if the widget value changed since the last callback.
- **void clearActive** ()
  
  Marks the widget as inactive without sending events or changing focus.
- **void clearChanged** ()
  
  Marks the value of the widget as unchanged.
- **void clearDamage (uchar c=0)**
  
  Clears or sets the damage flags.
- **void clearOutput** ()
  
  Sets a widget to accept input.
- **void clearVisible** ()
  
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
Sets the image to use as part of the widget label.

- 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 labelfont () const
  Gets the label type.

- void labelfont (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.

- 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.*

• 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 **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** (void ∗v)
  *Sets the user data for this widget.*

• unsigned int **visible** () const
  *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** () 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 *)
  Sets a single default window icon.

- static void default_icons (const Fl_RGB_Image *[], int)
  Sets the default window icons.

- static const char * default_xclass ()
  Returns the default xclass.

- static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.

- static void current (Fl_Group *)
  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 labelShortcut (const char *)
  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 Member Functions

- virtual void draw ()
  Draws the Fl_Gl_Window.

Protected Member Functions inherited from Fl_Window

- 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.
Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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.

Friends

- class _FI_Gl_Overlay
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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

Protected Attributes inherited from Fl_Widget

- shape_data_type * shape_data_

  non-null means the window has a non-rectangular shape

Static Protected Attributes inherited from Fl_Widget

- static Fl_Widget * current_

  Stores the last window that was made current.

9.46.1 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. Even though Fl_Gl_Window is derived from Fl_Group, it is not useful to add other FLTK Widgets as children, unless those widgets are modified to draw using OpenGL calls.

9.46.2 Constructor & Destructor Documentation

9.46.2.1 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.

9.46.2.2 Fl_Gl_Window() [2/2]

Fl_Gl_Window::Fl_Gl_Window ( int X, int Y, int W, int H, const char * l = 0 ) [inline]

Creates a new Fl_Gl_Window widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

### 9.46.3 Member Function Documentation

#### 9.46.3.1 as_gl_window()

```cpp
virtual Fl_Gl_Window * Fl_Gl_Window::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 from Fl_Widget.

#### 9.46.3.2 can_do()

```cpp
static int Fl_Gl_Window::can_do ( const int * m ) [inline], [static]
```

Returns non-zero if the hardware supports the given OpenGL mode.

**See also**

Fl_Gl_Window::mode(const int *a)

#### 9.46.3.3 can_do_overlay()

```cpp
int Fl_Gl_Window::can_do_overlay ( )
```

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.

#### 9.46.3.4 context()[1/2]

```cpp
void * Fl_Gl_Window::context ( ) const [inline]
```

Returns a pointer to the GLContext that this window is using.

**See also**

void context(void* v, int destroy_flag)

#### 9.46.3.5 context()[2/2]

```cpp
void Fl_Gl_Window::context ( void * 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.

---

Generated by Doxygen
9.46.3.6  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.
```

9.46.3.7  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 must override the draw() method.
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.
Reimplemented from Fl_Window.
Reimplemented in Fl_Glut_Window.
```

9.46.3.8  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.
```

9.46.3.9  handle()

```cpp
int Fl_Gl_Window::handle ( )
Handle some FLTK events as needed.
Reimplemented from Fl_Window.
Reimplemented in Fl_Glut_Window.
```

9.46.3.10  hide()

```cpp
void Fl_Gl_Window::hide ( ) [virtual]
Hides the window and destroys the OpenGL context.
Reimplemented from Fl_Window.
```

9.46.3.11  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.
```

9.46.3.12  make_overlay_current()

```cpp
void Fl_Gl_Window::make_overlay_current ( )
The make_overlay_current() method selects the OpenGL context for the widget's overlay.
It 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.
```

9.46.3.13  mode() [1/3]

```cpp
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).

### 9.46.3.14 mode() [2/3]

```cpp
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).

**MSWindows 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.

### 9.46.3.15 mode() [3/3]

```cpp
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. See more details in Using OpenGL 3.0 (or higher versions).
9.46.3.16 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.

9.46.3.17 pixel_h()

int Fl_Gl_Window::pixel_h ( ) [inline]
Gives the window height in OpenGL pixels.
Generally identical with the result of the h() function, but for a window mapped to an Apple 'retina' display, and if
Fl::use_high_res_GL(bool) is set to true, pixel_h() returns 2 * h(). This method detects when the window has been
moved between low and high resolution displays and automatically adjusts the returned value.

Version
1.3.4

9.46.3.18 pixel_w()

int Fl_Gl_Window::pixel_w ( ) [inline]
Gives the window width in OpenGL pixels.
Generally identical with the result of the w() function, but for a window mapped to an Apple 'retina' display, and if
Fl::use_high_res_GL(bool) is set to true, pixel_w() returns 2 * w(). This method detects when the window has been
moved between low and high resolution displays and automatically adjusts the returned value.

Version
1.3.4

9.46.3.19 pixels_per_unit()

float Fl_Gl_Window::pixels_per_unit ( ) [inline]
The number of pixels per FLTK unit of length for the window.
Returns 1, except for a window mapped to an Apple 'retina' display, and if Fl::use_high_res_GL(bool) is set to true,
when it returns 2. This method dynamically adjusts its value when the window is moved to/from a retina display.
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

9.46.3.20 redraw_overlay()

void Fl_Gl_Window::redraw_overlay ( )
This method 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.

9.46.3.21 resize()

void Fl_Gl_Window::resize (  
   int X,  
   int Y,

Generated by Doxygen
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_Window.

9.46.3.22 show()

void Fl_Gl_Window::show ( ) [virtual]

Puts the window on the screen. Usually (on X) this has the side effect of opening the display.

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_Window.

9.46.3.23 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.

9.46.3.24 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
def 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:
9.47  **Fl_Glut_Bitmap_Font Struct Reference**

ftk glut font/size attributes used in the glutXXX functions
#include <glut.H>

Public Attributes
- **Fl_Font font**
- **Fl_Fontsize size**

9.47.1  Detailed Description

ftk glut font/size attributes used in the glutXXX functions
The documentation for this struct was generated from the following file:
- glut.H

9.48  **Fl_Glut_StrokeChar Struct Reference**

Public Attributes
- **int Number**
- **GLfloat Right**
- **const Fl_Glut_StrokeStrip * Strips**

The documentation for this struct was generated from the following file:
- glut.H

9.49  **Fl_Glut_StrokeFont Struct Reference**

Public Attributes
- **const Fl_Glut_StrokeChar ** Characters**
- **GLfloat Height**
- **char * Name**
- **int Quantity**

The documentation for this struct was generated from the following file:
- glut.H

9.50  **Fl_Glut_StrokeStrip Struct Reference**

Public Attributes
- **int Number**
- **const Fl_Glut_StrokeVertex * Vertices**

The documentation for this struct was generated from the following file:
- glut.H
9.51 Fl_Glut_StrokeVertex Struct Reference

Public Attributes

- GLfloat X
- GLfloat Y

The documentation for this struct was generated from the following file:

- glut.H

9.52 Fl_Glut_Window Class Reference

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 *)
  Creates a glut window, registers to the glut windows list.
- Fl_Glut_Window (int x, int y, int w, int h, const char *)
  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

- virtual Fl_Gl_Window * as_gl_window ()
  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.
- void * context () const
  Returns a pointer to the GLContext that this window is using.
- void context (void *, 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**()
  Forces the window to be drawn, this window is also made current and calls draw().

• void **hide**()
  Hides the window and destroys the OpenGL context.

• void **hide_overlay**()
  Hides the window if it is not this window, does nothing in WIN32.

• 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**()
  The make_overlay_current() method 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**()
  This method causes draw_overlay() to be called at a later time.

• void **resize** (int, int, int, int)
  Changes the size and position of the window.

• void **show**()
  Puts the window on the screen.

• void **show** (int a, char **b)

• void **swap_buffers**()
  The swap_buffers() method swaps the back and front 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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int border () const
  See void Fl_Window::border(int)
- 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.
- 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 ()
  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w ()
  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.
- Fl_Window (int w, int h, const char *title=0)
  Creates a window from the given size and title.
- Fl_Window (int x, int y, int w, int h, const char *title=0)
  Creates a window from the given position, size 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.
- 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_RGBImage *)
  Sets or resets a single window icon.
• void icon (const void *ic)
  Sets the current icon window target dependent data.
• 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_RGBImage *[], int)
  Sets the window icons.
• 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.
• 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.
• unsigned int override () const
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.
• 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|FL_OVERRIDE.
• void set_tooltip_window ()
  Marks the window as a tooltip window.
• void shape (const FlImage &b)
  Set the window’s shape with an FlImage.
• void shape (const FlImage *img)
  Assigns a non-rectangular shape to the window.
• void show (int argc, char **argv)
  Puts the window on the screen 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 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.
- virtual Fl_Group ∗ as_group ()
  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 &i, 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
  
  See void Fl_Group::resizable(Fl_Widget *box)
- void resizable (Fl_Widget &o)
  
  See void Fl_Group::resizable(Fl_Widget *box)
- void resizable (Fl_Widget *o)
  
  The resizable widget defines the resizing box for the group.
- 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.
- 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, 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)
  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.

• const Fl_Image *deimage () const

• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void deimage (Fl_Img &img)
  Sets the image to use as part of the widget label.
- void do_callback ()
  Calls the widget callback.
- void do_callback (Fl_Widget &o, long arg)
  Calls the widget callback.
- void do_callback (Fl_Widget &o, void *arg=0)
  Calls the widget callback.
- 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_Img * image ()
  Gets the image that is used as part of the widget label.
- const Fl_Img * image () const
- void image (Fl_Img &img)
  Sets the image to use as part of the widget label.
- void image (Fl_Img *img)
  Sets the image to use as part of the widget label.
- 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.
- 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.
• 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 (void ∗v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ()
  Draws the Fl_Gl_Window.

- void draw_overlay ()
  You must implement this virtual function if you want to draw into the overlay.

- int handle (int)
  Handle some FLTK events as needed.

Protected Member Functions inherited from Fl_Window

- 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.
Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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_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 *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 const char * default_xclass ()
  
  Returns the default xclass.

• static void default_xclass (const char *)
  
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

• static Fl_Group * current ()
  
  Returns the currently active group.

• static void current (Fl_Group *)
  
  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 *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_ACCEISTS_MENU
  = 1<<19 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

  flags possible values enumeration.
9.53 Fl_Graphics_Driver Class Reference

Protected Attributes inherited from Fl_Window

- shape_data_type * shape_data_
  non-null means the window has a non-rectangular shape

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_
  Stores the last window that was made current.

9.52.1 Detailed Description

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):

9.52.2 Member Function Documentation

9.52.2.1 draw()

```cpp
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 must override the draw() method.
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.
Reimplemented from Fl_Gl_Window.

9.52.2.2 draw_overlay()

```cpp
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.

9.52.2.3 handle()

```cpp
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

9.53 Fl_Graphics_Driver Class Reference

A virtual class subclassed for each graphics driver FLTK uses.
#include <Fl_Device.H>

Inheritance diagram for Fl_Graphics_Driver:

```
Fl_Graphics_Driver
    Fl_Device
    Fl_GDI_Graphics_Driver
    Fl_PostScript_Graphics_Driver
    Fl_Quartz_Graphics_Driver
    Fl_Xlib_Graphics_Driver
    Fl_GDI_Printer_Graphics_Driver
```

Classes

- struct **matrix**
  
  A 2D coordinate transformation matrix.

Public Member Functions

- virtual const char * **class_name** ()
  
  Returns the name of the class of this object.

- **Fl_Color** **color** ()
  
  see *fl_color*(void).

- virtual int **descent** ()
  
  see *fl_descent()*.

- virtual int **draw_scaled** (Fl_Image *img, int X, int Y, int W, int H)
  
  Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.

- **Fl_Font** **font** ()
  
  see *fl_font*(void).

- virtual void **font** (Fl_Font face, Fl_Fontsize fsize)
  
  see *fl_font*(Fl_Font face, Fl_Fontsize size).

- **Fl_Font_Descriptor** *font_descriptor** ()
  
  Returns a pointer to the current Fl_Font_Descriptor for the graphics driver.

- void **font_descriptor** (Fl_Font_Descriptor *d)
  
  Sets the current Fl_Font_Descriptor for the graphics driver.

- virtual int **height** ()
  
  see *fl_height*().

- **Fl_Fontsize** **size** ()
  
  see *fl_size*().

- virtual void **text_extents** (const char *str, int n, int &dx, int &dy, int &w, int &h)
  
  see *fl_text_extents*(const char *str, int n, int &dx, int &dy, int &w, int &h).

- virtual double **width** (const char *str, int n)
  
  see *fl_width*(const char *str, int n).

- virtual double **width** (unsigned int c)
  
  see *fl_width*(unsigned int c).

- virtual ~Fl_Graphics_Driver** ()
  
  The destructor.

Public Member Functions inherited from Fl_Device

- virtual ~Fl_Device** ()
  
  Virtual destructor.
Static Public Attributes

- static const char * class_id = "Fl_Graphics_Driver"

Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"
  
  A string that identifies each subclass of Fl_Device.

Protected Member Functions

- virtual void arc (double x, double y, double r, double start, double end)
  
  see fl_arc(double x, double y, double r, double start, double end).

- virtual void arc (int x, int y, int w, int h, double a1, double a2)
  
  see fl_arc(int x, int y, int w, int h, double a1, double a2).

- virtual void begin_complex_polygon ()
  
  see fl_begin_complex_polygon().

- virtual void begin_line ()
  
  see fl_begin_line().

- virtual void begin_loop ()
  
  see fl_begin_loop().

- virtual void begin_points ()
  
  see fl_begin_points().

- virtual void begin_polygon ()
  
  see fl_begin_polygon().

- virtual void circle (double x, double y, double r)
  
  see fl_circle(double x, double y, double r).

- virtual int clip_box (int x, int y, int w, int h, int &X, int &Y , int &W, int &H)
  
  see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).

- Fl_Region clip_region ()
  
  see fl_clip_region().

- void clip_region (Fl_Region r)
  
  see fl_clip_region(Fl_Region r).

- virtual void color (Fl_Color c)
  
  see fl_color(Fl_Color c).

- virtual void color (uchar r, uchar g, uchar b)
  
  see fl_color(uchar r, uchar g, uchar b).

- virtual void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  
  see fl_copy_offscreen().

- virtual void curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
  
  see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).

- virtual void draw (const char *str, int n, int x, int y)
  
  see fl_draw(const char *str, int n, int x, int y).

- virtual void draw (Fl_Bitmap *bm, int XP , int YP , int WP , int HP , int cx, int cy)
  
  Draws an Fl_Bitmap object to the device.

- virtual void draw (Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  
  Draws an Fl_Pixmap object to the device.

- virtual void draw (Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy)
  
  Draws an Fl_RGB_Image object to the device.

- virtual void draw (int angle, const char *str, int n, int x, int y)
  
  see fl_draw(int angle, const char *str, int n, int x, int y).

- 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` (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  see `fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D)`.

• virtual void `draw_image_mono` (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)
  see `fl_draw_image_mono(const uchar* buf, int X, int Y, int W, int H, int D, int L)`.

• virtual void `draw_image_mono` (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)
  see `fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D)`.

• virtual void `end_complex_polygon` ()
  see `fl_end_complex_polygon()`.

• virtual void `end_line` ()
  see `fl_end_line()`.

• virtual void `end_loop` ()
  see `fl_end_loop()`.

• virtual void `end_points` ()
  see `fl_end_points()`.

• virtual void `end_polygon` ()
  see `fl_end_polygon()`.

**Fl_Graphics_Driver**

The constructor.

• virtual void `gap` ()
  see `fl_gap()`.

• virtual void `line` (int x, int y, int x1, int y1)
  see `fl_line(int x, int y, int x1, int y1)`.

• virtual void `line` (int x, int y, int x1, int y1, int x2, int y2)
  see `fl_line(int x, int y, int x1, int y1, int x2, int y2)`.

• virtual void `line_style` (int style, int width=0, char *dashes=0)
  see `fl_line_style(int style, int width, char* dashes)`.

• virtual void `loop` (int x0, int y0, int x1, int y1, int x2, int y2)
  see `fl_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)
  see `fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`.

• void `mult_matrix` (double a, double b, double c, double d, double x, double y)
  see `fl_mult_matrix(double a, double b, double c, double d, double x, double y)`.

• virtual int `not_clipped` (int x, int y, int w, int h)
  see `fl_not_clipped(int x, int y, int w, int h)`.

• virtual void `pie` (int x, int y, int w, int h, double a1, double a2)
  see `fl_pie(int x, int y, int w, int h, double a1, double a2)`.

• virtual void `point` (int x, int y)
  see `fl_point(int x, int y)`.

• virtual void `polygon` (int x0, int y0, int x1, int y1, int x2, int y2)
  see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2)`.

• virtual void `polygon` (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`.

• virtual void `pop_clip` ()
  see `fl_pop_clip()`.

• void `pop_matrix` ()
  see `fl_pop_matrix()`.

• virtual void `push_clip` (int x, int y, int w, int h)
  see `fl_push_clip(int x, int y, int w, int h)`.

• void `push_matrix` ()
  see `fl_push_matrix()`.
• virtual void `push_no_clip()`
  
  *see Fl_push_no_clip().*

• virtual void `rect(int x, int y, int w, int h)`
  
  *see Fl_rect(int x, int y, int w, int h).*

• virtual void `rectf(int x, int y, int w, int h)`
  
  *see Fl_rectf(int x, int y, int w, int h).*

• void `restore_clip()`
  
  *see Fl_restore_clip().*

• void `rotate(double d)`
  
  *see Fl_rotate(double d).*

• virtual void `rtl_draw(const char *str, int n, int x, int y)`
  
  *see Fl_rtl_draw(const char *str, int n, int x, int y).*

• void `scale(double x)`
  
  *see Fl_scale(double x).*

• void `scale(double x, double y)`
  
  *see Fl_scale(double x, double y).*

• double `transform_dx(double x, double y)`
  
  *see Fl_transform_dx(double x, double y).*

• double `transform_dy(double x, double y)`
  
  *see Fl_transform_dy(double x, double y).*

• double `transform_x(double x, double y)`
  
  *see Fl_transform_x(double x, double y).*

• double `transform_y(double x, double y)`
  
  *see Fl_transform_y(double x, double y).*

• virtual void `transformed_vertex(double xf, double yf)`
  
  *see Fl_transformed_vertex(double xf, double yf).*

• void `translate(double x, double y)`
  
  *see Fl_translate(double x, double y).*

• virtual void `vertex(double x, double y)`
  
  *see Fl_vertex(double x, double y).*

• virtual void `xyline(int x, int y, int x1)`
  
  *see Fl_xyline(int x, int y, int x1).*

• virtual void `xyline(int x, int y, int x1, int y2)`
  
  *see Fl_xyline(int x, int y, int x1, int y2).*

• virtual void `xyline(int x, int y, int x1, int y2, int x3)`
  
  *see Fl_xyline(int x, int y, int x1, int y2, int x3).*

• virtual void `yxline(int x, int y, int y1)`
  
  *see Fl_yxline(int x, int y, int y1).*

• virtual void `yxline(int x, int y, int y1, int x2)`
  
  *see Fl_yxline(int x, int y, int y1, int x2).*

• virtual void `yxline(int x, int y, int y1, int x2, int y3)`
  
  *see Fl_yxline(int x, int y, int y1, int x2, int y3).*

**Protected Attributes**

• `matrix * fl_matrix`
  
  *Points to the current coordinate transformation matrix.*
Friends

- **void fl_arc** (double x, double y, double r, double start, double end)
  
  Adds 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** ()
  
  Starts drawing a complex filled polygon.

- **void fl_begin_line** ()
  
  Starts drawing a list of lines.

- **void fl_begin_loop** ()
  
  Starts drawing a closed sequence of lines.

- **void fl_begin_points** ()
  
  Starts drawing a list of points.

- **void fl_begin_polygon** ()
  
  Starts drawing a convex filled polygon.

- **class Fl_Bitmap**

- **void fl_circle** (double x, double y, double r)
  
  fl_circle() is equivalent to fl_arc(x,y,r,0,360), but may be faster.

- **int fl_clip_box** (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)
  
  Intersects the rectangle with the current clip region and returns the bounding box of the result.

- **Fl_Region fl_clip_region** ()
  
  Returns the current clipping region.

- **void fl_clip_region** (Fl_Region r)
  
  Replaces the top of the clipping stack with a clipping region of any shape.

- **void fl_color** (Fl_Color c)
  
  Sets the color for all subsequent drawing operations.

- **void fl_color** (uchar r, uchar g, uchar b)
  
  Sets the color for all subsequent drawing operations.

- **FL_EXPORT 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.

- **void fl_curve** (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
  
  Adds a series of points on a Bezier curve to the path.

- **void fl_end_complex_polygon** ()
  
  Ends complex filled polygon, and draws.

- **void fl_end_line** ()
  
  Ends list of lines, and draws.

- **void fl_end_loop** ()
  
  Ends closed sequence of lines, and draws.
• void **fl_end_points** ()
  
  Ends list of points, and draws.

• void **fl_end_polygon** ()
  
  Ends convex filled polygon, and draws.

• void **fl_font** (Fl_Font face, Fl_Fontsize size)
  
  Sets the current font, which is then used in various drawing routines.

• void **fl_gap** ()
  
  Call **fl_gap()** to separate loops of the path.

• void **fl_line** (int x, int y, int x1, int y1)
  
  Draws a line from (x,y) to (x1,y1)

• void **fl_line** (int x, int y, int x1, int y1, int x2, int y2)
  
  Draws a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)

• void **fl_line_style** (int style, int width, char ∗dashes)
  
  Sets how to draw lines (the "pen").

• void **fl_loop** (int x0, int y0, int x1, int y1, int x2, int y2)
  
  Outlines a 3-sided polygon with lines.

• void **fl_loop** (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  
  Outlines a 4-sided polygon with lines.

• void **fl_mult_matrix** (double a, double b, double c, double d, double x, double y)
  
  Concatenates 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?

• void **fl_pie** (int x, int y, int w, int h, double a1, double a2)
  
  Draw filled ellipse sections using integer coordinates.

• class Fl_Pixmap

• void **fl_point** (int x, int y)
  
  Draws a single pixel at the given coordinates.

• void **fl_polygon** (int x0, int y0, int x1, int y1, int x2, int y2)
  
  Fills a 3-sided polygon.

• void **fl_polygon** (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  
  Fills a 4-sided polygon.

• void **fl_pop_clip** ()
  
  Restores the previous clip region.

• void **fl_pop_matrix** ()
  
  Restores the current transformation matrix from the stack.

• void **fl_push_clip** (int x, int y, int w, int h)
  
  Intersects the current clip region with a rectangle and pushes this new region onto the stack.

• void **fl_push_matrix** ()
  
  Saves the current transformation matrix on the stack.

• void **fl_push_no_clip** ()
  
  Pushes an empty clip region onto the stack so nothing will be clipped.

• void **fl_rect** (int x, int y, int w, int h)
  
  Draws a 1-pixel border inside the given bounding box.

• void **fl_rectf** (int x, int y, int w, int h)
  
  Colors with current color a rectangle that exactly fills the given bounding box.

• void **fl_restore_clip** ()
  
  Undoes any clobbering of clip done by your program.

• class Fl_RGB_Image

• void **fl_rotate** (double d)
  
  Concatenates rotation transformation onto the current one.

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

- **void fl_scale (double x)**
  Concatenates scaling transformation onto the current one.

- **void fl_scale (double x, double y)**
  Concatenates scaling transformation onto the current one.

- **double fl_transform_dx (double x, double y)**
  Transforms distance using current transformation matrix.

- **double fl_transform_dy (double x, double y)**
  Transforms distance using current transformation matrix.

- **double fl_transform_x (double x, double y)**
  Transforms coordinate using the current transformation matrix.

- **double fl_transform_y (double x, double y)**
  Transforms coordinate using the current transformation matrix.

- **void fl_transformed_vertex (double xf, double yf)**
  Adds coordinate pair to the vertex list without further transformations.

- **void fl_translate (double x, double y)**
  Concatenates translation transformation onto the current one.

- **void fl_vertex (double x, double y)**
  Adds a single vertex to the current path.

- **void fl_xyline (int x, int y, int x1)**
  Draws a horizontal line from $(x,y)$ to $(x1,y)$

- **void fl_xyline (int x, int y, int x1, int y2)**
  Draws 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)**
  Draws 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)**
  Draws a vertical line from $(x,y)$ to $(x,y1)$

- **void fl_yxline (int x, int y, int y1, int x2)**
  Draws 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)**
  Draws 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 $(x2,y3)$

**FL_EXPORT void gl_start ()**

Creates an OpenGL context.

### 9.53.1 Detailed Description

A virtual class subclassed for each graphics driver FLTK uses. Typically, FLTK applications do not use directly objects from this class. Rather, they perform drawing operations (e.g., `fl_rectf()`) that operate on the current drawing surface (see `Fl_Surface_Device`). Drawing operations are functionally presented in Drawing Things in FLTK and as function lists in the Drawing functions and Color & Font functions modules. The `fl_graphics_driver` global variable gives at any time the graphics driver used by all drawing operations. Its value changes when drawing operations are directed to another drawing surface by `Fl_Surface_Device::set_current()`.

The `Fl_Graphics_Driver` class is of interest if one wants to perform new kinds of drawing operations. An example would be to draw to a PDF file. This would involve creating a new `Fl_Graphics_Driver` derived class. This new class should implement all virtual methods of the `Fl_Graphics_Driver` class to support all FLTK drawing functions.
9.53.2 Member Function Documentation

9.53.2.1 arc() [1/2]

```cpp
void Fl_Graphics_Driver::arc(
    double x,
    double y,
    double r,
    double start,
    double end) [protected], [virtual]
```

see fl_arc(double x, double y, double r, double start, double end).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.2 arc() [2/2]

```cpp
void Fl_Graphics_Driver::arc(
    int x,
    int y,
    int w,
    int h,
    double a1,
    double a2) [protected], [virtual]
```

see fl_arc(int x, int y, int w, int h, double a1, double a2).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.3 begin_complex_polygon()

```cpp
void Fl_Graphics_Driver::begin_complex_polygon() [protected], [virtual]
```

see fl_begin_complex_polygon().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.4 begin_line()

```cpp
void Fl_Graphics_Driver::begin_line() [protected], [virtual]
```

see fl_begin_line().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.5 begin_loop()

```cpp
void Fl_Graphics_Driver::begin_loop() [protected], [virtual]
```

see fl_begin_loop().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.6 begin_points()

```cpp
void Fl_Graphics_Driver::begin_points() [protected], [virtual]
```

see fl_begin_points().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.7 begin_polygon()

```cpp
void Fl_Graphics_Driver::begin_polygon() [protected], [virtual]
```

see fl_begin_polygon().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.8 circle()

```cpp
void Fl_Graphics_Driver::circle(
    double x,
```
double y,
    double r) [protected], [virtual]
see fl_circle(double x, double y, double r).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.9 class_name()

virtual const char * Fl_Graphics_Driver::class_name ( ) [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
    if ( instance->class_name() == Fl_Printer::class_id ) { ... }
Reimplemented from Fl_Device.

9.53.2.10 clip_box()

int Fl_Graphics_Driver::clip_box (  
    int x,
    int y,
    int w,
    int h,
    int & X,
    int & Y,
    int & W,
    int & H ) [protected], [virtual]
see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).
ReImplemented in Fl_PostScript_Graphics_Driver.

9.53.2.11 color() [1/2]

virtual void Fl_Graphics_Driver::color (  
    Fl_Color c ) [inline], [protected], [virtual]
see fl_color(Fl_Color c).
ReImplemented in Fl_Quartz_Graphics_Driver, Fl_GDI_Graphics_Driver, Fl_Xlib_Graphics_Driver, and
Fl_PostScript_Graphics_Driver.

9.53.2.12 color() [2/2]

virtual void Fl_Graphics_Driver::color (  
    uchar r,
    uchar g,
    uchar b ) [inline], [protected], [virtual]
see fl_color(uchar r, uchar g, uchar b).
ReImplemented in Fl_Quartz_Graphics_Driver, Fl_GDI_Graphics_Driver, Fl_Xlib_Graphics_Driver, and
Fl_PostScript_Graphics_Driver.

9.53.2.13 copy_offscreen()

void Fl_Graphics_Driver::copy_offscreen (  
    int x,
    int y,
    int w,
    int h,
    Fl_Offscreen pixmap,
    int srcx,
    int srcy ) [protected], [virtual]
see fl_copy_offscreen()
9.53.2.14 curve()

void Fl_Graphics_Driver::curve (  
    double X0,  
    double Y0,  
    double X1,  
    double Y1,  
    double X2,  
    double Y2,  
    double X3,  
    double Y3 )  [protected],  [virtual]

see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).  
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.15 descent()

virtual int Fl_Graphics_Driver::descent ( )  [inline],  [virtual]

see fl_descent().  

9.53.2.16 draw() [1/5]

virtual void Fl_Graphics_Driver::draw (  
    const char ∗ str,  
    int n,  
    int x,  
    int y )  [inline],  [protected],  [virtual]

see fl_draw(const char ∗ str, int n, int x, int y).  

9.53.2.17 draw() [2/5]

virtual void Fl_Graphics_Driver::draw (  
    Fl_Bitmap ∗ bm,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy )  [inline],  [protected],  [virtual]

Draws an Fl_Bitmap object to the device.  
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.  

9.53.2.18 draw() [3/5]

virtual void Fl_Graphics_Driver::draw (  
    Fl_Pixmap ∗ pixmap,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy )  [inline],  [protected],  [virtual]

Draws an Fl_Pixmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.


9.53.2.19 draw() [4/5]

virtual void Fl_Graphics_Driver::draw (Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy) [inline], [protected], [virtual]

Draws an Fl_RGB_Image object to the device.

Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.


9.53.2.20 draw() [5/5]

virtual void Fl_Graphics_Driver::draw (int angle, const char *str, int n, int x, int y) [inline], [protected], [virtual]

see fl_draw(int angle, const char *str, int n, int x, int y).


9.53.2.21 draw_image() [1/2]

virtual void Fl_Graphics_Driver::draw_image (const uchar *buf, int X, int Y, int W, int H, int D = 3, int L = 0) [inline], [protected], [virtual]

see fl_draw_image(const uchar * buf, int X,int Y,int W,int H, int D, int L).


9.53.2.22 draw_image() [2/2]

virtual void Fl_Graphics_Driver::draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D)

see fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X,int Y,int W,int H, int D).

9.53.2.23 draw_image_mono() [1/2]

virtual void Fl_Graphics_Driver::draw_image_mono
(const uchar * buf,
 int X,
 int Y,
 int W,
 int H,
 int D = 1,
 int L = 0) [inline], [protected], [virtual]

see fl_draw_image_mono(const uchar* buf, int X, int Y, int W, int H, int D, int L).

9.53.2.24 draw_image_mono() [2/2]

virtual void Fl_Graphics_Driver::draw_image_mono
(Fl_Draw_Image_Cb cb,
 void * data,
 int X,
 int Y,
 int W,
 int H,
 int D = 1) [inline], [protected], [virtual]

see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).

9.53.2.25 draw_scaled()

int Fl_Graphics_Driver::draw_scaled
(Fl_Image * img,
 int X,
 int Y,
 int W,
 int H) [virtual]

Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.

Returns

zero when the graphics driver doesn’t implement scaled drawing, non-zero if it does implement it.


9.53.2.26 end_complex_polygon()

void Fl_Graphics_Driver::end_complex_polygon () [protected], [virtual]

see fl_end_complex_polygon().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.27 end_line()

void Fl_Graphics_Driver::end_line () [protected], [virtual]

see fl_end_line().
Reimplemented in Fl_PostScript_Graphics_Driver.
9.53.2.28 end_loop()

void Fl_Graphics_Driver::end_loop ( ) [protected], [virtual]
see fl_end_loop().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.29 end_points()

void Fl_Graphics_Driver::end_points ( ) [protected], [virtual]
see fl_end_points().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.30 end_polygon()

void Fl_Graphics_Driver::end_polygon ( ) [protected], [virtual]
see fl_end_polygon().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.31 font()

virtual void Fl_Graphics_Driver::font (  
    Fl_Font face,
    Fl_Fontsize fsize ) [inline], [virtual]
see fl_font(Fl_Font face, Fl_Fontsize size).

9.53.2.32 gap()

void Fl_Graphics_Driver::gap ( ) [protected], [virtual]
see fl_gap().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.33 height()

virtual int Fl_Graphics_Driver::height ( ) [inline], [virtual]
see fl_height().

9.53.2.34 line() [1/2]

void Fl_Graphics_Driver::line (  
    int x,
    int y,
    int x1,
    int y1 ) [protected], [virtual]
see fl_line(int x, int y, int x1, int y1).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.35 line() [2/2]

void Fl_Graphics_Driver::line (  
    int x,
    int y,
    int x1,
    int y1,
    int x2,
    int y2 ) [protected], [virtual]
see fl_line(int x, int y, int x1, int y1, int x2, int y2).
Reimplemented in Fl_PostScript_Graphics_Driver.

### 9.53.2.36 line_style()

```cpp
decl void Fl_Graphics_Driver::line_style ( 
    int style, 
    int width = 0, 
    char * dashes = 0 ) [protected], [virtual] 
```

Reimplemented in Fl_PostScript_Graphics_Driver.

### 9.53.2.37 loop() [1/2]

```cpp
decl void Fl_Graphics_Driver::loop ( 
    int x0, 
    int y0, 
    int x1, 
    int y1, 
    int x2, 
    int y2 ) [protected], [virtual] 
```

see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2).
Reimplemented in Fl_PostScript_Graphics_Driver.

### 9.53.2.38 loop() [2/2]

```cpp
decl void Fl_Graphics_Driver::loop ( 
    int x0, 
    int y0, 
    int x1, 
    int y1, 
    int x2, 
    int y2, 
    int x3, 
    int y3 ) [protected], [virtual] 
```

see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).
Reimplemented in Fl_PostScript_Graphics_Driver.

### 9.53.2.39 not_clipped()

```cpp
decl int Fl_Graphics_Driver::not_clipped ( 
    int x, 
    int y, 
    int w, 
    int h ) [protected], [virtual] 
```

see fl_not_clipped(int x, int y, int w, int h).
Reimplemented in Fl_PostScript_Graphics_Driver.

### 9.53.2.40 pie()

```cpp
decl void Fl_Graphics_Driver::pie ( 
    int x, 
    int y, 
    int w, 
    int h, 
    double a1, 
    double a2 ) [protected], [virtual] 
```

see fl_pie(int x, int y, int w, int h, double a1, double a2).
Reimplemented in Fl_PostScript_Graphics_Driver.
9.53.2.41  point()

```cpp
void Fl_Graphics_Driver::point (
    int x,
    int y ) [protected], [virtual]
```

see fl_point(int x, int y).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.42  polygon() [1/2]

```cpp
void Fl_Graphics_Driver::polygon (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2 ) [protected], [virtual]
```

see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.43  polygon() [2/2]

```cpp
void Fl_Graphics_Driver::polygon (
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    int x3,
    int y3 ) [protected], [virtual]
```

see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.44  pop_clip()

```cpp
void Fl_Graphics_Driver::pop_clip ( ) [protected], [virtual]
```

see fl_pop_clip().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.45  push_clip()

```cpp
void Fl_Graphics_Driver::push_clip ( int x,
    int y,
    int w ) [protected], [virtual]
```

see fl_push_clip(int x, int y, int w).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.46  push_no_clip()

```cpp
void Fl_Graphics_Driver::push_no_clip ( ) [protected], [virtual]
```

see fl_push_no_clip().
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.47  rect()

```cpp
void Fl_Graphics_Driver::rect ( int x,
```

Generated by Doxygen
int y, int w, int h) [protected], [virtual]

see fl_rect(int x, int y, int w, int h).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.48 rectf()

void Fl_Graphics_Driver::rectf (int x, int y, int w, int h) [protected], [virtual]

see fl_rectf(int x, int y, int w, int h).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.49 rtl_draw()

virtual void Fl_Graphics_Driver::rtl_draw (const char *str, int n, int x, int y) [inline], [protected], [virtual]

see fl_rtl_draw(const char *str, int n, int x, int y).

9.53.2.50 text_extents()

void Fl_Graphics_Driver::text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h) [virtual]

see fl_text_extents(const char *, int n, int & dx, int & dy, int & w, int & h).

9.53.2.51 transformed_vertex()

void Fl_Graphics_Driver::transformed_vertex (double xf, double yf) [protected], [virtual]

see fl_transformed_vertex(double xf, double yf).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.52 vertex()

void Fl_Graphics_Driver::vertex (double x, double y) [protected], [virtual]

see fl_vertex(double x, double y).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.53 width() [1/2]

virtual double Fl_Graphics_Driver::width (
const char * str,
    int n) [inline], [virtual]
see fl_width(const char *str, int n).

9.53.2.54 width() [2/2]
virtual double Fl_Graphics_Driver::width (     
    unsigned int c) [inline], [virtual]
see fl_width(unsigned int).

9.53.2.55 xyline() [1/3]
void Fl_Graphics_Driver::xyline (     
    int x,
    int y,
    int x1) [protected], [virtual]
see fl_xyline(int x, int y, int x1).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.56 xyline() [2/3]
void Fl_Graphics_Driver::xyline (     
    int x,
    int y,
    int x1,
    int y2) [protected], [virtual]
see fl_xyline(int x, int y, int x1, int y2).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.57 xyline() [3/3]
void Fl_Graphics_Driver::xyline (     
    int x,
    int y,
    int x1,
    int y2,
    int x3) [protected], [virtual]
see fl_xyline(int x, int y, int x1, int y2, int x3).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.58 yxline() [1/3]
void Fl_Graphics_Driver::yxline (     
    int x,
    int y,
    int y1) [protected], [virtual]
see fl_yxline(int x, int y, int y1).
Reimplemented in Fl_PostScript_Graphics_Driver.

9.53.2.59 yxline() [2/3]
void Fl_Graphics_Driver::yxline (     
    int x,
    int y,
see \texttt{fl\_yxline(int x, int y, int y1, int x2)}.
Reimplemented in \texttt{Fl\_PostScript\_Graphics\_Driver}.

9.53.2.60 \texttt{yxline()} [3/3]

\begin{verbatim}
void Fl_Graphics_Driver::yxline (  
  int x,  
  int y,  
  int y1,  
  int x2,  
  int y3) [protected], [virtual]
\end{verbatim}
see \texttt{fl\_yxline(int x, int y, int y1, int x2, int y3)}.
Reimplemented in \texttt{Fl\_PostScript\_Graphics\_Driver}.

9.53.3 Friends And Related Symbol Documentation

9.53.3.1 \texttt{fl\_arc} [1/2]

\begin{verbatim}
void fl\_arc {
  double x,  
  double y,  
  double r,  
  double start,  
  double end) [friend]
\end{verbatim}

Adds 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 \texttt{fl\_arc()}.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{x,y,r}</td>
<td>center and radius of circular arc</td>
</tr>
<tr>
<td>\texttt{start,end}</td>
<td>angles of start and end of arc measured in degrees counter-clockwise from 3 o'clock. If \texttt{end} is less than \texttt{start} then it draws the arc in a clockwise direction.</td>
</tr>
</tbody>
</table>

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
fl\_begin\_line();
fl\_arc(200.0, 100.0, 50.0, 0.0, 180.0);
fl\_end\_line();

// Draw filled arc
fl\_begin\_polygon();
fl\_arc(300.0, 100.0, 50.0, 0.0, 180.0);
fl\_end\_polygon();

9.53.3.2 \texttt{fl\_arc} [2/2]

\begin{verbatim}
void fl\_arc {
  int x,  
  int y,  
  int w,  
  int h,  
  double s1,  
  double s2) [friend]
\end{verbatim}

Draw ellipse sections using integer coordinates.
These functions match the rather limited circle drawing code provided by X and WIN32. 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, $a_2$ must be greater or equal to $a_1$.

$\text{fl}_{\text{arc}}()$ draws a series of lines to approximate the arc. Notice that the integer version of $\text{fl}_{\text{arc}}()$ has a different number of arguments than the double version $\text{fl}_{\text{arc}}(\text{double } x, \text{ double } y, \text{ double } r, \text{ double } \text{start}, \text{ double } \text{end})$

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>$x,y,w,h$</th>
<th>bounding box of complete circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>$a_1,a_2$</td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. $a_2$ must be greater than or equal to $a_1$.</td>
</tr>
</tbody>
</table>

### 9.53.3.3 fl_begin_complex_polygon

void fl_begin_complex_polygon ( ) [friend]

Starts drawing a complex filled polygon.

The polygon may be concave, may have holes in it, or may be several disconnected pieces. Call $\text{fl}_{\text{gap}}()$ to separate loops of the path.

To outline the polygon, use $\text{fl}_{\text{begin_loop}}()$ and replace each $\text{fl}_{\text{gap}}()$ with $\text{fl}_{\text{end_loop}}();\text{fl}_{\text{begin_loop}}()$ 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.

### 9.53.3.4 fl_begin_points

void fl_begin_points ( ) [friend]

Starts drawing a list of points.

Points are added to the list with $\text{fl}_{\text{vertex}}()$

### 9.53.3.5 fl_circle

void fl_circle ( double $x$, double $y$, double $r$ ) [friend]

$\text{fl}_{\text{circle}}()$ is equivalent to $\text{fl}_{\text{arc}}(x,y,r,0,360)$, 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 $\text{fl}_{\text{arc}}()$

**Parameters**

| in | $x,y,r$ | center and radius of circle |

### 9.53.3.6 fl_clip_box

int fl_clip_box ( int $x$, int $y$, int $w$, int $h$, int & $X$, int & $Y$, int & $W$, int & $H$ ) [friend]

Intersects the rectangle with the current clip region and returns the bounding box of the result.
Returns non-zero if the resulting rectangle is different to 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.

### 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.

#### 9.53.3.7 fl_clip_region

```c
void fl_clip_region ( Fl_Region r ) [friend]
```

Replaces the top of the clipping stack with a clipping region of any shape. Fl_Region is an operating system specific type.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>r</th>
<th>clipping region</th>
</tr>
</thead>
</table>

#### 9.53.3.8 fl_color [1/2]

```c
void fl_color ( Fl_Color c ) [friend]
```

Sets the color for all subsequent drawing operations.

For colormapped 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

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>color</th>
</tr>
</thead>
</table>

#### 9.53.3.9 fl_color [2/2]

```c
void fl_color ( uchar r, uchar g, uchar b ) [friend]
```

Sets 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

<table>
<thead>
<tr>
<th>in</th>
<th>r,g,b</th>
<th>color components</th>
</tr>
</thead>
</table>

#### 9.53.3.10 fl_copy_offscreen

```c
FL_EXPORT void fl_copy_offscreen (```

---

 Generated by Doxygen
int x,
int y,
int w,
int h,
F1_Offscreen pixmap,
int srcx,
int srcy ) [friend]

Copy a rectangular area of the given offscreen buffer into the current drawing destination.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y</td>
<td>position where to draw the copied rectangle</td>
</tr>
<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>

9.53.3.11 fl_curve

void fl_curve (  
    double X0,
    double Y0,
    double X1,
    double Y1,
    double X2,
    double Y2,
    double X3,
    double Y3 ) [friend]

Adds a series of points on a Bezier curve to the path. The curve ends (and two of the points) are at X0,Y0 and X3,Y3.

Parameters

<table>
<thead>
<tr>
<th>Parameter</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>

9.53.3.12 fl_draw

void fl_draw (  
    int angle,
    const char * str,
    int n,
    int x,
    int y ) [friend]

Draws 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.

9.53.3.13 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 ) [friend]

Draws 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:

```cpp
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.

9.53.3.14 fl_draw_image [2/2]

```cpp
void fl_draw_image ( 
  Fl_Draw_Image_Cb cb,
  void * data,
  int X,
  int Y,
  int W,
  int H,
  int D = 3 ) [friend]
```

Draws 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 easily.

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>
<tr>
<td>in</td>
<td>X, Y</td>
<td>screen position of top left pixel</td>
</tr>
<tr>
<td>in</td>
<td>W, H</td>
<td>image width and height</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>data size in bytes (must be greater than 0)</td>
</tr>
</tbody>
</table>
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.

### 9.53.3.15 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 ) [friend]
```

Draws 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)
```

### 9.53.3.16 fl_draw_image_mono [2/2]

```c
FLEXPORT void fl_draw_image_mono ( 
    Fl_Draw_Image_Cb cb,
    void * data,
    int X,
    int Y,
    int W,
    int H,
    int D = 1 ) [friend]
```

Draws 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)
```

### 9.53.3.17 fl_font

```c
void fl_font ( 
    Fl_Font face,
    Fl_Fontsize size ) [friend]
```

Sets the current font, which is then used in various drawing routines.

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 pixels and not "points". Lines should be spaced size pixels apart or more.

### 9.53.3.18 fl_gap

```c
void fl_gap ( ) [friend]
```

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 vertex, or several times in a row.
9.53.3.19 fl_line_style

void fl_line_style (  
    int style,  
    int width = 0,  
    char * dashes = 0 )  [friend]

Sets 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

<table>
<thead>
<tr>
<th>in</th>
<th>style</th>
<th>A bitmask which is a bitwise-OR of a line style, a cap style, and a join style. 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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>width</td>
<td>The thickness of the lines in pixels. Zero results in the system defined default, which on both X and Windows is somewhat different and nicer than 1.</td>
</tr>
<tr>
<td>in</td>
<td>dashes</td>
<td>A pointer to an array of dash lengths, measured in pixels. 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.</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 Windows 95, 98 or Me, since those operating systems do not support complex line styles.

9.53.3.20 fl_mult_matrix

void fl_mult_matrix (  
    double a,  
    double b,  
    double c,  
    double d,  
    double x,  
    double y )  [friend]

Concatenates another transformation onto the current one.

Parameters

| in | a,b,c,d,x,y | transformation matrix elements such that \( X' = aX + cY + x \) and \( Y' = bX + dY \) |

9.53.3.21 fl_not_clipped

int fl_not_clipped (  
    int x,  
    int y,  
    int w,  
    int h )  [friend]

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.

Note

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

### 9.53.3.22 fl_pie

```c
void fl_pie ( int x, int y, int w, int h, double a1, double a2 ) [friend]
```

Draw filled ellipse sections using integer coordinates.
Like `fl_arc()`, but `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`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>x,y,w,h</code></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. `a2` must be greater than or equal to `a1`.

### 9.53.3.23 fl_polygon [1/2]

```c
void fl_polygon ( int x0, int y0, int x1, int y1, int x2, int y2 ) [friend]
```

Fills a 3-sided polygon.
The polygon must be convex.

### 9.53.3.24 fl_polygon [2/2]

```c
void fl_polygon ( int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3 ) [friend]
```

Fills a 4-sided polygon.
The polygon must be convex.

### 9.53.3.25 fl_pop_clip

```c
void fl_pop_clip ( ) [friend]
```

Restores 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.

### 9.53.3.26 fl_push_clip

```c
void fl_push_clip (  
    int x,  
    int y,  
    int w,  
    int h )  [friend]
```

Intersects the current clip region with a rectangle and pushes 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>

### 9.53.3.27 fl_push_matrix

```c
void fl_push_matrix ( )  [friend]
```

Saves the current transformation matrix on the stack.

The maximum depth of the stack is 32.

### 9.53.3.28 fl_rect

```c
void fl_rect (  
    int x,  
    int y,  
    int w,  
    int h )  [friend]
```

Draws a 1-pixel border 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.

### 9.53.3.29 fl_rotate

```c
void fl_rotate (  
    double d )  [friend]
```

Concatenates rotation transformation onto the current one.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>d</th>
<th>- rotation angle, counter-clockwise in degrees (not radians)</th>
</tr>
</thead>
</table>

### 9.53.3.30 fl_scale [1/2]

```c
void fl_scale (  
    double x )  [friend]
```

Concatenates scaling transformation onto the current one.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>scale factor in both x-direction and y-direction</th>
</tr>
</thead>
</table>

### 9.53.3.31 fl_scale [2/2]

```c
void fl_scale (  
```

Generated by Doxygen
double x, double y) [friend]

Concatenates scaling transformation onto the current one.

Parameters

- **in** x, y: scale factors in x-direction and y-direction

### 9.53.3.32 fl_transform_dx

double fl_transform_dx (double x, double y) [friend]

Transforms distance using current transformation matrix.

Parameters

- **in** x, y: coordinate

### 9.53.3.33 fl_transform_dy

double fl_transform_dy (double x, double y) [friend]

Transforms distance using current transformation matrix.

Parameters

- **in** x, y: coordinate

### 9.53.3.34 fl_transform_x

double fl_transform_x (double x, double y) [friend]

Transforms coordinate using the current transformation matrix.

Parameters

- **in** x, y: coordinate

### 9.53.3.35 fl_transform_y

double fl_transform_y (double x, double y) [friend]

Transforms coordinate using the current transformation matrix.

Parameters

- **in** x, y: coordinate
9.53.36  fl_transformed_vertex

void fl_transformed_vertex (  
    double xf,  
    double yf ) [friend]

Adds coordinate pair to the vertex list without further transformations.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>xf, yf</strong></td>
<td>transformed coordinate</td>
</tr>
</tbody>
</table>

9.53.37  fl_translate

void fl_translate (  
    double x,  
    double y ) [friend]

Concatenates translation transformation onto the current one.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>x, y</strong></td>
<td>translation factor in x-direction and y-direction</td>
</tr>
</tbody>
</table>

9.53.38  fl_vertex

void fl_vertex (  
    double x,  
    double y ) [friend]

Adds a single vertex to the current path.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>x, y</strong></td>
<td>coordinate</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Device.H
- fl_arc.cxx
- fl_arci.cxx
- fl_curve.cxx
- Fl_Device.cxx
- Fl_Double_Window.cxx
- Fl_Image.cxx
- fl_line_style.cxx
- fl_rect.cxx
- fl_vertex.cxx

9.54  Fl_Group Class Reference

The Fl_Group class is the FLTK container widget.

```c
#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.
- `virtual FL_Group * as_group ()`
  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)`
  Generates by Doxygen
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 *)**
  
  This is for forms compatibility only.

- **void handle (int)**
  
  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**
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget &o)**
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget *) o**
  
  The resizable widget defines the resizing box for the group.

- **void resize (int, int, int, int)**
  
  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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• 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, 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)
  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.

- **const Fl_IMAGE ∗ deimage () const**
  
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image ∗ img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗ o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗ o, void ∗ arg=0)**
  
  Calls the widget callback.

- **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.

- **const Fl_IMAGE ∗ image () const**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_IMAGE &img)**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_IMAGE ∗ img)**
  
  Sets the image to use as part of the widget label.

- **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.
• **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.
• **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 (void *v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  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 () 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 ∗cb, void ∗d)
  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

- void draw ()
  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 () 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 ()
  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_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,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

9.54.1 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.

9.54.2 Constructor & Destructor Documentation

9.54.2.1 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.

9.54.2.2 ~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 child widgets’ destructors would be called twice!
9.54.3 Member Function Documentation

9.54.3.1 array()  

```c++
Fl_Widget *const * Fl_Group::array ( ) const
```

Returns a pointer to the array of children.  
This pointer is only valid until the next time a child is added or removed.

9.54.3.2 as_group()  

```c++
virtual 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:
```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 from Fl_Widget.

9.54.3.3 begin()  

```c++
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!

9.54.3.4 child()  

```c++
Fl_Widget * Fl_Group::child (  
    int n ) const [inline]
```

Returns array()[n].  
No range checking is done!

9.54.3.5 clear()  

```c++
void Fl_Group::clear ( )
```

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.

9.54.3.6 clip_children() [1/2]  

```c++
unsigned int Fl_Group::clip_children ( ) [inline]
```

Returns the current clipping mode.
9.54 Fl_Group Class Reference

Returns

true, if clipping is enabled, false otherwise.

See also
void Fl_Group::clip_children(int c)

9.54.3.7 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.

9.54.3.8 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).

9.54.3.9 current() [2/2]

void Fl_Group::current (  
    Fl_Group * g  ) [static]

Sets the current group.

See also
Fl_Group::current()

9.54.3.10 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.
Reimplemented in Fl_Help_View, Fl_Pack, Fl_Scroll, Fl_Tabs, Fl_Text_Display, Fl_Tree, Fl_Window, and Fl_Table.

9.54.3.11 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.

9.54.3.12 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.
9.54.3.13  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.

9.54.3.14  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.

9.54.3.15  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();

9.54.3.16  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.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</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>

Return values

See also  
    Fl_Event
Reimplemented from Fl_Widget.
Reimplemented in Fl_Table, 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.

9.54.3.17  init_sizes()
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 \texttt{resize()} occurs.

See also

\texttt{sizes()}

\subsection{insert} \label{fl__group_8h_1a6}

\begin{verbatim}
void Fl_Group::insert (  
    Fl_Widget & o,  
    int index )
\end{verbatim}

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 \(n > children().\) This can also be used to rearrange the widgets inside a group.

\subsection{insert} \label{fl__group_8h_1a10}

\begin{verbatim}
void Fl_Group::insert (  
    Fl_Widget & o,  
    Fl_Widget * before ) [inline]
\end{verbatim}

This does \texttt{insert(w, find(before))}. This will append the widget if \texttt{before} is not in the group.

\subsection{remove} \label{fl__group_8h_1a22}

\begin{verbatim}
void Fl_Group::remove (  
    Fl_Widget & o )
\end{verbatim}

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 \texttt{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 \texttt{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.

\subsection{remove} \label{fl__group_8h_1a26}

\begin{verbatim}
void Fl_Group::remove (  
    Fl_Widget * o ) [inline]
\end{verbatim}

Removes the widget \texttt{o} from the group.

\subsection{remove} \label{fl__group_8h_1a30}

\begin{verbatim}
void Fl_Group::remove (  
    int index )
\end{verbatim}

Removes the widget at \texttt{index} from the group but does not delete it. This method does nothing if \texttt{index} is out of bounds. This method differs from the \texttt{clear()} method in that it only affects a single widget and does not delete it from memory.

Since

\texttt{FLTK 1.3.0}
9.54.3.23 resizable()

void Fl_Group::resizable ( 
    Fl_Widget * o ) [inline]

The resizable widget defines the resizing box for the group. 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.

In these examples the gray area is the resizable:

![Figure 9.14 before resize](image)

![Figure 9.15 after resize](image)

The resizable may be set to the group itself, in which case all the contents are resized. This is the default value for Fl_Group, although NULL is the default for Fl_Window and Fl_Pack.

If the resizable is NULL then all widgets remain a fixed size and distance from the top-left corner.

It is possible to achieve any type of resize behavior by using an invisible Fl_Box as the resizable and/or by using a hierarchy of child Fl_Group's.

9.54.3.24 resize()

void Fl_Group::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*)
9.54.3.25 sizes()

int * Fl_Group::sizes () [protected]

Returns the internal array of widget sizes and positions.
If the sizes() array does not exist, it will be allocated and filled with the current widget sizes and positions.

Note
You should never need to use this method directly, unless you have special needs to rearrange the children of a Fl_Group. Fl_Tile uses this to rearrange its widget positions.

See also
init_sizes()

Todo Should the internal representation of the sizes() array be documented?

9.54.3.26 update_child()

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_compatability.cxx

9.55 Fl_GTK_File_Chooser Class Reference

Inheritance diagram for Fl_GTK_File_Chooser:

```
Fl_FLTK_File_Chooser
  `- Fl_GTK_File_Chooser
```

Friends

- class Fl_Native_File_Chooser
Additional Inherited Members

Protected Member Functions inherited from Fl_FLTK_File_Chooser

- const char* directory() const
- void directory(const char* val)
- const char* errmsg() const
- void errmsg(const char* msg)
- int exist_dialog()
- const char* filter() const
- void filter(const char*)
- int filter_value()
- void filter_value(int i)
- int filters()
- Fl_FLTK_File_Chooser(int val)
- int options() const
- void options(int)
- void parse_filter()
- const char* preset_file() const
- void preset_file(const char*)
- int type()
- int type_fl_file(int)

Protected Attributes inherited from Fl_FLTK_File_Chooser

- int _btype
- char* _directory
- char* _errmsg
- Fl_File_Chooser* _file_chooser
- char* _filter
- int _filtvalue
- int _nfilters
- int _options
- char* _parsedfilt
- char* _preset_file
- char* _prevvalue

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser.H
- Fl_Native_File_Chooser_GTK.cxx

9.56  Fl_Help_Block Struct Reference

Public Attributes

- Fl_Color bgcolor
- uchar border
- const char* end
- int h
- int line[32]
- const char* start
- int w
- int x
- int y

The documentation for this struct was generated from the following file:

- Fl_Help_View.H
9.57 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.

- **void 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.
9.57.1 Detailed Description

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.

Figure 9.16 Fl_Help_Dialog

9.57.2 Member Function Documentation

9.57.2.1 load()

```cpp
void 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").

9.57.2.2 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.

9.57.2.3 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

9.57.2.4 value() [1/2]

```cpp
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.

9.57.2.5 value() [2/2]

```cpp
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.hxx
9.58 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)`  
  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_[100]`  
  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

9.59 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)`  
  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.

9.59.1 Detailed Description

Fl_Help_View font stack element definition.
The documentation for this struct was generated from the following file:

- Fl_Help_View.H
9.60  **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.

9.60.1  **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

9.61  **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.

9.61.1  **Detailed Description**

Fl_Help_Target structure.
The documentation for this struct was generated from the following file:

• Fl_Help_View.H

9.62  **Fl_Help_View Class Reference**

The Fl_Help_View widget displays HTML text.
#include <Fl_Help_View.H>

Inheritance diagram for Fl_Help_View:
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)
  
  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)
  
  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)

- 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_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.

• virtual Fl_Group * **as_group** ()
  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 &i, 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
  See void Fl_Group::resizable(Fl_Widget ∗box)
- void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget ∗box)
- void resizable (Fl_Widget ∗o)
  The resizable widget defines the resizing box for the group.
- 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 Fl_Window ∗as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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)**
  
  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 ()**
  
  Generated by Doxygen
Gets the image that is used as part of the widget label.

* const Fl_Image * deimage () const
  Sets the image to use as part of the widget label.
  
  void deimage (Fl_Image & img)

Sets the image to use as part of the widget label.

* void deimage (Fl_Image * img)

Sets the image to use as part of the widget label.

* void do_callback ()
  Calls the widget callback.

* void do_callback (Fl_Widget * wgt, long arg)
  Calls the widget callback.

* void do_callback (Fl_Widget * wgt, void * arg=0)
  Calls the widget callback.

* 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.

* const Fl_Image * image () const
  Sets the image to use as part of the widget label.
  
  void image (Fl_Image & img)

Sets the image to use as part of the widget label.

* void image (Fl_Image * img)

Sets the image to use as part of the widget label.

* 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.

• **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.

• **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 (void *v)
  Sets the user data for this widget.
- unsigned int visible () const
  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 () 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 ()
  Draws the Fl_Help_View widget.

### Protected Member Functions inherited from Fl_Group

- 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()**
  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_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 *cb, void *d)**
  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.
9.62 Fl_Help_View Class Reference

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,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

9.62.1 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**
- **STRONG**
- **TABLE**: TH/TD/TR/BORDER/BGCOLOR/COLSPAN/ALIGN=CENTER|RIGHT|LEFT
9.62.2 Constructor & Destructor Documentation

9.62.2.1 ∼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.
9.62.3 Member Function Documentation

9.62.3.1 draw()
void Fl_Help_View::draw {
  void ) [protected], [virtual]
Draws the Fl_Help_View widget.
Reimplemented from Fl_Group.

9.62.3.2 find()
int Fl_Help_View::find {
  const char * s,
  int p = 0 )
Finds the specified string s at starting position p.
Returns
  the matching position or -1 if not found

9.62.3.3 handle()
int Fl_Help_View::handle ( {
  int event ) [virtual]
Handles events in the widget.
Reimplemented from Fl_Group.

9.62.3.4 leftline()
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) |

9.62.3.5 link()
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:
const char *fn(Fl_Widget *w, const char *url);
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.

9.62.3.6 load()
int Fl_Help_View::load ( {
  const char * f )
Loads the specified file.
This method loads the specified file or URL.
9.62.3.7 resize()

```cpp
void Fl_Help_View::resize (  
    int xx,  
    int yy,  
    int ww,  
    int hh } [virtual]
```

Resizes the help widget.
Reimplemented from Fl_Group.

9.62.3.8 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)

9.62.3.9 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

<table>
<thead>
<tr>
<th>in</th>
<th>newSize</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

See also
Fl::scrollbar_size()

9.62.3.10 topline() [1/2]

```cpp
void Fl_Help_View::topline (  
    const char * n )
```

Scrolls the text to the indicated position, given a named destination.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>target name</td>
</tr>
</tbody>
</table>
9.62.3.11 topline() [2/2]

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) |

9.62.3.12 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

9.63 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.

```c
#include <Fl_Hold_Browser.H>
```

Inheritance diagram for Fl_Hold_Browser:

```
Fl_Widget  
  |  
  v  
Fl_Group  
  |  
Fl_Browser_  
  |  
Fl_Browser  
  |  
Fl_Hold_Browser  
```

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 ()**
  
  *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)**

---

*Generated by Doxygen*
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 ()
  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)
  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 position () const
  Gets the vertical scroll position of the list as a pixel position pos.
• void position (int pos)
  Sets the vertical scroll position of the list to pixel position pos.
• void resize (int X, int Y, int W, int H)
  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
  This method has been deprecated, existing for backwards compatibility only.
• void scrollbar_width (int width)
  This method has been deprecated, existing for backwards compatibility only.
• 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.

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.
• virtual Fl_Group ∗ as_group ()
  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 *)

- **void forms_end** ()
  - This is for forms compatibility only.

- **int handle** (int)
  - 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
  - See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable** (Fl_Widget &)
  - See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable** (Fl_Widget *)
  - The resizable widget defines the resizing box for the group.

- **void resize** (int, int, int, int)
  - 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 Fl_Window ∗ as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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)
  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.
- const Fl_Image *deimage () const
- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void do_callback ()
  Calls the widget callback.
- void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
- void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
- 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.
- const Fl_Image *image () const
- void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
- 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.
• 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 ()
  Generated by Doxygen
Enables keyboard focus navigation with this widget.

- 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 (void ∗v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 *cb, void *d)
  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, 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
  The height of the entire list of all visible() items in pixels.
- int incr_height () const
  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
  Return the item at specified line.
- void item_draw (void ∗ item, int X, int Y, int W, int H) const
  Draws item at the position specified by XYWH.
- void ∗ item_first () const
  Returns the very first item in the list.
- int item_height (void ∗ item) const
  Returns height of item in pixels.
- void ∗ item_last () const
  Returns the very last item in the list.
- void ∗ item_next (void ∗ item) const
  Returns the next item after item.
- void ∗ item_prev (void ∗ item) const
  Returns the previous item before item.
- void item_select (void ∗ item, int val)
  Change the selection state of item to the value val.
- int item_selected (void ∗ item) const
  See if item is selected.
- void item_swap (void ∗a, void ∗b)
  Swap the items a and b.
- const char ∗ item_text (void ∗ item) const
  Returns the label text for item.
- int item_width (void ∗ item) const
  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 ()
  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**

- void draw ()
  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_Boxtyle 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_Boxtyle 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** ()
  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_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.

9.63.1 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.

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.

9.63.2 Constructor & Destructor Documentation

9.63.2.1 Fl_Hold_Browser()

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

Generated by Doxygen
## 9.64 Fl_Hor_Fill_Slider Class Reference

Inheritance diagram for Fl_Hor_Fill_Slider:

```
Fl_Widget
   ↓
Fl_Valuator
   ↓
Fl_Slider
   ↓
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)
  
  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)

- 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 Fl_Group ∗ as_group ()
  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.

- 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, 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)
  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.

- **const Fl_Image** * deimage () const
- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const Fl_Image** * image () const
- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **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 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.

• 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.

• 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
9.64 Fl_Hor_Fill_Slider Class Reference

Gets the widget type.

• void type (uchar t)

       Sets the widget type.

• int use_accent_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 (void *v)

       Sets the user data for this widget.

• unsigned int visible () const

       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 () 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 *cb, void *d)

       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,
    }

Generated by Doxygen
GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19
USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

• void draw ()
  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 ()
  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_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

### 9.65 Fl_Hor_Nice_Slider Class Reference

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)
  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 Fl_Group * as_group ()
  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.
- 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, 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)
  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.

  - **const FL_Image * deimage ()** const
  
  - **void deimage (FL_Image &img)**
    
    Sets the image to use as part of the widget label.
  
  - **void deimage (FL_Image *img)**
    
    Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

  - **const FL_Image * image ()** const
  
  - **void image (FL_Image &img)**
    
    
    Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

- `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 (void * v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  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 () 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 ∗cb, void ∗d)
  
  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 '38x' 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 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

• void draw ()
  
  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` ()
  
  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_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_Nice_Slider.H
• Fl_Slider.cxx
9.66  Fl_Hor_Slider Class Reference

Horizontal Slider class.

#include <Fl_Hor_Slider.H>

Inheritance diagram for Fl_Hor_Slider:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Valuator</td>
</tr>
<tr>
<td>Fl_Slider</td>
</tr>
<tr>
<td>Fl_Hor_Slider</td>
</tr>
</tbody>
</table>
```

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)
  
  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.

void 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 Fl_Group ∗ as_group ()
  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.
- `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, void)`
  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.
• const Fl_Image ∗ deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image ∗ img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget ∗ o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget ∗ o, void ∗ arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image ∗ image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image ∗ img)
  Sets the image to use as part of the widget label.
• 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.
• `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.
• `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 (void *v)**
  - Sets the user data for this widget.
- **unsigned int visible () const**
  - 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 () 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 *cb, void *d)**
  - 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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw ()
  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 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()**
  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_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.

### 9.66.1 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

### 9.67 Fl_Hor_Value_Slider Class Reference

Inheritance diagram for Fl_Hor_Value_Slider:
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)**
  
  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.

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 *f)**
  
  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 Fl_Group ∗ as_group ()
  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.
• **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_p cb)**
  
  Sets the current callback function for the widget.

• **void callback (Fl_Callback_p cb, void_p 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)**
  
  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_p w) const**
  
  Checks if w is a child of this widget.

• **void copy_label (const char_p new_label)**
  
  Sets the current label.

• **void copy_tooltip (const char_p new_tooltip)**
  
  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)**

Generated by Doxygen
Internal use only.

- **void deactivate ()**
  Deactivates the widget.

- **Fl_Image ∗ deimage ()**
  Gets the image that is used as part of the widget label.
  ```cpp
  void deimage (Fl_Image &img)
  ```
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image ∗img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

  ```cpp
  void do_callback (Fl_Widget ∗o, long arg)
  ```
  Calls the widget callback.

  ```cpp
  void do_callback (Fl_Widget ∗o, void ∗arg=0)
  ```
  Calls the widget callback.

- **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.
  ```cpp
  const Fl_Image ∗ image () const
  ```
  Gets the image to use as part of the widget label.

  ```cpp
  void image (Fl_Image &img)
  ```
  Sets the image to use as part of the widget label.

  ```cpp
  void image (Fl_Image ∗img)
  ```
  Sets the image to use as part of the widget label.

- **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.

  ```cpp
  void label (const char ∗text)
  ```
  Sets the current label pointer.

  ```cpp
  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.

  ```cpp
  void labelcolor (Fl_Color c)
  ```
  Sets the label color.

- **Fl_Font labelfont () const**
  Gets the font to use.

  ```cpp
  void labelfont (Fl_Font f)
  ```
  Sets the font to use.

- **Fl_Fontsize labelsize () const**
  Gets the font size in pixels.

  ```cpp
  void labelsize (Fl_Fontsize pix)
  ```
  Sets the font size in pixels.
- \texttt{Fl\_Labeltype\ labeltype()}\ const
  
  Gets the label type.
- \texttt{void\ labeltype(Fl\_Labeltype\ a)}
  
  Sets the label type.
- \texttt{void\ measure\ label(int &ww, int &hh)\ const}
  
  Sets width \(ww\) and height \(hh\) accordingly with the label size.
- \texttt{unsigned\ int\ output()}\ const
  
  Returns if a widget is used for output only.
- \texttt{Fl\_Group *\ parent()}\ const
  
  Returns a pointer to the parent widget.
- \texttt{void\ parent(Fl\_Group *p)}
  
  Internal use only - "for hacks only".
- \texttt{void\ position(int X, int Y)}
  
  Repositions the window or widget.
- \texttt{void\ redraw()}
  
  Schedules the drawing of the widget.
- \texttt{void\ redraw\_label()}
  
  Schedules the drawing of the label.
- \texttt{virtual\ void\ resize(int x, int y, int w, int h)}
  
  Changes the size or position of the widget.
- \texttt{Fl\_Color\ selection\_color()}\ const
  
  Gets the selection color.
- \texttt{void\ selection\_color(Fl\_Color\ a)}
  
  Sets the selection color.
- \texttt{void\ set\_active()}
  
  Marks the widget as active without sending events or changing focus.
- \texttt{void\ set\_changed()}
  
  Marks the value of the widget as changed.
- \texttt{void\ set\_output()}
  
  Sets a widget to output only.
- \texttt{void\ set\_visible()}
  
  Makes the widget visible.
- \texttt{void\ set\_visible\_focus()}
  
  Enables keyboard focus navigation with this widget.
- \texttt{virtual\ void\ show()}
  
  Makes a widget visible.
- \texttt{void\ size(int W, int H)}
  
  Changes the size of the widget.
- \texttt{int\ take\_focus()}
  
  Gives the widget the keyboard focus.
- \texttt{unsigned\ int\ takesevents()}\ const
  
  Returns if the widget is able to take events.
- \texttt{int\ test\_shortcut()}
  
  Returns true if the widget's label contains the entered '\&x' shortcut.
- \texttt{const\ char\ *\ tooltip()}\ const
  
  Gets the current tooltip text.
- \texttt{void\ tooltip(const\ char\ *text)}
  
  Sets the current tooltip text.
- \texttt{Fl\_Window\ *\ top\_window()\ const}
  
  Returns a pointer to the top-level window for the widget.
- \texttt{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 (void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  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 () 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 *cb, void *d)**
  
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Value_Slider

- void draw ()
  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** ()
  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_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_Value_Slider.H
- Fl_Value_Slider.cxx

9.68 **Fl_Image Class Reference**

Base class for image caching 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
```

**Public Member Functions**

• virtual void **color_average** (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the FLTK color value c.

• Fl_Image * **copy** ()
  The copy() method creates a copy of the specified image.

• virtual Fl_Image * **copy** (int W, int H)
  The copy() method creates a copy of the specified image.

• int **count** () const
The `count()` method returns the number of data values associated with the image.

- **int d () const**
  
  *Returns the current image depth.*

- **const char *const * data () const**
  
  *Returns a pointer to the current image data array.*

- **virtual void desaturate ()**
  
  *The desaturate() method converts an image to grayscale.*

- **void draw (int X, int Y)**
  
  *Draws the image.*

- **virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)**
  
  *Draws the image with a bounding box.*

- **int fail ()**
  
  *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 height in pixels.*

- **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)**
  
  *The label() methods are an obsolete way to set the image attribute of a widget or menu item.*

- **virtual void label (Fl_Widget *w)**
  
  *The label() methods are 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 uncache ()**
  
  *If the image has been cached for display, delete the cache data.*

- **int w () const**
  
  *Returns the current image width in pixels.*

- **virtual ~Fl_Image ()**
  
  *The destructor is a virtual method that frees all memory used by the image.*

### Static Public Member Functions

- **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 Public Attributes

- **static const int ERR_FILE_ACCESS = -2**
- **static const int ERR_FORMAT = -3**
- **static const int ERR_NO_IMAGE = -1**
Protected Member Functions

- `void d (int D)`
  Sets the current image depth.
- `void data (const char * const *p, int c)`
  Sets the current array 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.
- `void h (int H)`
  Sets the current image height in pixels.
- `void ld (int LD)`
  Sets the current line data size in bytes.
- `void w (int W)`
  Sets the current image width in pixels.

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)`

9.68.1 Detailed Description

Base class for image caching and drawing.

`Fl_Image` is the base class used for caching 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.

Since the `Fl_Image` class does not support image drawing by itself, calling the `draw()` method results in a box with an X in it being drawn instead.

9.68.2 Constructor & Destructor Documentation

9.68.2.1 `Fl_Image()`

`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.

9.68.3 Member Function Documentation

9.68.3.1 `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 FLTK color value `c`. The `i` 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 before changes are applied, to avoid modifying the original image. Reimplemented in `Fl_RGB_Image`, `Fl_Pixmap`, `Fl_Shared_Image`, and `Fl_Tiled_Image`. 
9.68.3.2 copy() [1/2]

Fl_Image * Fl_Image::copy () [inline]
The copy() method creates a copy of the specified image. If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.

9.68.3.3 copy() [2/2]

Fl_Image * Fl_Image::copy ( int W, int H ) [virtual]
The copy() method creates a copy of the specified image. If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it. Reimplemented in Fl_Bitmap, Fl_RGB_image, Fl_Pixmap, Fl_Shared_Image, and Fl_Tiled_Image.

9.68.3.4 count()

int Fl_Image::count () const [inline]
The count() method 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.

9.68.3.5 d()

int Fl_Image::d () const [inline]
Returns the current image depth. The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

9.68.3.6 data()

const char *const * Fl_Image::data () const [inline]
Returns a pointer to the current image data array. Use the count() method to find the size of the data array.

9.68.3.7 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 before changes are applied, to avoid modifying the original image. Reimplemented in Fl_RGB_image, Fl_Pixmap, Fl_Shared_Image, and Fl_Tiled_Image.

9.68.3.8 draw() [1/2]

void Fl_Image::draw ( int X, int Y ) [inline]
Draws the image. This form specifies the upper-lefthand corner of the image.

9.68.3.9 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 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_Shared_Image, Fl_Tiled_Image, Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

9.68.3.10 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.

9.68.3.11 fail()

int Fl_Image::fail (  

Returns a value that is not 0 if there is currently no image available.
Example use:
[..]  
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

ERR_NO_IMAGE if no image was found
ERR_FILE_ACCESS if there was a file access related error (errno should be set)
ERR_FORMAT if image decoding failed.

9.68.3.12 inactive()

void Fl_Image::inactive ( ) [inline]

The inactive() method calls 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.

9.68.3.13 label() [1/2]

void Fl_Image::label (  
    Fl_Menu_Item ∗ m ) [virtual]

The label() methods are an obsolete way to set the image attribute of a widget or menu item.
Use the image() or deimage() methods of the Fl_Widget and Fl_Menu_Item classes instead.
Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

9.68.3.14 label() [2/2]

void Fl_Image::label (  
    Fl_Widget ∗ widget ) [virtual]

The label() methods are an obsolete way to set the image attribute of a widget or menu item.
Use the image() or deimage() methods of the Fl_Widget and Fl_Menu_Item classes instead. Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

9.68.3.15  ld() [1/2]

int Fl_Image::ld ( ) const [inline]
Returns the current line data size in bytes.

See also
ld(int)

9.68.3.16  ld() [2/2]

void Fl_Image::ld (int LD ) [inline], [protected]
Sets the current line data size in bytes. Color images may contain extra data that is included after every line of color image data and is normally not present. If LD is zero, then line data size is assumed to be w() * d() bytes. If LD is non-zero, then it must be positive and larger than w() * d() to account for the extra data per line.

9.68.3.17  RGB_scaling()

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.

9.68.3.18  uncache()

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_Bitmap, Fl_RGB_Image, Fl_Pixmap, and Fl_Shared_Image.

The documentation for this class was generated from the following files:
• Fl_Image.H
• Fl_Image.cpp

9.69  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_Device
   Fl_Surface_Device
      Fl_Image_Surface

Public Member Functions

• const char * class_name ( )

   Returns the name of the class of this object.
• void draw (Fl_Widget *, int delta_x=0, int delta_y=0)
  Draws a widget in the image surface.
• void draw_decorated_window (Fl_Window *, int delta_x=0, int delta_y=0)
  Draws a window and its borders and title bar to the image drawing surface.
• Fl_Image_Surface (int w, int h, int highres=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 an image made of all drawings sent to the Fl_Image_Surface object.
• void set_current ()
  Make this surface the current drawing surface.
• ~Fl_Image_Surface ()
  The destructor.

Public Member Functions inherited from Fl_Surface_Device
• const char * class_name ()
  Returns the name of the class of this object.
• Fl_Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.
• void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.
• virtual ~Fl_Surface_Device ()
  The destructor.

Public Member Functions inherited from Fl_Device
• virtual ~Fl_Device ()
  Virtual destructor.

Static Public Attributes
• static const char * class_id = "Fl_Image_Surface"

Static Public Attributes inherited from Fl_Surface_Device
• static const char * class_id = "Fl_Surface_Device"

Static Public Attributes inherited from Fl_Device
• static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device
• static Fl_Surface_Device * surface ()
  The current drawing surface.

Protected Member Functions inherited from Fl_Surface_Device
• Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.
9.69 Fl_Image_Surface Class Reference

9.69.1 Detailed Description

Directs all graphics requests to an Fl_Image.
After creation of an Fl_Image_Surface object, call set_current() on it, 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
Fl_Widget *g = ...; // a widget you want to draw in an image
Fl_Image_Surface *img_surf = new Fl_Image_Surface(g->w(), g->h()); // create an Fl_Image_Surface object
img_surf->set_current(); // direct graphics requests to the image
fl_color(FL_WHITE); fl_rectf(0, 0, g->w(), g->h()); // draw a white background
img_surf->draw(g); // draw the g widget in the image
Fl_RGB_Image* image = img_surf->image(); // get the resulting image
delete img_surf; // delete the img_surf object
Fl_Display_Device::display_device()->set_current(); // direct graphics requests back to the display
```

9.69.2 Constructor & Destructor Documentation

9.69.2.1 Fl_Image_Surface()

```c
Fl_Image_Surface::Fl_Image_Surface ( 
int w, 
int h, 
int highres = 0 )
```

Constructor with optional high resolution.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>give the size in pixels of the resulting image.</td>
</tr>
<tr>
<td>h</td>
<td>if non-zero, the surface pixel size is twice as high and wide as w and h, which is useful to draw it later on a high resolution display (e.g., retina display). This is implemented for the Mac OS platform only. If highres is non-zero, use Fl_Image_Surface::highres_image() to get the image data.</td>
</tr>
<tr>
<td>highres</td>
<td></td>
</tr>
</tbody>
</table>

Version

1.3.4 and requires compilation with -DFL_ABI_VERSION=10304 (1.3.3 without the highres parameter)

9.69.3 Member Function Documentation

9.69.3.1 class_name()

```c
const char * Fl_Image_Surface::class_name ( ) [inline], [virtual]
```

Returns the name of the class of this object.

Use of the class_name() function is discouraged because it will be removed from future FLTK versions.

The class of an instance of an Fl_Device subclass can be checked with code such as:

```c
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```

Reimplemented from Fl_Device.

9.69.3.2 draw()

```c
void Fl_Image_Surface::draw ( 
    Fl_Widget * widget, 
    int delta_x = 0, 
    int delta_y = 0 )
```

Draws a widget in the image surface.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>any FLTK widget (e.g., standard, custom, window, GL view) to draw in the image</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \delta \rightarrow x )</td>
<td>and</td>
</tr>
<tr>
<td>( \delta \rightarrow y )</td>
<td>give the position in the image of the top-left corner of the widget</td>
</tr>
</tbody>
</table>

### 9.69.3.3 draw_decorated_window()

```cpp
def Fl_Draw_Surface::draw_decorated_window:
    Fl_Window * win,
    int delta_x = 0,
    int delta_y = 0
```

Draws a window and its borders and title bar to the image drawing surface.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{win} )</td>
<td>an FLTK window to draw in the image</td>
</tr>
<tr>
<td>( \delta \rightarrow x )</td>
<td>and</td>
</tr>
<tr>
<td>( \delta \rightarrow y )</td>
<td>give the position in the image of the top-left corner of the window's title bar</td>
</tr>
</tbody>
</table>

### 9.69.3.4 highres_image()

```
def Fl_Draw_Surface::highres_image:
```

Returns a possibly high resolution image made of all drawings sent to the Fl_Draw_Surface object. The Fl_Draw_Surface object should have been constructed with Fl_Draw_Surface(W, H, 1). The returned image is scaled to a size of WxH drawing units and may have a pixel size twice as wide and high. The returned object should be deallocated with Fl_Shared_Image::release() after use.

**Version**

1.3.4 and requires compilation with `-DFL_ABI_VERSION=10304`

### 9.69.3.5 image()

```
def Fl_Draw_Surface::image:
```

Returns an image made of all drawings sent to the Fl_Draw_Surface object. The returned object contains its own copy of the RGB data. Prefer Fl_Draw_Surface::highres_image() if the surface was constructed with the highres option on.

### 9.69.3.6 set_current()

```cpp
void Fl_Draw_Surface::set_current {
    void  } [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Reimplemented from Fl_Surface_Device.

The documentation for this class was generated from the following files:

- Fl_Draw_Surface.H
- Fl_Draw_Surface.cxx
9.70  **Fl_Input Class Reference**

This is the FLTK text input widget.

```cpp
#include <Fl_Input.H>
```

Inheritance diagram for Fl_Input:

```
Fl_Widget
  ↓
Fl_Input_
  ↓
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)**
  
  Handles the specified event.

### Public Member Functions inherited from Fl_Input_

- **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.

- **Fl_Input_ (int, int, int, int, const char ∗=0)**
  
  Creates a new **Fl_Input_** widget.

- **Fl_Char 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 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 position of the text cursor.
- int **position** (int p)  
  Sets the cursor position and mark.
- int **position** (int p, int m)  
  Sets the index for the cursor and mark.
- 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 **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)  
  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 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 Fl_Group * as_group ()
  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.
• 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, void *p)
  Sets the current callback function 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.

- `const Fl_Image ∗deimage () const`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image ∗img)`
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, long arg)`
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, void ∗arg=0)`
  Calls the widget callback.

- `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.

- `const FL_Image * image () const`

- `void image (FL_Image &img)`
  
  Sets the image to use as part of the widget label.

- `void image (FL_Image *img)`
  
  Sets the image to use as part of the widget label.

- `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.

- `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.

- **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 (void ∗v)**

  Sets the user data for this widget.

- **unsigned int visible () const**

  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 () const**

  Returns whether a widget and all its parents are visible.
• int $w$ () const
  
  Gets the widget width.

• Fl_When $\text{when} ()$ const
  
  Returns the conditions under which the callback is called.

• void $\text{when} (\text{uchar} \ i)$
  
  Sets the flags used to decide when a callback is called.

• Fl_Window * $\text{window} ()$ 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 $\sim\text{Fl_Widget} ()$
  
  Destroys the widget.

Protected Member Functions

• void $\text{draw} ()$
  
  Draws the widget.

Protected Member Functions inherited from Fl_Input_

• void $\text{drawtext} (\text{int}, \text{int}, \text{int}, \text{int})$
  
  Draws the text in the passed bounding box.

• void $\text{handle\_mouse} (\text{int}, \text{int}, \text{int}, \text{int}, \text{int} \ \text{keepmark}=0)$
  
  Handles mouse clicks and mouse moves.

• int $\text{handletext} (\text{int} \ \text{e}, \text{int}, \text{int}, \text{int})$
  
  Handles all kinds of text field related events.

• int $\text{line\_end} (\text{int} \ \text{i})$ const
  
  Finds the end of a line.

• int $\text{line\_start} (\text{int} \ \text{i})$ const
  
  Finds the start of a line.

• int $\text{linesPerPage} ()$

• void $\text{maybe\_do\_callback} ()$

• int $\text{up\_down\_position} (\text{int}, \text{int} \ \text{keepmark}=0)$
  
  Moves the cursor to the column given by $\text{up\_down\_pos}$.

• int $\text{word\_end} (\text{int} \ \text{i})$ const
  
  Finds the end of a word.

• int $\text{word\_start} (\text{int} \ \text{i})$ const
  
  Finds the start of a word.

• int $\text{xscroll} ()$ const

• int $\text{yscroll} ()$ const

• void $\text{yscroll} (\text{int} \ \text{yOffset})$

Protected Member Functions inherited from Fl_Widget

• void $\text{clear\_flag} (\text{unsigned} \ \text{int} \ \text{c})$
  
  Clears a flag in the flags mask.

• void $\text{draw\_backdrop} ()$ const
  
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

• void $\text{draw\_box} ()$ 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**()
  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_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 ∗cb, void ∗d)
  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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.
9.70.1 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 \textasciitilde 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 MSWindows 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 9.190 Keyboard and mouse bindings.

<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 \textasciitilde X (copy+cut).</td>
</tr>
<tr>
<td>Enter</td>
<td>May cause the callback, see when().</td>
</tr>
</tbody>
</table>

Table 9.191 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>
<tr>
<td>^J</td>
<td>^J</td>
<td>Insert a LineFeed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Similar to literal 'Enter' character)</td>
</tr>
<tr>
<td>^L</td>
<td>^L</td>
<td>Insert a Form Feed.</td>
</tr>
<tr>
<td>^M</td>
<td>^M</td>
<td>Insert a Carriage Return.</td>
</tr>
<tr>
<td>^V, Shift-Insert</td>
<td>Command-V</td>
<td>Paste the clipboard.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Macs keyboards don't have &quot;Insert&quot; keys, but if they did, Shift-Insert would work)</td>
</tr>
<tr>
<td>^X, Shift-Delete, Shift-Delete</td>
<td>Command-X, Shift-Delete</td>
<td>Cut. Copy the selection to the clipboard and delete it. (If there's no selection, Shift-Delete acts like Delete)</td>
</tr>
<tr>
<td>^Z</td>
<td>Command-Z</td>
<td>Undo.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is a single-level undo mechanism, but all adjacent deletions and insertions are concatenated into a single &quot;undo&quot;. Often this will undo a lot more than you expected.</td>
</tr>
<tr>
<td>Shift-^Z</td>
<td>Shift-Command-Z</td>
<td>Redo.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Currently same behavior as ^Z. Reserved for future multilevel undo/redo.</td>
</tr>
<tr>
<td>Arrow Keys</td>
<td>Arrow Keys</td>
<td>Standard cursor movement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
</tbody>
</table>
### 9.70.2 Constructor & Destructor Documentation

#### 9.70.2.1 Fl_Input()

```cpp
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`.

### 9.70.3 Member Function Documentation

#### 9.70.3.1 draw()

```cpp
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:

```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw();           // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

#### 9.70.3.2 handle()

```cpp
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.
9.71 Fl_Input_ Class Reference

Parameters

\[ \text{in} \quad \text{event} \quad \text{the kind of event received} \]

Return values

\[ 0 \quad \text{if the event was not used or understood} \]
\[ 1 \quad \text{if the event was used and can be deleted} \]

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Secret_Input.
The documentation for this class was generated from the following files:

- Fl_Input.H
- Fl_Input.cxx

9.71 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_Input
    Fl_File_Input Fl_Float_Input Fl_Int_Input Fl_Multiline_Input Fl_Output Fl_Secret_Input
    Fl_Multiline_Output
```

Public Member Functions

- **int copy (int clipboard)**
  
  \textit{Put the current selection into the clipboard.}

- **int copy_cuts ()**
  
  \textit{Copies the yank buffer to the clipboard.}

- **FL_Color cursor_color () const**
  
  \textit{Gets the color of the cursor.}

- **void cursor_color (FL_Color n)**
  
  \textit{Sets the color of the cursor.}

- **int cut ()**
  
  \textit{Deletes the current selection.}

- **int cut (int a, int b)**
  
  \textit{Deletes all characters between index \textit{a} and \textit{b}.}

- **int cut (int n)**
  
  \textit{Deletes the next \textit{n} bytes rounded to characters before or after the cursor.}

- **Fl_Input_ (int, int, int, int, const char * = \text{0})**
  
  \textit{Creates a new Fl_Input_ widget.}
• **Fl_Char 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 **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 position of the text cursor.

• int **position** (int p)
  
  Sets the cursor position and mark.

• int **position** (int p, int m)
  
  Sets the index for the cursor and mark.

• 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 **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)
  
  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.

• int value (const char *)

Changes the widget text.

• int value (const char *, int)

Changes the widget text.

• 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 Fl_Group * as_group ()

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.

• 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, 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)
  
  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.
• const FL_Image * deimage () const
  Sets the image to use as part of the widget label.
• void deimage (FL_Image &img)
• void deimage (FL_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (FL_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (FL_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const FL_Image * image () const
• void image (FL_Image &img)
  Sets the image to use as part of the widget label.
• void image (FL_Image *img)
  Sets the image to use as part of the widget label.
• 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
- **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.*

- 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.*

- 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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 drawtext (int, int, int, int)
  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 ()
• 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 ()
  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_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 ∗ cb, void ∗ d)
  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.
9.71 Fl_Input_ Class Reference

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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.71.1 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_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

9.71.2 Constructor & Destructor Documentation

9.71.2.1 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 |
| _________ | _______________________________ |
| l | an optional label text |

Generated by Doxygen
9.71.2.2 \texttt{\textasciitilde\texttt{Fl\_Input\_}()} 

\texttt{Fl\_Input\_::\textasciitilde\texttt{Fl\_Input\_}()} 

Destroys the widget.

The destructor clears all allocated buffers and removes the widget from the parent \texttt{Fl\_Group}.

9.71.3 Member Function Documentation

9.71.3.1 \texttt{copy()} 

\begin{verbatim}
int Fl\_Input\_::copy (  
  int clipboard  )
\end{verbatim}

Put the current selection into the clipboard.

This function copies the current selection between \texttt{mark()} and \texttt{position()} into the specified \texttt{clipboard}. This does not replace the old clipboard contents if \texttt{position()} and \texttt{mark()} are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.

Parameters

\begin{tabular}{ll}
   \textit{clipboard} & the clipboard destination 0 or 1 \\
\end{tabular}

Returns

0 if no text is selected, 1 if the selection was copied

See also

\texttt{Fl::copy(const char \*, int, int)}

9.71.3.2 \texttt{copy\_cuts()} 

\begin{verbatim}
int Fl\_Input\_::copy\_cuts ( )
\end{verbatim}

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 $\wedge K$ shortcut key.

Returns

0 if the operation did not change the clipboard

See also

\texttt{copy(int), cut()}

9.71.3.3 \texttt{cursor\_color() [1/2]} 

\begin{verbatim}
Fl\_Color Fl\_Input\_::cursor\_color ( ) const [inline]
\end{verbatim}

Gets the color of the cursor.

Returns

the current cursor color

9.71.3.4 \texttt{cursor\_color() [2/2]} 

\begin{verbatim}
void Fl\_Input\_::cursor\_color (  
  Fl\_Color n ) [inline]
\end{verbatim}

Sets the color of the cursor.

The default color for the cursor is \texttt{FL\_BLACK}. 
Parameters

- **n** the new cursor color

### 9.71.3.5 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

### 9.71.3.6 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

### 9.71.3.7 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

### 9.71.3.8 drawtext()

```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

### 9.71.3.9 handle_mouse()

```cpp
void Fl_Input_::handle_mouse (
    int \(X, Y\),
    int ,
    int ,
    int drag = 0 ) [protected]
```

Handles mouse clicks and mouse moves.

**Todo** Add comment and parameters

### 9.71.3.10 handletext()

```cpp
int Fl_Input_::handletext (
    int event,
    int \(X, Y\),
    int \(W, H\) ) [protected]
```

Handles all kinds of text field related events. This is called by derived classes.

**Todo** Add comment and parameters

### 9.71.3.11 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

- \(i\) index into the value field

Returns

- the character at index \(i\)

### 9.71.3.12 input_type() [1/2]

```cpp
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.
Returns
the current input type

9.71.3.13  input_type() [2/2]
void Fl_Input::*input_type (  
  int t ) [inline]
Sets the input field type.  
A redraw() is required to reformat the input field.

Parameters

| in  | t   | new input type |

9.71.3.14  insert()
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

| in  | t   | text that will be inserted |
| in  | l   | length of text, or 0 if the string is terminated by nul. |

Returns
0 if no text was inserted

9.71.3.15  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
end of the line

9.71.3.16  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

|   | i | starting index for the search |

Returns

start of the line

9.71.3.17 mark() [1/2]

int Fl_Input_::mark () const [inline]

Gets the current selection mark.

Returns

index into the text

9.71.3.18 mark() [2/2]

int Fl_Input_::mark ( int m ) [inline]

Sets the current selection mark.

mark(n) is the same as position(position(),n).

Parameters

|   | m | new index of the mark |

Returns

0 if the mark did not change

See also

position(), position(int, int)

9.71.3.19 maximum_size() [1/2]

int Fl_Input_::maximum_size ( ) const [inline]

Gets the maximum length of the input field in characters.

See also

maximum_size(int).

9.71.3.20 maximum_size() [2/2]

void Fl_Input_::maximum_size ( int m ) [inline]

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).

9.71.3.21 position() [1/3]

int Fl_Input_::position ( ) const [inline]

Gets the position of the text cursor.
Returns
the cursor position as an index in the range 0..size()

See also
position(int, int)

9.71.3.22 position() [2/3]

int Fl_Input_::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
position(int, int), position(), mark(int)

9.71.3.23 position() [3/3]

int Fl_Input_::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)

9.71.3.24 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

9.71.3.25 readonly() [2/2]

int Fl_Input_::readonly (int b) [inline]

Sets the read-only state of the input field.

Parameters

| in  | b | if b is 0, the text in this widget can be edited by the user |

9.71.3.26 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 \( \text{mark}() \) and \( \text{position}() \) to the end of the insertion. Does the callback if \( \text{when}() \ & \ \text{FL_WHEN_CHANGED} \) 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

| in  | b | beginning index of text to be deleted |
| in  | e | ending index of text to be deleted and insertion position |
| in  | text | string that will be inserted |
| in  | ilen | length of text or 0 for null terminated strings |

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).
9.71.3.27 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

- `in X,Y,W,H` new size of the widget

See also

- `Fl_Widget::resize(int, int, int, int)`

Reimplemented from `Fl_Widget`.

9.71.3.28 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()`

9.71.3.29 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()`

9.71.3.30 size() [1/2]

```cpp
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
9.71.3.31 size() [2/2]

```cpp
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)`

9.71.3.32 static_value() [1/2]

```cpp
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

9.71.3.33 static_value() [2/2]

```cpp
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**

| 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

9.71.3.34  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.

9.71.3.35  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-t 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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val is 1, Tab advances focus (default).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>val is 0, Tab inserts a tab character (old FLTK behavior).</td>
<td></td>
</tr>
</tbody>
</table>

See also
tab_nav(), Fl::OPTION_ARROW_FOCUS.

9.71.3.36  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)

9.71.3.37  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

**\texttt{in n}** new text color

See also

textcolor()

\textbf{9.71.3.38 textfont() [1/2]}

\texttt{Fl_Font Fl\_Input\_::textfont ( ) const \[inline\]}

Gets the font of the text in the input field.

Returns

the current \texttt{Fl\_Font} index

\textbf{9.71.3.39 textfont() [2/2]}

\texttt{void Fl\_Input\_::textfont ( }
  \texttt{Fl\_Font s) \[inline\]}

Sets the font of the text in the input field.
The text font defaults to \texttt{FL\_HELVETICA}.

Parameters

**\texttt{in s}** the new text font

\textbf{9.71.3.40 textsize() [1/2]}

\texttt{Fl\_Fontsize Fl\_Input\_::textsize ( ) const \[inline\]}

Gets the size of the text in the input field.

Returns

the text height in pixels

\textbf{9.71.3.41 textsize() [2/2]}

\texttt{void Fl\_Input\_::textsize ( }
  \texttt{Fl\_Fontsize s) \[inline\]}

Sets the size of the text in the input field.
The text height defaults to \texttt{FL\_NORMAL\_SIZE}.

Parameters

**\texttt{in s}** the new font height in pixel units

\textbf{9.71.3.42 undo()}

\texttt{int Fl\_Input\_::undo ( )}

Undoes previous changes to the text buffer.
This call undoes a number of previous calls to \texttt{replace()}. 
Returns
non-zero if any change was made.

9.71.3.43  up_down_position()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>index into the beginning of a line of text</td>
</tr>
<tr>
<td>keepmark</td>
<td>if set, move only the cursor, but not the mark</td>
</tr>
</tbody>
</table>

Returns
index to new cursor position

9.71.3.44  value() [1/3]

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*)

9.71.3.45  value() [2/3]

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 "".  

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>the new text</td>
</tr>
</tbody>
</table>

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()
9.71.3.46 value() [3/3]

```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

<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

See also

`Fl_Input_::value(const char* str), Fl_Input_::value()`

9.71.3.47 word_end()

```c++
int Fl_Input_::word_end (int i) const [protected]
```

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

| in | i | starting index for the search |

Returns

end of the word

9.71.3.48 word_start()

```c++
int Fl_Input_::word_start (int i) const [protected]
```

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

| in | i | starting index for the search |
9.71.3.49 wrap() [1/2]

int Fl_Input_::wrap ( ) const [inline]

Gets the word wrapping state of the input field.
Word wrap is only functional with multi-line input fields.

9.71.3.50 wrap() [2/2]

void Fl_Input_::wrap (  
        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

9.72 Fl_Input_Choice Class Reference

A combination of the input widget and a menu button.
#include <Fl_Input_Choice.H>

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)
  Resizes the Fl_Group widget and all of its children.

- 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 c.

- Fl_Font textfont () const
  Gets the Fl_Input text field's font style.

- void textfont (Fl_Font f)
  Sets the Fl_Input text field's font style to f.

- Fl_Fontsize textsize () const
  Gets the Fl_Input text field's font size.

- void textsize (Fl_Fontsize s)
  Sets the Fl_Input text field's font size to s.

- const char * value () const
  Returns the Fl_Input text field's current contents.

- void value (const char *val)
  Sets the Fl_Input text field's contents to val.

- void value (int val)
  Chooses item# val in the menu, and sets the Fl_Input text field to that value.

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.

- virtual Fl_Group * as_group ()
  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 ()
  Generated by Doxygen
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 *)
- void forms_end ()
  This is for forms compatibility only.

- int handle (int)
  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
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable (Fl_Widget *) o)
  The resizable widget defines the resizing box for the group.

- 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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• 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, 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)
  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.

- **const Fl_Image * deimage () const**

- **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const Fl_Image * image () const**

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  
  Sets the image to use as part of the widget label.

- **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_Labelette 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.
- `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.
- `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 (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 *cb, void *d)
  `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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

• void draw ()
  `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 ()
  `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_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 **w**(int v)
  - Internal use only.
- void **set_flag**(unsigned int c)
  - Sets a flag in the flags mask.
- void **h**(int v)
  - Internal use only.
- void **x**(int v)
  - Internal use only.
- void **y**(int v)
  - Internal use only.

### 9.72.1 Detailed Description

A combination of the input widget and a menu button.

![Figure 9.17 Fl_Input_Choice widget](image.png)

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:**

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

void choice_cb(Fl_Widget *w, void *userdata) {
  // Show info about the picked item
  Fl_Input_Choice *choice = (Fl_Input_Choice*)w;
  const Fl_Menu_Item *item = choice->menubutton()->mvalue();
  printf("*** Choice Callback:\n");
  printf(" item label()='%s', item ? item->label() : "No item"));
  printf(" item value()=%d", choice->menubutton()->value());
  printf(" The user %s, choice->menubutton()->changed()
    ? "picked a menu item"
    : "typed text");
}
```

Generated by Doxygen
```cpp
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();
}

9.72.2 Constructor & Destructor Documentation

9.72.2.1 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.

9.72.3 Member Function Documentation

9.72.3.1 add()

void Fl_Input_Choice::add {
    const char * s } [inline]

Adds an item to the menu.
You can access the more complex Fl_Menu_Button::add() methods (setting 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);
```  

9.72.3.2 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.

9.72.3.3 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.

```cpp
    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)"); 
    }
```  

9.72.3.4 resize()

void Fl_Input_Choice::resize {
    int X,
    int Y,
    int W,
    int H } [inline], [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.

9.72.3.5 value() [1/2]

void Fl_Input_Choice::value (const char * val) [inline]

Sets the Fl_Input text field's contents to val. Does not affect the menu selection.

9.72.3.6 value() [2/2]

void Fl_Input_Choice::value (int val) [inline]

Chooses item# val in the menu, and sets the Fl_Input text field to that value. Any previous text is cleared.

The documentation for this class was generated from the following files:

- Fl_Input_Choice.H
- Fl_Group.cxx

9.73 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 0xaef).

#include <Fl_Int_Input.H>

Inheritance diagram for Fl_Int_Input:

```
Fl_Widget
    ▼
   /   
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)
  
  Handles the specified event.
Public Member Functions inherited from Fl_Input_  

- **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.*

- **Fl_Input_ (int, int, int, int, const char * =0)**  
  *Creates a new Fl_Input_ widget.*

- **Fl_Char 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 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 position of the text cursor.*

- **int position (int p)**  
  *Sets the cursor position and mark.*

- **int position (int p, int m)**  
  *Sets the index for the cursor and mark.*

- **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)**  
  *Changes the size of the widget.*

- **int shortcut () const**  
  *Return the shortcut key associated with this widget.*

- **void shortcut (int s)**  
  Generated by Doxygen
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 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.
• **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_Group * as_group ()**
  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.

• **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, 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)**
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.

- `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.

- `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(void*v)
  Sets the user data for this widget.
• unsigned int visible() const
  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() 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 ∗cb, void ∗d)
  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, 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_ACCEANTS_MENU
  = 1<<19, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

• void draw ()
  Draws the widget.

Protected Member Functions inherited from Fl_Input_

• void drawtext (int, int, int, int)
  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 ()

• 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)
9.73 Fl_Int_Input Class Reference

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()
  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_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 *l = 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.

9.73.1 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 0xaf).

9.73.2 Constructor & Destructor Documentation

9.73.2.1 Fl_Int_Input()

Fl_Int_Input::Fl_Int_Input (int X, int Y, int W, int H, const char *l = 0L)

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_Int_Input.cxx

Generated by Doxygen
The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.

```cpp
#include <Fl_JPEG_Image.H>
```

Inheritance diagram for Fl_JPEG_Image:

```
Fl_Image
   ↓
Fl_RGB_Image
   ↓
Fl_JPEG_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)**
  
  The constructor loads the JPEG image from memory.

### Public Member Functions inherited from Fl_RGB_Image

- **virtual void color_average (Fl_Color c, float i)**
  
  The color_average() method averages the colors in the image with the FLTK color value `c`.

- **Fl_Image ∗ copy ()**
  
  The copy() method creates a copy of the specified image.

- **virtual void desaturate ()**
  
  The desaturate() method converts an image to grayscale.

- **void draw (int X, int Y)**
  
  Draws the image 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 W, int H, int D=3, int LD=0)**
  
  The constructor creates a new image from the specified data.

- **virtual void label (Fl_Menu_Item ∗m)**
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- **virtual void label (Fl_Widget ∗w)**
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- **virtual void uncache ()**
  
  If the image has been cached for display, delete the cache data.

- **virtual ~Fl_RGB_Image ()**
  
  The destructor frees all memory and server resources that are used by the image.

### Public Member Functions inherited from Fl_Image

- **Fl_Image ∗ copy ()**
  
  The copy() method creates a copy of the specified image.

- **int count () const**
  
  The count() method returns the number of data values associated with the image.
• int d () const  
  Returns the current image depth.
• const char ∗const ∗ data () const  
  Returns a pointer to the current image data array.
• void draw (int X, int Y)  
  Draws the image.
• int fail ()  
  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 height in pixels.
• 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.
• int w () const  
  Returns the current image width in pixels.
• 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_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).

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_NO_IMAGE = -1

Generated by Doxygen
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 array 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.

- void h(int H)
  
  Sets the current image height in pixels.

- void ld(int LD)
  
  Sets the current line data size in bytes.

- void w(int W)
  
  Sets the current image width in pixels.

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)

9.74.1 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.

9.74.2 Constructor & Destructor Documentation

9.74.2.1 Fl_JPEG_Image() [1/2]

Fl_JPEG_Image::Fl_JPEG_Image {
  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

| in | filename | a full path and name pointing to a valid jpeg file |

9.74.2.2 Fl_JPEG_Image() [2/2]

Fl_JPEG_Image::Fl_JPEG_Image {
  const char * name,
  const unsigned char * data
}

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.
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>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_JPEG_Image.H
- Fl_JPEG_Image.cxx

9.75  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

9.75.1  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.
9.75.2 Member Function Documentation

9.75.2.1 draw()

```c
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.

9.75.2.2 measure()

```c
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 needed to fit the label |

9.75.3 Member Data Documentation

9.75.3.1 type

```c
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

9.76 Fl_Light_Button Class Reference

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.

```c
#include <Fl_Light_Button.H>
```

Inheritance diagram for Fl_Light_Button:

```
Fl_Widget -> Fl_Button -> Fl_Light_Button -> Fl_Check_Button, Fl_Radio_Light_Button, Fl_Round_Button, Fl_Radio_Round_Button
```
Public Member Functions

- **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.
- virtual int handle (int)
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()
  Same as value(0).
- 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_window ()  
  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_Window * as_window ()  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• 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, 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)  
  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.
- `const Fl_Image * deimage () const`
- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label.
- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label.
- `void do_callback ()`
  Calls the widget callback.
- `void do_callback (Fl_Widget *o, long arg)`
  Calls the widget callback.
- `void do_callback (Fl_Widget *o, void *arg=0)`
  Calls the widget callback.
- `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.
- `const Fl_Image * image () const`
- `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label.
- `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label.
- `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.
• 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.
• 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.
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 (void ∗v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  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 () 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 void draw ()**
  
  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** ()
  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_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_Widget

- static void **default_callback** (Fl_Widget *cb, void *d)
  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,
  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

9.76.1 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().

![Figure 9.18 Fl_Light_Button](image_url)

9.76.2 Constructor & Destructor Documentation

9.76.2.1 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 destructor deletes the check button.

9.76.3 Member Function Documentation

9.76.3.1 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Button.

### 9.76.3.2 handle()

```cpp
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.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

#### 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

### 9.77 Fl_Line_Dial Class Reference

Inheritance diagram for Fl_Line.Dial:

```
Fl_Widget
   |
   v
Fl_Valuator
   |
   v
Fl_Dial
   |
   v
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)
  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 ()
  
  Activates the widget.

• void _set_fullscreen ()
  
  Activates the widget.

• 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 Fl_Group ∗ as_group ()
  
  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.

• 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, 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)
  
  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.

- **const FL_Image * deimage () const**
  
  void deimage (FL_Image &img)
  
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (FL_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (FL_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const FL_Image * image () const**
  
  void image (FL_Image &img)
  
  Sets the image to use as part of the widget label.
• **void image (Fl_Imagew *img)***
  
  *Sets the image to use as part of the widget label.*

• **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.*

• **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.

- `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) 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(void *v)`  
  Sets the user data for this widget.

- `unsigned int visible() const`  
  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 () 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 ∗cb, void ∗d)
    
    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
    ,
    USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

    flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

• void draw ()
    
    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.
9.78 Fl_Mac_App_Menu Class Reference

- **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 ()**
  
  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_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**

9.78  Fl_Mac_App_Menu Class Reference

Mac OS-specific class allowing to customize and localize the application menu.
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 = "About %@"
  
  Localizable text for the "About xxx" application menu item.
- static const char * hide = "Hide %@"
  
  Localizable text for the "Hide xxx" application menu item.
- static const char * hide_others = "Hide Others"
  
  Localizable text for the "Hide Others" application menu item.
- static const char * print = "Print Front Window"
  
  Localizable text for the "Print Front Window" application menu item.
- static const char * quit = "Quit %@"
  
  Localizable text for the "Quit xxx" application menu item.
- static const char * services = "Services"
  
  Localizable text for the "Services" application menu item.
- static const char * show = "Show All"
  
  Localizable text for the "Show All" application menu item.

9.78.1 Detailed Description

Mac OS-specific class allowing to customize and localize the application menu.
The public class attributes are used to build the application menu. They can be localized at run time to any UTF-8
text by placing instructions such as this before fl_open_display() gets called:

Fl_Mac_App_Menu::print = "Imprimer la fenêtre";

See also

The Apple OS X Interface for another way to localization.

9.78.2 Member Function Documentation

9.78.2.1 custom_application_menu_items()

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" item, or at its place if it was removed with Fl_Mac_App_Menu::print = "".

Parameters

- m zero-ending array of Fl_Menu_Item's.

9.78.3 Member Data Documentation

9.78.3.1 print

const char * Fl_Mac_App_Menu::print = "Print Front Window" [static]

Localizable text for the "Print Front Window" application menu item.
This menu item 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 files:

Generated by Doxygen
9.79 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_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 ∗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)**
  
  See Fl_Boxtype Fl_Menu_::down_box() const

- **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.

• `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 *, int shortcut, Fl_Callback *, 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.

• `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.

• `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 `Fl_Group * as_group ()`
  
  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`.

• `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, 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)`
  
  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.
• const Fl_Image ∗deimage () const
• void deimage (Fl_Image &img)
    Sets the image to use as part of the widget label.
• void deimage (Fl_Image ∗img)
    Sets the image to use as part of the widget label.
• void do_callback ()
    Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)
    Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
    Calls the widget callback.
• 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.
- const Fl_Image * image () const
- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.
- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
- 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.
- 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.

• 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(void *v)`
  Sets the user data for this widget.

• unsigned int `visible()` const
  Returns whether a widget is visible.

• unsigned int `visible_focus()`
Checks whether this widget has a visible focus.

- void **visible_focus** (int v)
  Modifications 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
  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** ()
  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_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.}

Protected Attributes

- uchar \texttt{alloc}
- uchar \texttt{down_box_}
- Fl\_Color \texttt{textcolor_}
- Fl\_Font \texttt{textfont_}
- Fl\_Fontsize \texttt{textsize_}

Additional Inherited Members

**Static Public Member Functions inherited from Fl\_Widget**

- static void \texttt{default\_callback (Fl\_Widget \ast cb, void \ast d)}
  \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 'sx' 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 'sx' 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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

**9.79.1 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, \texttt{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 \texttt{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 \texttt{value()}, \texttt{mvalue()}, \texttt{item\_pathname()}, etc.
9.79.2 Constructor & Destructor Documentation

9.79.2.1 Fl_Menu_()

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.

9.79.3 Member Function Documentation

9.79.3.1 add() [1/2]

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().  
No items must be added to a menu during a callback to the same menu.

Parameters

| str | string containing multiple menu labels as described above |

Returns

the index into the menu() array, where the entry was added

9.79.3.2 add() [2/2]

int Fl_Menu_::add (  
  const char * label,  
  int shortcut,  
  Fl_Callback * callback,  
  void * userdata = 0,  
  int flags = 0)

Adds a new menu item.

Parameters

| label | The text label for the menu item. |
| shortcut | Optional keyboard shortcut that can be an int or string: (FL_CTRL+"a") or "^a". Default 0 if none. |
| callback | Optional callback invoked when user clicks the item. Default 0 if none. |
| userdata | Optional user data passed as an argument to the callback. Default 0 if none. |
| flags | Optional flags that control the type of menu item; see below. Default is 0 for none. |

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.

Detailed Description of Parameters

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 "/" character is ignored if it appears as the first character of the label string, e.g. "/File/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.

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 http://fltk.org/newsgroups.←
php?gfltk.development+v:10086 and results of STR#2344

### 9.79.3.3 clear()

```cpp
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.

---

Generated by Doxygen
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:
```
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");
menubar->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)

copy()

```
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).

down_box()

```
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.

find_index()

```
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 |

Returns

The index of the matching item, or -1 if not found.

See also

item_pathname()

9.79.3.8 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.
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.

```cpp
// 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

| in | item | The item to be found |

Returns

The index of the item, or -1 if not found.

See also

menu()

9.79.3.9 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

| cb | Find the first item with this callback |

Returns

The index of the item with the specific callback, or -1 if not found.
9.79.3.10  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:

```c
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()

9.79.3.11  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

| cb | find the first item with this callback |

Returns

The item found, or NULL if not found

See also

find_item(const char*)

9.79.3.12  global()

    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 (i.e., 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!)

### 9.79.3.13 insert()

```c
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 (e.g., "Quit"), and not a 'menu pathname' (e.g., "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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `index`   | The menu array’s index position where the new item is inserted. If -1, behavior is the same as `add()`.
| `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. |
| `shortcut` | Optional keyboard shortcut. Can be an int (FL_CTRL+’a’) or a string (”&a”). Default is 0. |
| `callback` | Optional callback invoked when user clicks the item. Default 0 if none. |
| `userdata` | Optional user data passed as an argument to the callback. Default 0 if none. |
| `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()`

### 9.79.3.14 item_pathname()

```c
int Fl_Menu_::item_pathname ( char * name, int namelen, const Fl_Menu_Item * finditem = 0 ) const
```

Get the menu ‘pathname’ for the specified menu item. If `finditem==NULL`, `mvalue()` is used (the most recently picked menu item).

**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
    }
}
```

Generated by Doxygen
```c
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()`

### 9.79.3.15 `menu()` [1/2]

```c
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.

See also:

`size()` — returns the size of the Fl_Menu_Item array.

**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
}
```

### 9.79.3.16 `menu()` [2/2]

```c
void Fl_Menu_::menu ( const Fl_Menu_Item * m )
```

Sets 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.

### 9.79.3.17 `mode()` [1/2]

```c
int Fl_Menu_::mode ( const [inline]
```

Gets the flags of item i.
For a list of the flags, see Fl_Menu_Item.

### 9.79.3.18 `mode()` [2/2]

```c
void Fl_Menu_::mode ( int i, int fl ) [inline]
```

Sets the flags of item i.
For a list of the flags, see Fl_Menu_Item.
### 9.79.3.19 mvalue()

```cpp
class Fl_Menu_ 

const Fl_Menu_Item * mvalue() const [inline]
```

Returns a pointer to the last menu item that was picked.

### 9.79.3.20 picked()

```cpp
class Fl_Menu_ 

const Fl_Menu_Item * picked ( ) const 
```

Returns a pointer to the last menu item that was picked.

### 9.79.3.21 remove()

```cpp
class Fl_Menu_ 

int remove ( )
```

Deletes item `i` from the menu.

### 9.79.3.22 replace()

```cpp
class Fl_Menu_ 

int replace ( )
```

Changes the text of item `i`.

### 9.79.3.23 size()

```cpp
class Fl_Menu_ 

int size ( ) const
```

This returns the number of `Fl_Menu_Item` structures that make up the menu, correctly counting submenus.

### 9.79.3.24 testShortcut()

```cpp
class Fl_Menu_ 

const Fl_Menu_Item * testShortcut ( ) [inline]
```

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.

9.79.3.25 text() [1/2]
const char * Fl_Menu_::text ( ) const [inline]
Returns the title of the last item chosen.

9.79.3.26 text() [2/2]
const char * Fl_Menu_::text ( int i ) const [inline]
Returns the title of item i.

9.79.3.27 textcolor()
Fl_Color Fl_Menu_::textcolor ( ) const [inline]
Get the current color of menu item labels.

9.79.3.28 textfont() [1/2]
Fl_Font Fl_Menu_::textfont ( ) const [inline]
Gets the current font of menu item labels.

9.79.3.29 textfont() [2/2]
void Fl_Menu_::textfont ( Fl_Font c ) [inline]
Sets the current font of menu item labels.

9.79.3.30 textsize() [1/2]
Fl_Fontsize Fl_Menu_::textsize ( ) const [inline]
Gets the font size of menu item labels.

9.79.3.31 textsize() [2/2]
void Fl_Menu_::textsize ( Fl_Fontsize c ) [inline]
Sets the font size of menu item labels.
9.79.3.32 value() [1/3]

```cpp
int Fl_Menu_::value ( ) const [inline]
```

Returns the index into menu() of the last item chosen by the user. It is zero initially.

9.79.3.33 value() [2/3]

```cpp
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.

9.79.3.34 value() [3/3]

```cpp
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_.cxx`
- `Fl_Menu_add.cxx`
- `Fl_Menu_global.cxx`

9.80 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

```cpp
#include "Fl_Menu_Bar.H"
```

Inheritance diagram for Fl_Menu_Bar:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Menu_</td>
</tr>
<tr>
<td>Fl_Menu_Bar</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
</tr>
</tbody>
</table>
```

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)`
  
  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 *, int=0)
  Adds a new menu item.
• int **add** (const char *, const char *, Fl_Callback *, void *, 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)
  See Fl_Boxtype Fl_Menu_::down_box() const
• 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.
• 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 *, int=0)
  Inserts a new menu item at the specified index position.
• int **insert** (int index, const char *, const char *, Fl_Callback *, 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 *, 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 *)
  Sets the menu array pointer directly.
• 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.
• **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**
  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**

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 Fl_Group * as_group ()
  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.
• 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, 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)
  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.

• const **Fl_Image** ∗**deimage** () const

• void **deimage** (**Fl_Image** &img)

  Sets the image to use as part of the widget label.

• void **deimage** (**Fl_Image** ∗img)

  Sets the image to use as part of the widget label.

• void **do_callback** ()

  Calls the widget callback.

• void **do_callback** (**Fl_Widget** ∗o, long arg)

  Calls the widget callback.

• void **do_callback** (**Fl_Widget** ∗o, void ∗arg=0)

  Calls the widget callback.

• 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.

• const **Fl_Image** ∗**image** () const

• void **image** (**Fl_Image** &img)

  Sets the image to use as part of the widget label.

• void **image** (**Fl_Image** ∗img)

  Sets the image to use as part of the widget label.

• 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.

- **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.

- **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 (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  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 () 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 ()
  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 ()`
  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_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 *cb, void *d)`
  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,
  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_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

9.80.1 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 9.19 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.

9.80.2 Constructor & Destructor Documentation

9.80.2.1 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.

9.80.3 Member Function Documentation

9.80.3.1 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:
Fl_Widget ∗s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Implements Fl_Widget.
Reimplemented in Fl_Sys_Menu_Bar.

9.80.3.2 handle()

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.

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_Menu_Bar.H
- Fl_Menu_Bar.cxx
9.81 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.

#include <Fl_Menu_Button.H>

Inheritance diagram for Fl_Menu_Button:

```
Fl_Widget
   Fl_Menu_
   Fl_Menu_Button
```

Public Types

- Enum popup_buttons {
  POPUP1 = 1, 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)
  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)
  See Fl_Boxtype Fl_Menu_::down_box() const
- 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 Fl_Callback *)`
  
  Find the menu item for the given callback `cb`.

- `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 *name, int shortcut, Fl_Callback *, void *, 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 menu item.

- `const Fl_Menu_Item *menu () const`
  
  Returns a pointer to the array of `Fl_Menu_Items`.

- `void menu (const Fl_Menu_Item *)`
  
  Sets the menu array pointer directly.

- `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.

- `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)`
  
  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 Fl_Group * as_group ()
  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.
- 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, void *p)
Sets the current callback function 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.

- const FL_Image ∗deimage () const

- void deimage (FL_Image &img)
  Sets the image to use as part of the widget label.

- void deimage (FL_Image ∗img)
Sets the image to use as part of the widget label.

- void do_callback()
  Calls the widget callback.
- void do_callback(Fl_Widget *o, long arg)
  Calls the widget callback.
- void do_callback(Fl_Widget *o, void *arg=0)
  Calls the widget callback.
- 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.
- const Fl_Image *image() const
- void image(Fl_Image &img)
  Sets the image to use as part of the widget label.
- void image(Fl_Image *img)
  Sets the image to use as part of the widget label.
- 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.
- 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.

- 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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ()
  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 ()
  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_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 *cb, void *d)
  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,
  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_Color **textcolor_**
• Fl_Font **textfont_**
• Fl_Fontsize **textsize_**
9.81 Fl_Menu_Button Class Reference

9.81.1 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.

9.81.2 Member Enumeration Documentation

9.81.2.1 popup_buttons

denum Fl_Menu_Button::popup_buttons
indicate what mouse buttons pop up the menu.
Values for type() used to indicate what mouse buttons pop up the menu. Fl_Menu_Button::POPUP3 is usually what you want.

<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>
</tbody>
</table>
Enumerator

**POPUP123** pops up with any mouse button.

### 9.81.3 Constructor & Destructor Documentation

#### 9.81.3.1 Fl_Menu_Button()

```cpp
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.

### 9.81.4 Member Function Documentation

#### 9.81.4.1 draw()

```cpp
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:

```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

#### 9.81.4.2 handle()

```cpp
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.

#### 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.
9.81.4.3 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.
The documentation for this class was generated from the following files:

- Fl_Menu_Button.H
- Fl_Menu_Button.cxx

9.82 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 *=0, int=0)
  Adds a menu item.
- int add (const char *a, const char *b, Fl_Callback *c, void *d=0, 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_∗, 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 &a)
  compatibility api for FLUID, same as a.label(this)

- void image (Fl_Image ∗a)
  compatibility api for FLUID, same as a->label(this)

- int insert (int, const char ∗, int, Fl_Callback ∗, void ∗=0, int=0)
  Inserts an item at position index.

- const char ∗ label () const
  Returns the title of the item.

- void label (const char ∗a)
  See const char ∗ Fl_Menu_Item::label() const

- void label (Fl_Labeltype a, const char ∗b)
  See const char ∗ Fl_Menu_Item::label() const

- 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 labelsize () const
  Gets the label font pixel size/height.

- void labelsize (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_∗) const
  Measures width of label, including effect of & characters.

- 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, int 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 ()

  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.
• 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 labelltype_

  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

9.82.1 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()
    ulong shortcut_; // shortcut
    Fl_Callback* callback_; // callback
    void* user_data_; // user_data
    int flags; // flags
    uchar labeltype_; // labeltype
    uchar labelfont_; // labelfont
    uchar labelsize_; // labelsize
    uchar labelcolor_; // labelcolor
}
```

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", FL_ALT+'i', 0, 0, FL_MENU_TOGGLE|FL_MENU_VALUE,
    "&truth", FL_ALT+'i', 0, 0, FL_MENU_TOGGLE,
    "&beauty", FL_ALT+'i', 0, 0, FL_MENU_INVISIBLE,
    "sub&menu", 0, 0, 0, FL_SUBMENU,
    "one",
    "two",
    "three",
    [0],
    "inactive", FL_ALT+'i', 0, 0, FL_MENU_INACTIVE|FL_MENU_DIVIDER,
    [0],
    [0],
};
```

produces:
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.

### 9.82.2 Member Function Documentation

#### 9.82.2.1 add()

```c
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)
Class Documentation

9.82.2.2 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.

9.82.2.3 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.

9.82.2.4 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.

9.82.2.5 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

9.82.2.6 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

9.82.2.7 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
9.82.2.8 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.  
This method does not set the 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

9.82.2.9 check()

void Fl_Menu_Item::check ( ) [inline]

back compatibility only.

Deprecated

9.82.2.10 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.

9.82.2.11 checked()

int Fl_Menu_Item::checked ( ) const [inline]

back compatibility only.

Deprecated

9.82.2.12 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.

9.82.2.13 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.

9.82.2.14 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.
9.82.2.15  **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.

9.82.2.16  **find_shortcut()**

```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**

<table>
<thead>
<tr>
<th>ip</th>
<th>returns the index of the item, if ip is not NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>require_alt</td>
<td>if true: match only if Alt key is pressed.</td>
</tr>
</tbody>
</table>

**Returns**

- found Fl_Menu_Item or NULL

9.82.2.17  **insert()**

```c
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>in index</th>
<th>insert new items here</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mytext</td>
<td>new label string, details see above</td>
</tr>
<tr>
<td>in sc</td>
<td>keyboard shortcut for new item</td>
</tr>
<tr>
<td>in cb</td>
<td>callback function for new item</td>
</tr>
<tr>
<td>in data</td>
<td>user data for new item</td>
</tr>
<tr>
<td>in myflags</td>
<td>menu flags as described in Fl_Menu_Item</td>
</tr>
</tbody>
</table>

**Returns**

- the index into the menu() array, where the entry was added
**9.82.2.18 label()**

```c++
const char * Fl_Menu_Item::label ( ) const [inline]
```

Returns the title of the 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.

**9.82.2.19 labelcolor() [1/2]**

```c++
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.

**9.82.2.20 labelcolor() [2/2]**

```c++
void Fl_Menu_Item::labelcolor ( Fl_Color a ) [inline]
```

Sets the menu item's label color.

See also

```c++
Fl_Color Fl_Menu_Item::labelcolor() const
```

**9.82.2.21 labelfont() [1/2]**

```c++
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.

**9.82.2.22 labelfont() [2/2]**

```c++
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.

**9.82.2.23 labeltype() [1/2]**

```c++
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.

**9.82.2.24 labeltype() [2/2]**

```c++
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.
### 9.82.2.25 measure()

```cpp
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.

### 9.82.2.26 next() [1/2]

```cpp
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.

### 9.82.2.27 next() [2/2]

```cpp
const Fl_Menu_Item * Fl_Menu_Item::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.

### 9.82.2.28 popup()

```cpp
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_ * 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.*

- **X,Y** is 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).
- **title** is a character string title for the menu. If non-zero a small box appears above the menu with the title in it. 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.
- **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.

### 9.82.2.29 pulldown()

```cpp
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_ * pbutton = 0,  
    const Fl_Menu_Item * t = 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 is centered over the rectangle (like `Fl_Choice` uses). If **picked** is zero 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.
9.82.2.30 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.

9.82.2.31 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 set_only() does.

9.82.2.32 setonly()

void Fl_Menu_Item::setonly ( )
Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

Deprecated This method is dangerous if radio items are first in the menu. Use Fl_Menu_::setonly(Fl_Menu_Item*) instead.

9.82.2.33 shortcut()

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).

9.82.2.34 size()

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:
schemechoice = new Fl_Choice(X+125,Y,140,25,"FLTK Scheme");
schemechoice->add("none");
schemechoice->add("plastic");
schemechoice->add("gtk+");
schemechoice->add("gleam");
printf("schemechoice->menu()->size() = %d\n", schemechoice->menu()->size());
Output:
schemechoice->menu()->size() = 5

9.82.2.35 submenu()

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.
9.82.2.36 test_shortcut()

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 '

9.82.2.37 uncheck()

void Fl_Menu_Item::uncheck ( ) [inline]
back compatibility only.

Deprecated

9.82.2.38 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. You should not rely on a particular value,
only zero or non-zero.

Note
The returned value for a checked menu item as of FLTK 1.3.2 is FL_MENU_VALUE (4), but may be 1 in a
future version.

The documentation for this struct was generated from the following files:

- Fl_Menu_Item.H
- Fl_Menu.cxx
- Fl_Menu_.cxx
- Fl_Menu_add.cxx

9.83 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:

```
+------------------+
| Fl_Widget        |
|                  |
+------------------+
| Fl_Group         |
|                  |
+------------------+
| Fl_Window        |
|                  |
+------------------+
| Fl_Single_Window |
|                  |
+------------------+
| Fl_Menu_Window    |
```

Public Member Functions

- void clear_overlay ()
  Tells FLTK to use normal drawing planes instead of overlay planes.
- void erase ()
  Erases the window, does nothing if HAVE_OVERLAY is not defined config.h.
- **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.

- void **flush** ()
  Forces the window to be drawn, this window is also made current and calls `draw()`.

- void **hide** ()
  Removes the window from the screen.

- unsigned int **overlay** ()
  Tells if hardware overlay mode is set.

- void **set_overlay** ()
  Tells FLTK to use hardware overlay planes if they are available.

- void **show** ()
  Puts the window on the screen.

- ~**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 **flush** ()
  Forces the window to be drawn, this window is also made current and calls `draw()`.

- int **make_current** ()

- void **show** ()
  Puts the window on the screen.

- void **show** (int a, char **b)

**Public Member Functions inherited from Fl_Window**

- virtual **Fl_Window** * as_window ()
  Returns an `Fl_Window` pointer if this widget is an `Fl_Window`.

- unsigned int **border** () const
  See void `Fl_Window::border(int)`

- 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.

- 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** ()
  Returns the window height including any window title bar and any frame added by the window manager.
• int decorated_w ()
  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.
• Fl_Window (int w, int h, const char ∗ title=0)
  Creates a window from the given size and title.
• Fl_Window (int x, int y, int w, int h, const char ∗ title=0)
  Creates a window from the given position, size 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.
• virtual int handle (int)
  Handles the specified event.
• 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)
  Sets the current icon window target dependent data.
• 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.
• 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.

- 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.

- unsigned int **override** () const
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

- virtual void **resize** (int X, int Y, int W, int H)
  Changes the size and position of 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/FL_OVERRIDE.

- void **set_tooltip_window** ()
  Marks the window as a tooltip window.

- 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** (int argc, char ∗∗argv)
  Puts the window on the screen 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 **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.
- **virtual Fl_Group * as_group ()**
  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**
  See void Fl_Group::resizable(Fl_Widget *box)
• void resizable (Fl_Widget &o)
   See void Fl_Group::resizable(Fl_Widget *box)
• void resizable (Fl_Widget *o)
   The resizable widget defines the resizing box for the group.
• 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.
• 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, 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)
   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.
• const Fl_Image ∗deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image ∗image () const
• void image (Fl_Image &img)
Sets the image to use as part of the widget label.

- **void** image (FL_Image *img)
  Sets the image to use as part of the widget label.

- **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.

- **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.

• 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** (void ∗ v)
  
  Sets the user data for this widget.

• unsigned int **visible** () const
  
  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** () 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_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 const char * default_xclass ()
  Returns the default xclass.
• static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

• static Fl_Group * current ()
  Returns the currently active group.
• static void current (Fl_Group *)
  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_ACCEENTS_MENU
  = 1<<19,
  USERFLAG2 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

• virtual void draw ()
  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`()
  Deletess all icons previously attached to the window.

Protected Member Functions inherited from `Fl_Group`

- 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 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` ()
  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_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_Window

- `shape_data_type * shape_data_`
  non-null means the window has a non-rectangular shape

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_
  Stores the last window that was made current.

9.83.1 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.

9.83.2 Member Function Documentation

9.83.2.1 clear_overlay()

```cpp
void Fl_Menu_Window::clear_overlay ( ) [inline]
```

Tells FLTK to use normal drawing planes instead of overlay planes.
This is usually necessary if your menu contains multi-color pixmaps.

9.83.2.2 flush()

```cpp
void Fl_Menu_Window::flush ( ) [virtual]
```

Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.

9.83.2.3 hide()

```cpp
void Fl_Menu_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_Window.

9.83.2.4 set_overlay()

```cpp
void Fl_Menu_Window::set_overlay ( ) [inline]
```

Tells FLTK to use hardware overlay planes if they are available.

9.83.2.5 show()

```cpp
void Fl_Menu_Window::show ( ) [virtual]
```

Puts the window on the screen.
Usually (on X) this has the side effect of opening the display.
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_Window.

The documentation for this class was generated from the following files:

- Fl_Menu_Window.H
- Fl_Menu_Window.cxx

9.84 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
   ↓
Fl_Group
   ↓
Fl_Browser_
   ↓
Fl_Browser
   ↓
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 ()
  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 ()
  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)
  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 position () const
  Gets the vertical scroll position of the list as a pixel position pos.
• void position (int pos)
  Sets the vertical scroll position of the list to pixel position pos.
• void resize (int X, int Y, int W, int H)
  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
  
  This method has been deprecated, existing for backwards compatibility only.

- **void scrollbar_width (int width)**
  
  This method has been deprecated, existing for backwards compatibility only.

- **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 **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 **newSize**.

**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.

- **virtual Fl_Group ∗as_group ()**
  
  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.

- **int handle (int)**
  
  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**
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget &o)**
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void resizable (Fl_Widget *o)**
  
  The resizable widget defines the resizing box for the group.

- **void resize (int, int, int, int)**
  
  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 Fl_Window ∗ as_window ()

  *Returns an Fl_Window pointer if this widget is an Fl_Window.*

• 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, 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)

  *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()()

  *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.

- **const Fl_Image * deimage () const**

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image * image () const**

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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.
• 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.
• 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 (void * v)`  
  Sets the user data for this widget.

- `unsigned int visible () const`  
  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 () 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 *cb, void *d)
  
  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 , VISISIBLE_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 ,
  
  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
  
  The height of the entire list of all visible() items in pixels.

- int incr_height () const
  
  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
  
  Return the item at specified line.

- void item_draw (void *item, int X, int Y, int W, int H) const
  
  Draws item at the position specified by XYWH.
• void * item_first () const
  Returns the very first item in the list.
• int item_height (void *item) const
  Returns height of item in pixels.
• void * item_last () const
  Returns the very last item in the list.
• void * item_next (void *item) const
  Returns the next item after item.
• void * item_prev (void *item) const
  Returns the previous item before item.
• void item_select (void *item, int val)
  Change the selection state of item to the value val.
• int item_selected (void *item) const
  See if item is selected.
• void item_swap (void *a, void *b)
  Swap the items a and b.
• const char * item_text (void *item) const
  Returns the label text for item.
• int item_width (void *item) const
  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 ()
  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

• void **draw** ()
  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** ()
  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_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)
9.84.1 Detailed Description

The `Fl_Multi_Browser` class is a subclass of `Fl_Browser` which lets the user select any set of the lines. 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.

9.84.2 Constructor & Destructor Documentation

9.84.2.1 `Fl_Multi_Browser()`

`Fl_Multi_Browser::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. 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`

9.85 `Fl_Multi_Label` Struct Reference

This struct allows multiple labels to be added to objects that might normally have only one label.

```
#include <Fl_Multi_Label.H>
```

**Public Member Functions**

- `void label (Fl_Menu_Item ∗)`
  
  This method is used to associate a `Fl_Multi_Label` with a `Fl_Menu_Item`.

- `void label (Fl_Widget ∗)`
  
  This method is used to associate a `Fl_Multi_Label` with a `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`
9.85.1 Detailed Description

This struct allows multiple labels to be added to objects that might normally have only one label. This struct allows a mixed text and/or graphics label to be applied to an object that would normally only have a single (usually text only) label. 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. 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 might 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.

See also
Fl_Label and Fl_Labeltype

9.85.2 Member Data Documentation

9.85.2.1 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*).

9.85.2.2 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*).

9.85.2.3 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.

9.85.2.4 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

9.86 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)
  
  *Creates a new Fl_Input widget using the given position, size, and label string.*
- **int handle** (int)
  
  *Handles the specified event.*

### Public Member Functions inherited from Fl_Input_
- **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
  
  *Sets 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.*
- **Fl_Input_** (int, int, int, int, const char ∗=0)
  
  *Creates a new Fl_Input_ widget.*
- **Fl_Char 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 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 position of the text cursor.

• int position (int p)
  
  Sets the cursor position and mark.

• int position (int p, int m)
  
  Sets the index for the cursor and mark.

• 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 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)
  
  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 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 Fl_Group * as_group ()
  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.
• 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, 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)
  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.
• const Fl_Image * deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image ∗image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

• 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** (void ∗ v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus ()
  Checks whether this widget has a visible focus.

• void **visible_focus** (int v)
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.

Additional Inherited Members

Static Public Member Functions inherited from `Fl_Widget`

- static void `default_callback` (Fl_Widget *`cb`, void *`d`)
  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 `requireAlt` = 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,
  USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31 }
  flags possible values enumeration.

Protected Member Functions inherited from `Fl_Input`

- void `draw` ()
  Draws the widget.
Protected Member Functions inherited from Fl_Input_

- void drawtext (int, int, int, int)
  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 ()
- 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 ()
  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_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.
9.86.1 Detailed Description

This input field displays `\n` characters as new lines rather than `\r`, 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.

9.86.2 Constructor & Destructor Documentation

9.86.2.1 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

9.87 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

- \texttt{Fl\_Multiline\_Output} (int \(X\), int \(Y\), int \(W\), int \(H\), const char *\(l\)=0)
  
  Creates a new \texttt{Fl\_Multiline\_Output} widget using the given position, size, and label string.

Public Member Functions inherited from \texttt{Fl\_Output}

- \texttt{Fl\_Output} (int \(X\), int \(Y\), int \(W\), int \(H\), const char *\(l\)=0)
  
  Creates a new \texttt{Fl\_Output} widget using the given position, size, and label string.

Public Member Functions inherited from \texttt{Fl\_Input}

- \texttt{Fl\_Input} (int, int, int, int, const char *\(l\)=0)
  
  Creates a new \texttt{Fl\_Input} widget using the given position, size, and label string.

- int \texttt{handle} (int)
  
  Handles the specified event.

Public Member Functions inherited from \texttt{Fl\_Input\_}

- int \texttt{copy} (int clipboard)
  
  
  \textit{Put the current selection into the clipboard.}

- int \texttt{copy\_cuts} ()
  
  \textit{Copies the yank buffer to the clipboard.}

- \texttt{Fl\_Color} \texttt{cursor\_color} () const
  
  \textit{Gets the color of the cursor.}

- void \texttt{cursor\_color} (\texttt{Fl\_Color} \(n\))
  
  \textit{Sets the color of the cursor.}

- int \texttt{cut} ()
  
  \textit{Deletes the current selection.}

- int \texttt{cut} (int \(a\), int \(b\))
  
  \textit{Deletes all characters between index \(a\) and \(b\).}

- int \texttt{cut} (int \(n\))
  
  \textit{Deletes the next \(n\) bytes rounded to characters before or after the cursor.}

- \texttt{Fl\_Input\_} (int, int, int, int, const char *\(l\)=0)
  
  Creates a new \texttt{Fl\_Input\_} widget.

- \texttt{Fl\_Char} \texttt{index} (int \(i\)) const
  
  \textit{Returns the character at index \(i\).}

- int \texttt{input\_type} () const
  
  \textit{Gets the input field type.}

- void \texttt{input\_type} (int \(t\))
  
  \textit{Sets the input field type.}

- int \texttt{insert} (const char *\(t\), int \(l\)=0)
  
  \textit{Inserts text at the cursor position.}

- int \texttt{mark} () const
  
  \textit{Gets the current selection mark.}

- int \texttt{mark} (int \(m\))
  
  \textit{Sets the current selection mark.}

- int \texttt{maximum\_size} () const
  
  \textit{Gets the maximum length of the input field in characters.}

- void \texttt{maximum\_size} (int \(m\))
  
  \textit{Sets the maximum length of the input field in characters.}

- int \texttt{position} () const
Gets the position of the text cursor.

- **int position (int p)**
  Sets the cursor position and mark.

- **int position (int p, int m)**
  Sets the index for the cursor and mark.

- **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 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)**
  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 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 Fl_Group ∗ as_group ()
  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.
• 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, 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)
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• `const Fl_Image * image () const`
• `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label.
• `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label.
• `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.
• `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.
• 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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ∗ cb, void ∗ d)
  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,
  USERFLAG3 = 1 << 29,
  USERFLAG2 = 1 << 30,
  USERFLAG1 = 1 << 31
};
```

#### Protected Member Functions inherited from Fl_Input

- void draw ()
  Draws the widget.

### Protected Member Functions inherited from Fl_Input_

- void drawtext (int, int, int, int)
  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`()

• 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` ()

  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_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.

9.87.1 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.
9.87.2 Constructor & Destructor Documentation

9.87.2.1 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_Input.cxx

9.88 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".
- \texttt{Fl\_Native\_File\_Chooser} (int val=BROWSE\_FILE)
  
The constructor.
- \texttt{int options () const}
  
  Gets the platform specific Fl\_Native\_File\_Chooser::Option flags.
- \texttt{void options (int o)}
  
  Sets the platform specific chooser options to \texttt{val}.
- \texttt{const char \* preset_file () const}
  
  Get the preset filename.
- \texttt{void preset_file (const char \* f)}
  
  Sets the default filename for the chooser.
- \texttt{int show ()}
  
  Post the chooser's dialog.
- \texttt{const char \* title () const}
  
  Get the title of the file chooser's dialog window.
- \texttt{void title (const char \* t)}
  
  Set the title of the file chooser's dialog window.
- \texttt{int type () const}
  
  Gets the current Fl\_Native\_File\_Chooser::Type of browser.
- \texttt{void type (int t)}
  
  Sets the current Fl\_Native\_File\_Chooser::Type of browser.
- \texttt{\textbar Fl\_Native\_File\_Chooser ()}
  
  Destructor.

\textbf{Static Public Attributes}

- \texttt{static const char \* file_exists_message = "File exists. Are you sure you want to overwrite?"}
  
  Localizable message.

\textbf{9.88.1 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
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE\_FILE);
fnfc.filter("Text\text\*\.txt\n" "C Files\text\*\.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 \texttt{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 \texttt{fl\_fopen()} or \texttt{fl\_open()} utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the MSWindows platform to open files with a non-ASCII name).

\textbf{Platform Specific Caveats}
• Under X windows, and if \texttt{FL::OPTION\_FNFC\_USES\_GTK} has not been switched off, the widget attempts to use standard GTK file chooser dialogs if they are available at run-time on the platform, and falls back to use FLTK’s \texttt{Fl\_FileChooser} if they are not. In the latter case, it’s best if you call \texttt{Fl\_FileChooser::load\_system\_icons()} at the start of \texttt{main()}, to enable the nicer looking file browser widgets. Use the static public attributes of class \texttt{Fl\_FileChooser} to localize the browser.

• Some operating systems support certain OS specific options; see \texttt{Fl\_Native\_FileChooser::options()} for a list.

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{NO_OPTIONS}</td>
<td>no options enabled</td>
</tr>
<tr>
<td>\texttt{SAVEAS_CONFIRM}</td>
<td>Show native ‘Save As’ overwrite confirm dialog.</td>
</tr>
<tr>
<td>\texttt{NEW_FOLDER}</td>
<td>Show ‘New Folder’ icon (if supported)</td>
</tr>
<tr>
<td>\texttt{PREVIEW}</td>
<td>enable preview mode (if supported)</td>
</tr>
<tr>
<td>\texttt{USE_FILTER_EXT}</td>
<td>Chooser filter pilots the output file extension (if supported)</td>
</tr>
</tbody>
</table>

Figure 9.22 The \texttt{Fl\_Native\_FileChooser} on different platforms

### 9.88.2 Member Enumeration Documentation

#### 9.88.2.1 Option

\begin{verbatim}
enum Fl_Native_File_Chooser::Option

enum Fl_Native_File_Chooser::Option

No_OPTIONS no options enabled
SAVEAS_CONFIRM Show native ‘Save As’ overwrite confirm dialog.
NEW_FOLDER Show ‘New Folder’ icon (if supported)
PREVIEW enable preview mode (if supported)
USE_FILTER_EXT Chooser filter pilots the output file extension (if supported)

#### 9.88.2.2 Type

\begin{verbatim}
enum Fl_Native_File_Chooser::Type

Generated by Doxygen
### Enumerator

<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>

### 9.88.3 Constructor & Destructor Documentation

#### 9.88.3.1 Fl_Native_File_Chooser()

```cpp
Fl_Native_File_Chooser::Fl_Native_File_Chooser ( int val = BROWSE_FILE )
```

The constructor.
Internally allocates the native widgets. Optional `val` presets the type of browser this will be, which can also be changed with `type()`.

#### 9.88.3.2 ~Fl_Native_File_Chooser()

```cpp
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ( )
```

Destructor.
Deallocates any resources allocated to this widget.

### 9.88.4 Member Function Documentation

#### 9.88.4.1 count()

```cpp
int Fl_Native_File_Chooser::count ( ) const
```

Returns the number of filenames (or directory names) the user selected.

**Example:**

```cpp
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for (int n=0; n<fnfc->count(); n++ ) {
        printf("%d) '%s'\n", n, fnfc->filename(n));
    }
}
```

#### 9.88.4.2 directory()

```cpp
void Fl_Native_File_Chooser::directory ( const char * val )
```

Preset the directory the browser will show when opened.
If `val` is NULL, or no directory is specified, the chooser will attempt to use the last non-cancelled folder.

#### 9.88.4.3 errmsg()

```cpp
const char * Fl_Native_File_Chooser::errmsg ( ) const
```

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 `errmsg()` will have info on failures.

#### 9.88.4.4 filename() [1/2]

```cpp
const char * Fl_Native_File_Chooser::filename ( ) const
```

Return the filename the user chose.
Use this if only expecting a single filename. If more than one filename is expected, use `filename(int)` instead. Return value may be "" if no filename was chosen (eg. user cancelled).
9.88.4.5 filename() [2/2]

const char * Fl_Native_File_Chooser::filename (  
        int i ) const

Return one of the filenames the user selected.  
Use count() to determine how many filenames the user selected.  

Example:
if ( fnfc->show() == 0 ) {  
    // Print all filenames user selected  
    for ( int n=0; n<fnfc->count(); n++ ) {  
        printf("%s\n", n, fnfc->filename(n));  
    }  
}

9.88.4.6 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.

9.88.4.7 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}\n*.txt")
- A list of descriptive names and wildcards (eg. "C++ Files\t*.{cxx,h}\nTxt Files\t*.txt")

The format of each filter is a wildcard, or an optional user description followed by 't' and the wildcard.  
On most platforms, each filter is available to the user via a pulldown menu in the file chooser.  
The 'All Files' option is always available to the user.

9.88.4.8 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.

9.88.4.9 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.

9.88.4.10 options()

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>
</table>

Generated by Doxygen
9.88.4.11  preset_file()

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.

9.88.4.12  show()

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

9.88.4.13  title() [1/2]

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.

9.88.4.14  title() [2/2]

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_FLTK.cxx

9.89  Fl_Nice_Slider Class Reference

Inheritance diagram for Fl_Nice_Slider:

Fl_Widget
   ↓
Fl_Valuator
   ↓
Fl_Slider
   ↓
Fl_Nice_Slider

Generated by Doxygen
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)
  
  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}.

- double **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 Fl_Group ∗ as_group ()
  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.
- 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, 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)
  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.
• const Fl_Image ∗deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.

• FI_**Image** ∗ **image** ()
  Gets the image that is used as part of the widget label.

• const FI_**Image** ∗ **image** () const
  Sets the image to use as part of the widget label.

• void **image** (FI_**Image** &img)
  Sets the image to use as part of the widget label.

• int **inside** (const FI_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 **labelsiz**e () const
  Gets the font size in pixels.

• void **labelsiz**e (FI_Fontsize pix)
  Sets the font size in pixels.

• FI_Labeltype **labelftype** () const
  Gets the label type.

• void **labelftype** (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.

• 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.

• virtual void **resize (int x, int y, int w, int h)**
  Changes the size or position of the widget.

• **Fl_Col**or **selection_color ()** const
  Gets the selection color.

• void **selection_color (Fl_Col 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.

• 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 (void ∗v)**
  Sets the user data for this widget.

• unsigned int **visible ()** const
  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** () 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 ∗cb, void ∗d)
  
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}

  flags possible values enumeration.

**Protected Member Functions inherited from Fl_Slider**

- void **draw** ()
  
  Draws the widget.

- void **draw** (int, int, int, int)

- int **handle** (int, int, int, int)
Protected Member Functions inherited from \texttt{Fl\_Valuator}

- \texttt{Fl\_Valuator (int X, int Y, int W, int H, const char *L)}
  
  \textit{Creates a new Fl\_Valuator widget using the given position, size, and label string.}

- \texttt{void handle\_drag (double newvalue)}
  
  \textit{Called during a drag operation, after an FL\_WHEN\_CHANGED event is received and before the callback.}

- \texttt{void handle\_push ()}
  
  \textit{Stores the current value in the previous value.}

- \texttt{void handle\_release ()}
  
  \textit{Called after an FL\_WHEN\_RELEASE event is received and before the callback.}

- \texttt{int horizontal () const}
  
  \textit{Tells if the valuator is an FL\_HORIZONTAL one.}

- \texttt{double previous\_value () const}
  
  \textit{Gets the previous floating point value before an event changed it.}

- \texttt{void set\_value (double v)}
  
  \textit{Sets the current floating point value.}

- \texttt{double softclamp (double)}
  
  \textit{Clamps the value, but accepts v if the previous value is not already out of range.}

- \texttt{virtual void value\_damage ()}
  
  \textit{Asks for partial redraw.}

Protected Member Functions inherited from \texttt{Fl\_Widget}

- \texttt{void clear\_flag (unsigned int c)}
  
  \textit{Clears a flag in the flags mask.}

- \texttt{void draw\_backdrop () const}
  
  \textit{If FL\_ALIGN\_IMAGE\_BACKDROP is set, the image or deimage will be drawn.}

- \texttt{void draw\_box () const}
  
  \textit{Draws the widget box according its box style.}

- \texttt{void 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.}

- \texttt{void 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.}

- \texttt{void draw\_focus ()}
  
  \textit{draws a focus rectangle around the widget}

- \texttt{void draw\_focus (Fl\_Boxtype t, int x, int y, int w, int h) const}
  
  \textit{Draws a focus box for the widget at the given position and size.}

- \texttt{void draw\_label () const}
  
  \textit{Draws the widget's label at the defined label position.}

- \texttt{void 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.}

- \texttt{unsigned int flags () const}
  
  \textit{Gets the widget flags mask.}

- \texttt{void h (int v)}
  
  \textit{Internal use only.}

- \texttt{void set\_flag (unsigned int c)}
  
  \textit{Sets a flag in the flags mask.}

- \texttt{void w (int v)}
  
  \textit{Internal use only.}

- \texttt{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

### 9.90 Fl_Output Class Reference

This widget displays a piece of text.

```cpp
#include <Fl_Output.H>
```

Inheritance diagram for Fl_Output:

```
Fl_Widget
   |
   V
Fl_Input_
   |
   V
Fl_Input
   |
   V
Fl_Output
   |
   V
Fl_Multiline_Output
```

#### 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 ∗=0)**
  
  Creates a new Fl_Input widget using the given position, size, and label string.

- **int handle (int)**
  
  Handles the specified event.

#### Public Member Functions inherited from Fl_Input_

- **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.

• **Fl_Input_** (int, int, int, int, const char *s=0)
  
  Creates a new Fl_Input_ widget.

• **Fl_Char 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 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 position of the text cursor.

• **int position** (int p)
  
  Sets the cursor position and mark.

• **int position** (int p, int m)
  
  Sets the index for the cursor and mark.

• **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 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)
  
  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** `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 ()`
  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 Fl_Group ∗ as_group ()**
  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`.

- **Fl_Boxtype** `box () const`
• Fl_Boxtype box (const Fl_Boxtype new_box)  
  Gets the box type of the widget.  

• Fl_Callback callback () const  
  Gets the current callback function for the widget.  

• Fl_Callback callback (Fl_Callback cb)  
  Sets the current callback function for the widget.  

• Fl_Callback callback (Fl_Callback cb, void *p)  
  Sets the current callback function for the widget.  

• Fl_Callback callback (Fl_Callback0 cb)  
  Sets the current callback function for the widget.  

• Fl_Callback 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.  

• Fl_Color color (const Fl_Color bg)  
  Sets the background color of the widget.  

• Fl_Color color (const Fl_Color bg, const 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_Imag**e **deimage** ()
  Gets the image that is used as part of the widget label.

• const **FL_Imag**e **deimage** () const

• void **deimage** (**FL_Imag**e &img)
  Sets the image to use as part of the widget label.

• void **deimage** (**FL_Imag**e *img)
  Sets the image to use as part of the widget label.

• void **do_callback** ()
  Calls the widget callback.

• void **do_callback** (**FL_Widget** *o, long arg)
  Calls the widget callback.

• void **do_callback** (**FL_Widget** *o, void *arg=0)
  Calls the widget callback.

• 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_Imag**e **image** ()
  Gets the image that is used as part of the widget label.

• const **FL_Imag**e **image** () const

• void **image** (**FL_Imag**e &img)
  Sets the image to use as part of the widget label.

• void **image** (**FL_Imag**e *img)
  Sets the image to use as part of the widget label.

• 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.
• 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.
• 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 (void ∗v)
  
  Sets the user data for this widget.

• unsigned int visible () const
  
  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 () 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 ∗cb, void ∗d)
  
  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 , VISISIBLE_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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 
}
Protected Member Functions inherited from Fl_Input

• void **draw** ()
  
  Draws the widget.

Protected Member Functions inherited from Fl_Input_

• void **drawtext** (int, int, int, int)
  
  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.

• **linesPerPage** ()

• void **maybe_do_callback** ()

• 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.

• **xscroll** () const

• **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** ()
  
  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_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)
Class Documentation

9.90 Class Documentation

9.90.1 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.

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.

9.90.2 Constructor & Destructor Documentation

9.90.2.1 `Fl_Output()`

```cpp
Fl_Output::Fl_Output (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0 )
```
9.91 Fl_Overlay_Window Class Reference

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

9.91 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

- `int can_do_overlay ()`
  
  Returns non-zero if there's hardware overlay support.

- `void flush ()`
  
  Forces the window to be redrawn.

- `void hide ()`
  
  Removes the window from the screen.

- `void redraw_overlay ()`
  
  Call this to indicate that the overlay data has changed and needs to be redrawn.

- `void resize (int, int, int, int)`
  
  Changes the size and position of the window.

- `void show ()`
  
  Puts the window on the screen.

- `void 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 (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 show (int a, char **b)`

- `~Fl_Double_Window ()`
  
  The destructor also deletes all the children.

Generated by Doxygen
Public Member Functions inherited from Fl_Window

• virtual Fl_Window * as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• unsigned int border () const
  
  See void Fl_Window::border(int)

• 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.

• 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 ()
  
  Returns the window height including any window title bar and any frame added by the window manager.

• int decorated_w ()
  
  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.

• Fl_Window (int w, int h, const char *title=0)
  
  Creates a window from the given size and title.

• Fl_Window (int x, int y, int w, int h, const char *title=0)
  
  Creates a window from the given position, size 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.

• virtual int handle (int)
  
  Handles the specified event.

• 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)
  Sets the current icon window target dependent data.

• 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.

• 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.

• 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.

• unsigned int override () const
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

• 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|FL_OVERRIDE.

• void set_tooltip_window ()
  Marks the window as a tooltip window.

• 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 (int argc, char **argv)
Puts the window on the screen 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 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.

- **virtual Fl_Group ∗as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget ∗) const**
  See int Fl_Group::find(const Fl_Widget ∗w) const.

- **int find (const Fl_Widget ∗) 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 &i, 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
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget &o)
  
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget *o)
  
  The resizable widget defines the resizing box for the group.

- **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.

- **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, 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)**
  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 ()**
  Generated by Doxygen
Gets the image that is used as part of the widget label.

- const Fl_Image * deimage () const
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image &img)

- void deimage (Fl_Image *img)

- void do_callback ()
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, long arg)

- void do_callback (Fl_Widget *o, void *arg=0)

- 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 ()

- const Fl_Image * image () const

- void image (Fl_Image &img)

- void image (Fl_Image *img)

- 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)

- 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.
• 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.

• 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 (void ∗v)
Sets the user data for this widget.

• unsigned int visible () const
  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 () 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_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)
  Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.

Protected Member Functions

• virtual void draw_overlay ()=0
  You must subclass Fl_Overlay_Window and provide this method.

• 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)

Protected Member Functions inherited from Fl_Double_Window

• void flush (int eraseoverlay)
  Forces the window to be redrawn.

Protected Member Functions inherited from Fl_Window

• virtual void draw ()
  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.
Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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_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 const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *)
  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *, void *d)
  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,
  USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31 }
  flags possible values enumeration.

Protected Attributes inherited from Fl_Double_Window

- char force_doublebuffering_
  Force double buffering, even if the OS already buffers windows (overlays need that on MacOS and Windows2000)
Protected Attributes inherited from **Fl_Window**
- **shape_data_type** *shape_data_*
  
  *non-null means the window has a non-rectangular shape*

Static Protected Attributes inherited from **Fl_Window**
- **static Fl_Window** *current_*

  *Stores the last window that was made current.*

### 9.91.1 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.

### 9.91.2 Constructor & Destructor Documentation

#### 9.91.2.1 **Fl_Overlay_Window()**

    Fl_Overlay_Window::Fl_Overlay_Window ( int X, int Y, int W, 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.

### 9.91.3 Member Function Documentation

#### 9.91.3.1 **draw_overlay()**

    virtual void Fl_Overlay_Window::draw_overlay ( ) [protected], [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>`.

#### 9.91.3.2 **flush()**

    void Fl_Overlay_Window::flush ( ) [virtual]

Forces the window to be redrawn.

Reimplemented from **Fl_Double_Window**.

#### 9.91.3.3 **hide()**

    void Fl_Overlay_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_Double_Window**.

#### 9.91.3.4 **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().

### 9.91.3.5 resize()

```cpp
void Fl_Overlay_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_Double_Window`.

### 9.91.3.6 show()

```cpp
void Fl_Overlay_Window::show ( ) [virtual]
```

Puts the window on the screen. Usually (on X) this has the side effect of opening the display.

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_Double_Window`.

The documentation for this class was generated from the following files:

- `Fl_Overlay_Window.H`
- `Fl_Double_Window.cxx`
- `Fl_Overlay_Window.cxx`

### 9.92 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`: 

```cpp
Generated by Doxygen
```
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
  Same as Fl_Group::type()
- 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.
- virtual Fl_Group ∗ as_group ()
  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, 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.

- int handle (int)
  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
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)

- void resizable (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.

- void resize (int, int, int, int)
  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 Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

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, 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)

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 copyTooltip (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.
• const Fl_Image * deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image * image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.

• `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.

• `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 (void ∗v)
  Sets the user data for this widget.
- **unsigned int** visible () const
  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 () 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** virtual ~Fl_Widget ()
  Destroys the widget.

Protected Member Functions

- **void** draw ()
  Draws the widget.

Protected Member Functions inherited from Fl_Group

- **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** ()
  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_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 ∗ cb, void ∗ d)
  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, COPY_TOOTPTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.92.1 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.
This widget is needed for the Fl_Tabs. In addition you may want to put the Fl_Pack inside an Fl_Scroll.
The resizable for Fl_Pack is set to NULL by default.
See also: Fl_Group::resizable()

9.92.2 Constructor & Destructor Documentation

9.92.2.1 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 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 it's children can be
automatic (local) variables, but you must declare the Fl_Pack first, so that it is destroyed last.

9.92.3 Member Function Documentation

9.92.3.1 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Group.
The documentation for this class was generated from the following files:

- Fl_Pack.H
- Fl_Pack.cxx
9.93  **Fl_Paged_Device Class Reference**

Represents page-structured drawing surfaces.

```cpp
#include <Fl_Paged_Device.H>
```

Inheritance diagram for Fl_Paged_Device:

![Inheritance diagram](image)

**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, 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.

**Public Member Functions**

- const char * class_name ()
  Returns the name of the class of this object.

- 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.

- virtual void origin (int x, int y)
  Sets the position in page coordinates of the current origin of graphics functions.

- virtual void origin (int x, int y)
  Sets the position in page coordinates of the origin of graphics functions.

- virtual void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
Draws the widget on the printed page.

- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Prints a window with its title bar and frame if any.

- virtual void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  Prints a rectangular part of an on-screen window.

- virtual int printable_rect (int *w, int *h)
  Computes the width and height of the printable area of the page.

- 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.

- virtual int start_job (int pagecount, int *frompage=NULL, int *topage=NULL)
  Starts a print job.

- virtual int start_page (void)
  Starts a new printed page.

- virtual void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.

- virtual void untranslate (void)
  Undoes the effect of a previous translate() call.

- virtual ∼Fl_Paged_Device ()
  The destructor.

Public Member Functions inherited from Fl_Surface_Device

- const char * class_name ()
  Returns the name of the class of this object.

- Fl_Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.

- void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.

- virtual void set_current (void)
  Make this surface the current drawing surface.

- virtual ∼Fl_Surface_Device ()
  The destructor.

Public Member Functions inherited from Fl_Device

- virtual ∼Fl_Device ()
  Virtual destructor.

Static Public Attributes

- static const char * class_id = "Fl_Paged_Device"

Static Public Attributes inherited from Fl_Surface_Device

- static const page_format page_formats [NO_PAGE_FORMATS]
  width, height and name of all elements of the enum Page_Format.

Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.
Protected Member Functions

• Fl_Paged_Device ()
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

• 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

Friends

• class Fl_Copy_Surface
• class Fl_Image_Surface

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

• static Fl_Surface_Device * surface ()
  The current drawing surface.

9.93.1 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.

9.93.2 Member Enumeration Documentation

9.93.2.1 Page_Format

enum Fl_Paged_Device::Page_Format
Possible page formats.
All paper formats with pre-defined width and height.

Enumerator

<table>
<thead>
<tr>
<th>A0</th>
<th>A0 format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A4</td>
<td>A4 format.</td>
</tr>
<tr>
<td>LETTER</td>
<td>Letter format.</td>
</tr>
</tbody>
</table>

9.93.2.2 Page_Layout

enum Fl_Paged_Device::Page_Layout
Possible page layouts.

Enumerator

<table>
<thead>
<tr>
<th>PORTRAIT</th>
<th>Portrait orientation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANDSCAPE</td>
<td>Landscape orientation.</td>
</tr>
</tbody>
</table>
9.93.3 Member Function Documentation

9.93.3.1 class_name()

```cpp
const char * Fl_Paged_Device::class_name ( ) [inline], [virtual]
```

Returns the name of the class of this object. Use of the `class_name()` function is discouraged because it will be removed from future FLTK versions. The class of an instance of an `Fl_Device` subclass can be checked with code such as:

```cpp
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```


9.93.3.2 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`, `Fl_System_Printer`, and `Fl_Printer`.

9.93.3.3 end_page()

```cpp
int Fl_Paged_Device::end_page ( void ) [virtual]
```

To be called at the end of each page.

Returns

- 0 if OK, non-zero if any error.

Reimplemented in `Fl_PostScript_File_Device`, `Fl_System_Printer`, and `Fl_Printer`.

9.93.3.4 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><code>out</code></th>
<th><code>left</code></th>
<th>If non-null, *left is set to the left margin size.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>top</code></td>
<td>If non-null, *top is set to the top margin size.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>right</code></td>
<td>If non-null, *right is set to the right margin size.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>bottom</code></td>
<td>If non-null, *bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented in `Fl_PostScript_File_Device`, `Fl_System_Printer`, and `Fl_Printer`.

9.93.3.5 origin() [1/2]

```cpp
void Fl_Paged_Device::origin ( 
```
Computes the page coordinates of the current origin of graphics functions.

Parameters

| in   | x          | If non-null, →x is set to the horizontal page offset of graphics origin. |
|      | y          | Same as above, vertically.                                     |

Reimplemented in Fl_PostScript_File_Device, Fl_System_Printer, and Fl_Printer.

9.93.3.6 origin() [2/2]

void Fl_Paged_Device::origin {
    int x,
    int y ) [virtual]
Sets the position in page coordinates of the origin of graphics functions.
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 page printable area.
Origin() calls are not affected by rotate() calls. Successive origin() calls don’t combine their effects.

Parameters

| in   | x          | Horizontal position in page coordinates of the desired origin of graphics functions. |
|      | y          | Same as above, vertically.                                     |

Reimplemented in Fl_PostScript_File_Device, Fl_System_Printer, and Fl_Printer.

9.93.3.7 print_widget()

void Fl_Paged_Device::print_widget {
    Fl_Widget * widget,
    int delta_x = 0,
    int delta_y = 0 ) [virtual]
Draws the widget on the printed page.
The widget's position on the printed page 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

| in   | widget    | Any FLTK widget (e.g., standard, custom, window). |
|      | delta_x   | Optional horizontal offset for positioning the widget relatively to the current origin of graphics functions. |
|      | delta_y   | Same as above, vertically. |

Reimplemented in Fl_Printer.

9.93.3.8 print_window()

void Fl_Paged_Device::print_window {
    Fl_Window * win,
    int x_offset = 0,
    int y_offset = 0 )
Prints a window with its title bar and frame if any.
x_offset and y_offset are optional coordinates of where to position the window top left. Equivalent to print_widget() 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 printed window.

### 9.93.3.9 print_window_part()

```cpp
void Fl_Paged_Device::print_window_part (Fl_Window * win, int x, int y, int w, int h, int delta_x = 0, int delta_y = 0) [virtual]
```

Prints a rectangular part of an on-screen window.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td>The window from where to capture.</td>
</tr>
<tr>
<td>x</td>
<td>The rectangle left</td>
</tr>
<tr>
<td>y</td>
<td>The rectangle top</td>
</tr>
<tr>
<td>w</td>
<td>The rectangle width</td>
</tr>
<tr>
<td>h</td>
<td>The rectangle height</td>
</tr>
<tr>
<td>delta_x</td>
<td>Optional horizontal offset from current graphics origin where to print the captured rectangle.</td>
</tr>
<tr>
<td>delta_y</td>
<td>As above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented in Fl_Printer.

### 9.93.3.10 printable_rect()

```cpp
int Fl_Paged_Device::printable_rect (int * w, int * h) [virtual]
```

Computes the width and height of the printable area of the page. Values are in the same unit as that used by FLTK drawing functions, are unchanged by calls to origin(), but are changed by scale() calls. Values account for the user-selected paper type and print orientation.

**Returns**

0 if OK, non-zero if any error

Reimplemented in Fl_PostScript_File_Device, Fl_System_Printer, and Fl_Printer.

### 9.93.3.11 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>angle</td>
<td>Rotation angle in counter-clockwise degrees.</td>
</tr>
</tbody>
</table>

Reimplemented in Fl_PostScript_File_Device, Fl_System_Printer, and Fl_Printer.
9.93.3.12 scale()

```cpp
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**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scale_x</code></td>
<td>Horizontal dimensions of plot are multiplied by this quantity.</td>
</tr>
<tr>
<td><code>scale_y</code></td>
<td>Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code>. Thus, <code>scale(factor);</code> is equivalent to <code>scale(factor, factor);</code></td>
</tr>
</tbody>
</table>

Reimplemented in `Fl_PostScript_File_Device`, `Fl_System_Printer`, and `Fl_Printer`.

9.93.3.13 start_job()

```cpp
int Fl_Paged_Device::start_job ( 
    int pagecount,
    int * frompage = NULL,
    int * topage = NULL ) [virtual]
```

Starts a print job.

**Parameters**

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pagecount</code></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><code>frompage</code></td>
<td>if non-null, <code>frompage</code> is set to the first page the user wants printed</td>
</tr>
<tr>
<td><code>topage</code></td>
<td>if non-null, <code>topage</code> is set to the last page the user wants printed</td>
</tr>
</tbody>
</table>

**Returns**

0 if OK, non-zero if any error


9.93.3.14 start_page()

```cpp
int Fl_Paged_Device::start_page ( 
    void ) [virtual]
```

Starts 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.

**Returns**

0 if OK, non-zero if any error

Reimplemented in `Fl_PostScript_File_Device`, `Fl_System_Printer`, and `Fl_Printer`.

9.93.3.15 translate()

```cpp
void Fl_Paged_Device::translate ( 
    int x,
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation. This function is only useful after a `rotate()` call. Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.

Reimplemented in `Fl_PostScript_File_Device`, `Fl_System_Printer`, and `Fl_Printer`.
9.93.3.16 untranslate()

```c
void Fl_Paged_Device::untranslate (
    void ) [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented in Fl_PostScript_File_Device, Fl_System_Printer, and Fl_Printer.
The documentation for this class was generated from the following files:

- Fl_Paged_Device.H
- Fl_Paged_Device.cxx

### 9.94 Fl_Pixmap Class Reference

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap) images, including transparency.

```c
#include <Fl_Pixmap.H>
```

Inheritance diagram for Fl_Pixmap:

```
Fl_Pixmap
   Fl_Image
      Fl_GIF Image Fl_XPM Image
```

#### Public Member Functions

- virtual void `color_average (Fl_Color c, float i)`

  The `color_average()` method averages the colors in the image with the FLTK color value `c`.

- Fl_Image * `copy ()`

  The `copy()` method creates a copy of the specified image.

- virtual Fl_Image * `copy (int W, int H)`

- virtual void `desaturate ()`

  The `desaturate()` method converts an image to grayscale.

- void `draw (int X, int Y)`

- virtual void `draw (int X, int Y, int W, int H, int cx=0, int cy=0)`

  Draws the image 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.

- virtual void `label (Fl_Menu_Item *m)`

  The `label()` methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void `label (Fl_Widget *w)`

  The `label()` methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void `uncache ()`

  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_Img

- **Fl_Img** ∗ **copy ()**
  The copy() method creates a copy of the specified image.

- **int count () const**
  The count() method returns the number of data values associated with the image.

- **int d () const**
  Returns the current image depth.

- **const char ∗ const ∗ data () const**
  Returns a pointer to the current image data array.

- **void draw (int X, int Y)**
  Draws the image.

- **int fail ()**
  Returns a value that is not 0 if there is currently no image available.

- **Fl_Img (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 height in pixels.

- **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.

- **int w () const**
  Returns the current image width in pixels.

- **virtual ~Fl_Img ()**
  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_Img

- **void d (int D)**
  Sets the current image depth.

- **void data (const char ∗ const ∗ p, int c)**
  Sets the current array 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.

- **void h (int H)**
  Sets the current image height in pixels.

- **void ld (int LD)**
  Sets the current line data size in bytes.

- **void w (int W)**
  Sets the current image width in pixels.
9.94 Fl_Pixmap Class Reference

Friends

- class Fl_GDI_Graphics_Driver
- class Fl_GDI_Printer_Graphics_Driver
- class Fl_Quartz_Graphics_Driver
- class Fl_Xlib_Graphics_Driver

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- 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 Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_NO_IMAGE = -1

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)

9.94.1 Detailed Description

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap ) images, including transparency.

9.94.2 Constructor & Destructor Documentation

9.94.2.1 Fl_Pixmap() [1/4]

Fl_Pixmap::Fl_Pixmap (char *const * D) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

9.94.2.2 Fl_Pixmap() [2/4]

Fl_Pixmap::Fl_Pixmap (uchar *const * D) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

9.94.2.3 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.
9.94.2.4 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.

9.94.3 Member Function Documentation

9.94.3.1 color_average()

void Fl_Pixmap::color_average (Fl_Color c, float i) [virtual]
The color_average() method averages the colors in the image with the FLTK color value c.
The i 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 before changes are applied, to avoid modifying the original image.
Reimplemented from Fl_Image.

9.94.3.2 copy()

Fl_Image ∗Fl_Pixmap::copy (int W, int H) [virtual]
The copy() method creates a copy of the specified image.
If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in
the case of Fl_Shared_Image, released) when you are done with it.
Reimplemented from Fl_Image.

9.94.3.3 desaturate()

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 before changes are applied, to avoid modifying the original image.
Reimplemented from Fl_Image.

9.94.3.4 draw()

void Fl_Pixmap::draw (int X, int Y, int W, int H, int cx = 0, int cy = 0) [virtual]
Draws the image 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.

9.94.3.5 label() [1/2]

void Fl_Pixmap::label (Fl_Menu_Item ∗m) [virtual]
The label() methods are an obsolete way to set the image attribute of a widget or menu item.
Use the `image()` or `deimage()` methods of the `Fl_Widget` and `Fl_Menu_Item` classes instead.
Reimplemented from `Fl_Image`.

**9.94.3.6 label() [2/2]**

```cpp
void FlPixmap::label (Fl_Widget *widget) [virtual]
```

The `label()` methods are an obsolete way to set the image attribute of a widget or menu item. Use the `image()` or `deimage()` methods of the `Fl_Widget` and `Fl_Menu_Item` classes instead.
Reimplemented from `Fl_Image`.

**9.94.3.7 uncache()**

```cpp
void FlPixmap::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.hxx`

**9.95 Fl_Plugin Class Reference**

`Fl_Plugin` allows link-time and run-time integration of binary modules.

```
#include <Fl_Plugin.H>
```

Inheritance diagram for `Fl_Plugin`:

```
Fl_Plugin
 |   |   |
|   |   |
|   |   |
|   |   |
|   |   |
|   FL_Device_Plugin
```

**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.

**9.95.1 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 initialisation 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++) {
```
9.95.2 Constructor & Destructor Documentation

9.95.2.1 Fl_Plugin()

Fl_Plugin::Fl_Plugin (
    const char * klass,
    const char * name )

Create a plugin.

Parameters

| in | klass | plugins are grouped in classes |
|    | name  | every plugin should have a unique name |

The documentation for this class was generated from the following files:

- Fl_Plugin.H
- Fl_Preferences.cxx

9.96 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:

```
Fl_Preferences
   ▼
   |
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 deleteAllEntries ()**

Generated by Doxygen
Delete all entries.

- **char deleteAllGroups ()**
  Delete all groups.

- **char deleteEntry (const char *entry)**
  Deletes a single name/value pair.

- **char deleteGroup (const char *group)**
  Deletes a group.

- **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 entryExists (const char *key)**
  Returns non-zero if an entry with this name exists.

- **Fl_Preferences (const char *path, const char *vendor, const char *application)**
  Use this constructor to create or read a preferences 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 (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 name/value pairs and child groups.

- **void flush ()**
  Writes all preferences to disk.

- **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 maxSize)**
  Reads an entry from the group.

- **char getUserdataPath (char *path, int pathlen)**
  Creates a path that is related to the preferences 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 groupExists (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 *filepath, 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 const char * newUUID ()**
  
  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 { SYSTEM =0 , USER }**
  
  Define the scope of the preferences.
Protected Attributes inherited from Fl_Preferences

- Node * node
- RootNode * rootNode

9.96.1 Detailed Description

Fl_Plugin_Manager manages link-time and run-time plugin binaries.

See also

Fl_Plugin

9.96.2 Constructor & Destructor Documentation

9.96.2.1 ~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.

9.96.3 Member Function Documentation

9.96.3.1 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.

9.96.3.2 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.

9.96.3.3 removePlugin()

void 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

9.97 Fl_PNG_Image Class Reference

The Fl_PNG_Image class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

#include <Fl_PNG_Image.H>

Inheritance diagram for Fl_PNG_Image:
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 void color_average (Fl_Color c, float i)
  
  The color_average() method averages the colors in the image with the FLTK color value c.

- **Fl_RGB_Image** ∗copy ()

- virtual **Fl_RGB_Image** ∗copy (int W, int H)
  
  The copy() method creates a copy of the specified image.

- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)

- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws the image 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 W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.

- virtual void label (Fl_Menu_Item ∗m)

  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void label (Fl_Widget ∗w)

  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void uncache ()

  If the image has been cached for display, delete the cache data.

- virtual ~**Fl_RGB_Image** ()

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from **Fl_Image**

- **Fl_RGB_Image** ∗copy ()

- int count () const

- int d () const
  
  Returns the current image depth.

- const char ∗const ∗data () const
  
  Returns a pointer to the current image data array.

- void draw (int X, int Y)

  Draws the image.
• int fail()
  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 height in pixels.
• 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.
• int w() const
  Returns the current image width in pixels.
• 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_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).

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_NO_IMAGE = -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 array 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.
• void h(int H)
Sets the current image height in pixels.

- void ld (int LD)

Sets the current line data size in bytes.

- void w (int W)

Sets the current image width in pixels.

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)

9.97.1 Detailed Description

The Fl_PNG_Image class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files. The class loads colormapped and full-color images and handles color- and alpha-based transparency.

9.97.2 Constructor & Destructor Documentation

9.97.2.1 Fl_PNG_Image() [1/2]

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_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

| in  | filename | Name of PNG file to read |

9.97.2.2 Fl_PNG_Image() [2/2]

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

| name_png | A name given to this image or NULL |
| buffer   | Pointer to the start of the PNG image in memory |
| maxsize  | Size in bytes of the memory buffer containing the PNG image |

The documentation for this class was generated from the following files:

- Fl_PNG_Image.H
- Fl_PNG_Image.cxx
9.98  **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
  |       |
  V       V
FL_RGB_Image
  |       |
  V       V
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 void **color_average (FL_Color c, float i)**
  
  The color_average() method averages the colors in the image with the FLTK color value c.

- **FL_Image * copy ()**

  The copy() method creates a copy of the specified image.

- virtual void **desaturate ()**

  The desaturate() method converts an image to grayscale.

- void **draw (int X, int Y)**

- virtual void **draw (int X, int Y, int W, int H, int cx=0, int cy=0)**

  Draws the image 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 W, int H, int D=3, int LD=0)**

  The constructor creates a new image from the specified data.

- virtual void **label (FL_Menu_Item *m)**

  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void **label (FL_Widget *w)**

  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void **uncache ()**

  If the image has been cached for display, delete the cache data.

- virtual ~**FL_RGB_Image ()**

  The destructor frees all memory and server resources that are used by the image.

**Public Member Functions inherited from FL_Image**

- **FL_Image * copy ()**

  The copy() method creates a copy of the specified image.

- int **count () const**

  The count() method returns the number of data values associated with the image.

- int **d () const**

  Returns the current image depth.
• const char *const * data () const
  Returns a pointer to the current image data array.
• void draw (int X, int Y)
  Draws the image.
• int fail ()
  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 height in pixels.
• 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.
• int w () const
  Returns the current image width in pixels.
• 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_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).

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_NO_IMAGE = -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 array 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.
- void h (int H)
  Sets the current image height in pixels.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the current image width in bytes.

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)

9.98.1 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.

9.98.2 Constructor & Destructor Documentation

9.98.2.1 Fl_PNM_Image()

Fl_PNM_Image::Fl_PNM_Image (const char * name )

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. The documentation for this class was generated from the following files:

- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

9.99 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)
  
  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 **xmininimum** () const
  
  Gets the X axis minimum.

- void **xmininimum** (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 **xvalue** (double)
  
  Sets the X axis coordinate.

- void **ybounds** (double, double)
  
  Sets the Y axis bounds.

- double **ymaximum** () const
  
  Gets the Y axis maximum.

- void **ymaximum** (double a)
  
  Same as ybounds(ymininimum(), a)

- double **yminimum** () const
  
  Gets the Y axis minimum.

- void **yminimum** (double a)
  
  Same as ybounds(a, ymaximum())

- void **ystep** (double a)
  
  Sets the stepping value for the Y axis.

- double **yvalue** () const
  
  Gets the Y axis coordinate.

- int **yvalue** (double)
  
  Sets the Y axis coordinate.

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
9.99 Fl_Positioner Class Reference

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 Fl_Gl_Window * as_gl_window ()
  
  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_Window * as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- 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, 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)
  
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.

- **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.

- **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_ACCESTS_MENU flag is set, 0 otherwise.

• void **∗** **user_data** () const
  
  Gets 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** ()
  
  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** ()
  
  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 ()
  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_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 ∗cb, void ∗d)
  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 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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

9.99.1 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 9.24 Fl_Positioner

9.99.2 Constructor & Destructor Documentation

9.99.2.1 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.

9.99.3 Member Function Documentation

9.99.3.1 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:

```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
9.99.3.2 handle()

```
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.

**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_Positioner.H
- Fl_Positioner.cxx

9.100 Fl_PostScript_File_Device Class Reference

To send graphical output to a PostScript file.

```
#include <Fl_PostScript.H>
```

Inheritance diagram for Fl_PostScript_File_Device:

```
Fl_Device

Fl_Surface_Device

Fl_Paged_Device

Fl_PostScript_File_Device

Fl_PostScript_Printer
```

**Public Member Functions**

- const char * class_name ()
  
  Returns the name of the class of this object.
- void end_job (void)
  
  To be called at the end of a print job.
- int end_page (void)
  
  To be called at the end of each page.
• **Fl_PostScript_File_Device ()**
  
  *The constructor.*

• **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 origin (int *x, int *y)**
  
  *Computes the page coordinates of the current origin of graphics functions.*

• **void origin (int x, int y)**
  
  *Sets the position in page coordinates of the origin of graphics functions.*

• **int printable_rect (int *w, int *h)**
  
  *Computes the width and height of the printable area of the page.*

• **void rotate (float angle)**
  
  *Rotates the graphics operations relatively to paper.*

• **void scale (float scale_x, float scale_y=0.)**
  
  *Changes the scaling of page coordinates.*

• **int start_job (FILE *ps_output, int pagecount, 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 start_job (int pagecount, 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.*

• **int start_job (int pagecount, int *from, int *to)**
  
  *Don't use with this class.*

• **int start_page (void)**
  
  *Starts a new printed page.*

• **void translate (int x, int y)**
  
  *Translates the current graphics origin accounting for the current rotation.*

• **void untranslate (void)**
  
  *Undoes the effect of a previous translate() call.*

• **~Fl_PostScript_File_Device ()**
  
  *The destructor.*

### Public Member Functions inherited from Fl_Paged_Device

• **virtual void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)**
  
  *Draws the widget on the printed page.*

• **void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)**
  
  *Prints a window with its title bar and frame if any.*

• **virtual void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  
  *Prints a rectangular part of an on-screen window.*

• **virtual ~Fl_Paged_Device ()**
  
  *The destructor.*

### Public Member Functions inherited from Fl_Surface_Device

• **const char * class_name ()**
  
  *Returns the name of the class of this object.*

• **FL_Graphics_Driver * driver ()**
  
  *Returns the graphics driver of this drawing surface.*

• **void driver (FL_Graphics_Driver *graphics_driver)**
  
  *Sets the graphics driver of this drawing surface.*

• **virtual void set_current (void)**
  
  *Make this surface the current drawing surface.*

• **virtual ~Fl_Surface_Device ()**
  
  *The destructor.*
Public Member Functions inherited from Fl_Device

• virtual ~Fl_Device ()
  
  Virtual destructor.

Static Public Attributes

• static const char *class_id = "Fl_PostScript_File_Device"
• static const char *file_chooser_title = "Select a .ps file"
  
  Label of the PostScript file chooser window.

Static Public Attributes inherited from Fl_Paged_Device

• static const char *class_id = "Fl_Paged_Device"
• static const page_format page_formats [NO_PAGE_FORMATS]
  
  width, height and name of all elements of the enum Page_Format.

Static Public Attributes inherited from Fl_Surface_Device

• static const char *class_id = "Fl_Surface_Device"

Static Public Attributes inherited from Fl_Device

• static const char *class_id = "Fl_Device"
  
  A string that identifies each subclass of Fl_Device.

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_Surface_Device

• 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 * surface ()
  
  The current drawing surface.

Protected Attributes inherited from Fl_Paged_Device

- int x_offset
  
  horizontal offset to the origin of graphics coordinates
- int y_offset
  
  vertical offset to the origin of graphics coordinates

9.100.1 Detailed Description

To send graphical output to a PostScript file.

This class is used exactly as the Fl_Printer class except for the start_job() call, two variants of which are usable and allow to specify what page format and layout are desired.

9.100.2 Member Function Documentation

9.100.2.1 class_name()

const char * Fl_PostScript_File_Device::class_name ( ) [inline], [virtual]

Returns the name of the class of this object.

Use of the class_name() function is discouraged because it will be removed from future FLTK versions.

The class of an instance of an Fl_Device subclass can be checked with code such as:

```cpp
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```

Reimplemented from Fl_Paged_Device.

Reimplemented in Fl_PostScript_Printer.

9.100.2.2 end_job()

void Fl_PostScript_File_Device::end_job ( void ) [virtual]

To be called at the end of a print job.

Reimplemented from Fl_Paged_Device.

9.100.2.3 end_page()

int Fl_PostScript_File_Device::end_page ( void ) [virtual]

To be called at the end of each page.

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

9.100.2.4 margins()

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>out</th>
<th>left</th>
<th>If non-null, left is set to the left margin size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>top</td>
<td>If non-null, top is set to the top margin size.</td>
</tr>
<tr>
<td>out</td>
<td>right</td>
<td>If non-null, right is set to the right margin size.</td>
</tr>
<tr>
<td>out</td>
<td>bottom</td>
<td>If non-null, bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

### 9.100.2.5 origin() [1/2]

```cpp
void Fl_PostScript_File_Device::origin ( 
    int * x, 
    int * y ) [virtual]
```

Computes the page coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x</th>
<th>If non-null, x is set to the horizontal page offset of graphics origin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

### 9.100.2.6 origin() [2/2]

```cpp
void Fl_PostScript_File_Device::origin ( 
    int x, 
    int y ) [virtual]
```

Sets the position in page coordinates of the origin of graphics functions. 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 page printable area. Origin() calls are not affected by `rotate()` calls. Successive `origin()` calls don’t combine their effects.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>Horizontal position in page coordinates of the desired origin of graphics functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

### 9.100.2.7 printable_rect()

```cpp
int Fl_PostScript_File_Device::printable_rect ( 
    int * w, 
    int * h ) [virtual]
```

Computes the width and height of the printable area of the page. Values are in the same unit that used by FLTK drawing functions, are unchanged by calls to `origin()`, but are changed by `scale()` calls. Values account for the user-selected paper type and print orientation.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

### 9.100.2.8 rotate()

```cpp
void Fl_PostScript_File_Device::rotate ( 
    float angle ) [virtual]
```

Generated by Doxygen
Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive \texttt{rotate()} calls don’t combine their effects.

**Parameters**

\begin{verbatim}
| angle | Rotation angle in counter-clockwise degrees. |
\end{verbatim}

Reimplemented from \texttt{Fl_Paged_Device}.

**9.100.2.9 scale()**

\begin{verbatim}
void Fl_PostScript_File_Device::scale {
    float scale_x,
    float scale_y = 0. } [virtual]
\end{verbatim}

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a \texttt{scale()} call, do a \texttt{printable_rect()} call to get the new dimensions of the printable page area. Successive \texttt{scale()} calls don’t combine their effects.

**Parameters**

\begin{verbatim}
| 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); |
\end{verbatim}

Reimplemented from \texttt{Fl_Paged_Device}.

**9.100.2.10 start_job() [1/3]**

\begin{verbatim}
int Fl_PostScript_File_Device::start_job (  
    FILE * ps_output,  
    int pagecount,  
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,  
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
\end{verbatim}

Begins the session where all graphics requests will go to \texttt{FILE} pointer.

**Parameters**

\begin{verbatim}
| ps_output | A writable \texttt{FILE} pointer that will receive PostScript output and that should not be closed until after \texttt{end_job()} has been called. |
| pagecount | The total number of pages to be created. Use 0 if this number is unknown when this function is called. |
| format | Desired page format. |
| layout | Desired page layout. |
\end{verbatim}

Returns

always 0.

**9.100.2.11 start_job() [2/3]**

\begin{verbatim}
int Fl_PostScript_File_Device::start_job (  
    int pagecount,  
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,  
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
\end{verbatim}

Begins the session where all graphics requests will go to a local PostScript file.

---

Generated by Doxygen
Opens a file dialog entitled with Fl_PostScript_File_Device::file_chooser_title to select an output PostScript file.

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.

9.100.2.12 start_job() [3/3]

```cpp
int Fl_PostScript_File_Device::start_job (  
    int pagecount,  
    int * from,  
    int * to ) [virtual]
```

Don't use with this class.
Reimplemented from Fl_Paged_Device.
Reimplemented in Fl_PostScript_Printer.

9.100.2.13 start_page()

```cpp
int Fl_PostScript_File_Device::start_page (  
    void ) [virtual]
```

Starts 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.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

9.100.2.14 translate()

```cpp
void Fl_PostScript_File_Device::translate (  
    int x,  
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.
This function is only useful after a rotate() call. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Paged_Device.

9.100.2.15 untranslate()

```cpp
void Fl_PostScript_File_Device::untranslate (  
    void ) [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Paged_Device.
The documentation for this class was generated from the following files:

- Fl_PostScript.H
- Fl_PostScript.cxx

Generated by Doxygen
9.101 Fl_PostScript_Graphics_Driver Class Reference

PostScript graphical backend.
#include <Fl_PostScript.H>
Inheritance diagram for Fl_PostScript_Graphics_Driver:

```
Fl_Device
  |
  V
Fl_Graphics_Driver
  |
  V
Fl_PostScript_Graphics_Driver
```

Public Member Functions

- void arc (double x, double y, double r, double start, double a)
  
  see fl_arc(double x, double y, double r, double start, double end).
- void arc (int x, int y, int w, int h, double a1, double a2)
  
  see fl_arc(int x, int y, int w, int h, double a1, double a2).
- void begin_complex_polygon ()
  
  see fl_begin_complex_polygon().
- void begin_line ()
  
  see fl_begin_line().
- void begin_loop ()
  
  see fl_begin_loop().
- void begin_points ()
  
  see fl_begin_points().
- void begin_polygon ()
  
  see fl_begin_polygon().
- void circle (double x, double y, double r)
  
  see fl_circle(double x, double y, double r).
- const char ∗ class_name ()
  
  Returns the name of the class of this object.
- int clip_box (int x, int y, int w, int h, int &x, int &y , int &w, int &h)
  
  see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).
- int locale_printf (const char ∗ format,...)
  
  Shields output PostScript data from modifications of the current locale.
- void color (Fl_Color c)
  
  see fl_color(Fl_Color c).
- void color (uchar r, uchar g, uchar b)
  
  see fl_color(uchar r, uchar g, uchar b).
- void curve (double x, double y, double x1, double y1, double x2, double y2, double x3, double y3)
  
  see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).
- int descent ()
  
  see fl_descent().
- void draw (const char ∗s, int nBytes, int x, int y)
  
  see fl_draw(const char ∗ str, int n, int x, int y).
- void draw (Fl_Bitmap ∗bitmap, int XP, int YP, int WP, int HP, int cx, int cy)
  
  Draws an Fl_Bitmap object to the device.
- void draw (Fl_Pixmap ∗pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  
  Draws an Fl_Pixmap object to the device.
• void **draw** (Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_RGB_Image object to the device.
• void **draw** (int angle, const char *str, int n, int x, int y)
  see fl_draw(int angle, const char *str, int n, int x, int y).
• void **draw_image** (const uchar *d, int x, int y, int w, int h, int delta=3, int ldelt=0)
  see fl_draw_image(const uchar* d, int X, int Y, int W, int H, int D, int L).
• void **draw_image** (Fl_Draw_Image_Cb call, void *data, int x, int y, int w, int h, int delta=3)
  see fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
• void **draw_image_mono** (const uchar *d, int x, int y, int w, int h, int delta=1, int ldelt=0)
  see fl_draw_image_mono(const uchar* d, int X, int Y, int W, int H, int D).
• void **draw_scaled** (Fl_Image *img, int XP, int YP, int WP, int HP)
  Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.
• void **end_complex_polygon** ()
  see fl_end_complex_polygon().
• void **end_line** ()
  see fl_end_line().
• void **end_loop** ()
  see fl_end_loop().
• void **end_points** ()
  see fl_end_points().
• void **end_polygon** ()
  see fl_end_polygon().
• **Fl_PostScript_Graphics_Driver** ()
  The constructor.
• void **font** (int face, int size)
  see fl_font(Fl_Font face, Fl_Fontsize size).
• void **gap** ()
  see fl_gap().
• int **height** ()
  see fl_height().
• void **line** (int x1, int y1, int x2, int y2)
  see fl_line(int x, int y, int x1, int y1).
• void **line** (int x1, int y1, int x2, int y2, int x3, int y3)
  see fl_line(int x, int y, int x1, int y1, int x2, int y2).
• void **line_style** (int style, int width=0, char *dashes=0)
  see fl_line_style(int style, int width, char* dashes).
• void **loop** (int x0, int y0, int x1, int y1, int x2, int y2)
  see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2).
• void **loop** (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).
• int **not_clipped** (int x, int y, int w, int h)
  see fl_not_clipped(int x, int y, int w, int h).
• void **pie** (int x, int y, int w, int h, double a1, double a2)
  see fl_pie(int x, int y, int w, int h, double a1, double a2).
• void **point** (int x, int y)
  see fl_point(int x, int y).
• void **polygon** (int x0, int y0, int x1, int y1, int x2, int y2)
  see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2).
• void **polygon** (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`.

- void `pop_clip()`
  
  see `fl_pop_clip()`.

- void `push_clip(int x, int y, int w, int h)`
  
  see `fl_push_clip(int x, int y, int w, int h)`.

- void `push_no_clip()`
  
  see `fl_push_no_clip()`.

- void `rect(int x, int y, int w, int h)`
  
  see `fl_rect(int x, int y, int w, int h)`.

- void `rectf(int x, int y, int w, int h)`
  
  see `fl_rectf(int x, int y, int w, int h)`.

- void `rtl_draw(const char *s, int n, int x, int y)`
  
  see `fl_rtl_draw(const char *str, int n, int x, int y)`.

- void `text_extents(const char *c, int n, int &dx, int &dy, int &w, int &h)`
  
  see `fl_text_extents(const char *, int n, int &dx, int &dy, int &w, int &h)`.

- void transformed_vertex(double x, double y)
  
  see `fl_transformed_vertex(double x, double y)`.

- void `vertex(double x, double y)`
  
  see `fl_vertex(double x, double y)`.

- double `width(const char *, int)`
  
  see `fl_width(const char *, int)`.

- double `width(unsigned int u)`
  
  see `fl_width(unsigned int)`.

- void `xyline(int x, int y, int x1)`
  
  see `fl_xyline(int x, int y, int x1)`.

- void `xyline(int x, int y, int x1, int y2)`
  
  see `fl_xyline(int x, int y, int x1, int y2)`.

- void `yline(int x, int y, int y1)`
  
  see `fl_yxline(int x, int y, int y1)`.

- void `xyline(int x, int y, int y1, int x2)`
  
  see `fl_yxline(int x, int y, int y1, int x2)`.

- void `xyline(int x, int y, int y1, int x2, int y2)`
  
  see `fl_yxline(int x, int y, int y1, int x2, int y2)`.

- void `~Fl_PostScript_Graphics_Driver()`
  
  The destructor.

### Public Member Functions inherited from `Fl_Graphics_Driver`

- `Fl_Color color()`
  
  see `fl_color(void)`.

- `Fl_Font font()`
  
  see `fl_font(void)`.

- `Fl_Font_Descriptor * font_descriptor()`
  
  Returns a pointer to the current `Fl_Font_Descriptor` for the graphics driver.

- void `font_descriptor(Fl_Font_Descriptor *d)`
  
  Sets the current `Fl_Font_Descriptor` for the graphics driver.

- `Fl_Fontsize size()`
  
  see `fl_fontsize()`.

- virtual `~Fl_Graphics_Driver()`
  
  The destructor.
Public Member Functions inherited from Fl_Device

- virtual ~Fl_Device ()
  Virtual destructor.

Static Public Attributes

- static const char * class_id = "Fl_PostScript_Graphics_Driver"

Static Public Attributes inherited from Fl_Graphics_Driver

- static const char * class_id = "Fl_Graphics_Driver"

Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.

Additional Inherited Members

Protected Member Functions inherited from Fl_Graphics_Driver

- Fl_Region clip_region ()
  see fl_clip_region().
- void clip_region (Fl_Region r)
  see fl_clip_region(Fl_Region r).
- virtual void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  see fl_copy_offscreen()
- Fl_Graphics_Driver ()
  The constructor.
- void mult_matrix (double a, double b, double c, double d, double x, double y)
  see fl_mult_matrix(double a, double b, double c, double d, double x, double y).
- void pop_matrix ()
  see fl_pop_matrix().
- void push_matrix ()
  see fl_push_matrix().
- void restore_clip ()
  see fl_restore_clip().
- void rotate (double d)
  see fl_rotate(double d).
- void scale (double x)
  see fl_scale(double x).
- void scale (double x, double y)
  see fl_scale(double x, double y).
- double transform_dx (double x, double y)
  see fl_transform_dx(double x, double y).
- double transform_dy (double x, double y)
  see fl_transform_dy(double x, double y).
- double transform_x (double x, double y)
  see fl_transform_x(double x, double y).
- double transform_y (double x, double y)
  see fl_transform_y(double x, double y).
- void translate (double x, double y)
  see fl_translate(double x, double y).
Protected Attributes inherited from Fl_Graphics_Driver

- matrix * fl_matrix

Points to the current coordinate transformation matrix.

9.101.1 Detailed Description

PostScript graphical backend.
PostScript text uses vectorial fonts when using the FLTK standard fonts and the latin alphabet or a few other characters listed in the following table. 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.

<table>
<thead>
<tr>
<th>Char</th>
<th>Code-point</th>
<th>Name</th>
<th>Char</th>
<th>Code-point</th>
<th>Name</th>
<th>Char</th>
<th>Code-point</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ƒ</td>
<td>U+0192</td>
<td>florin</td>
<td>′</td>
<td>U+201A</td>
<td>quotesingle™</td>
<td>†</td>
<td>U+2020</td>
<td>dagger</td>
</tr>
<tr>
<td>^</td>
<td>U+02C6</td>
<td>circumflex</td>
<td>&quot;</td>
<td>U+201D</td>
<td>quotedblleft</td>
<td>‡</td>
<td>U+2021</td>
<td>daggerdbl</td>
</tr>
<tr>
<td>ˇ</td>
<td>U+02C7</td>
<td>caron</td>
<td>&quot;</td>
<td>U+201E</td>
<td>quotedblbase</td>
<td>˛</td>
<td>U+2022</td>
<td>bullet</td>
</tr>
<tr>
<td>˘</td>
<td>U+02D8</td>
<td>breve</td>
<td>†</td>
<td>U+2020</td>
<td>dagger</td>
<td>˚</td>
<td>U+2023</td>
<td>bullet</td>
</tr>
<tr>
<td>™</td>
<td>U+2024</td>
<td>dotaccent</td>
<td>†</td>
<td>U+2021</td>
<td>dagger</td>
<td>†</td>
<td>U+2024</td>
<td>fraction</td>
</tr>
</tbody>
</table>

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.

9.101.2 Member Function Documentation

9.101.2.1 arc() [1/2]

```cpp
void Fl_PostScript_Graphics_Driver::arc {
    double x,
    double y,
    double r,
    double start,
    double end  } [virtual]
```

see fl_arc(double x, double y, double r, double start, double end).
Reimplemented from Fl_Graphics_Driver.

9.101.2.2 arc() [2/2]

```cpp
void Fl_PostScript_Graphics_Driver::arc {
    int x,
    int y,
    int w,
    int h,
    double a1,
    double a2 ) [virtual]
```
see fl_arc(int x, int y, int w, int h, double a1, double a2).
Reimplemented from Fl_Graphics_Driver.

9.101.2.3  begin_complex_polygon()

void Fl_PostScript_Graphics_Driver::begin_complex_polygon() [inline], [virtual]
see fl_begin_complex_polygon().
Reimplemented from Fl_Graphics_Driver.

9.101.2.4  begin_line()

void Fl_PostScript_Graphics_Driver::begin_line() [virtual]
see fl_begin_line().
Reimplemented from Fl_Graphics_Driver.

9.101.2.5  begin_loop()

void Fl_PostScript_Graphics_Driver::begin_loop() [virtual]
see fl_begin_loop().
Reimplemented from Fl_Graphics_Driver.

9.101.2.6  begin_points()

void Fl_PostScript_Graphics_Driver::begin_points() [virtual]
see fl_begin_points().
Reimplemented from Fl_Graphics_Driver.

9.101.2.7  begin_polygon()

void Fl_PostScript_Graphics_Driver::begin_polygon() [virtual]
see fl_begin_polygon().
Reimplemented from Fl_Graphics_Driver.

9.101.2.8  circle()

void Fl_PostScript_Graphics_Driver::circle(double x, double y, double r) [virtual]
see fl_circle(double x, double y, double r).
Reimplemented from Fl_Graphics_Driver.

9.101.2.9  class_name()

const char* Fl_PostScript_Graphics_Driver::class_name() [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if (instance->class_name() == Fl_Printer::class_id) {...}
Reimplemented from Fl_Graphics_Driver.

9.101.2.10  clip_box()

int Fl_PostScript_Graphics_Driver::clip_box(int x, int y, int w, int h, int & X, int & Y,
Reimplemented from Fl_Graphics_Driver.


```c
int & W,
int & H) [virtual]
```

see `fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.11 `locale_printf()`

```c
int Fl_PostScript_Graphics_Driver::locale_printf (const char * format, ...
```

Shields output PostScript data from modifications of the current locale.
It typically avoids PostScript errors caused if the current locale uses comma instead of dot as "decimal point".

**Parameters**

- `format` directives controlling output PostScript data

**Returns**

- value returned by `vfprintf()` call

### 9.101.2.12 `color()` [1/2]

```c
void Fl_PostScript_Graphics_Driver::color (Fl_Color c) [virtual]
```

see `fl_color(Fl_Color c)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.13 `color()` [2/2]

```c
void Fl_PostScript_Graphics_Driver::color (uchar r,
uchar g,
uchar b) [virtual]
```

see `fl_color(uchar r, uchar g, uchar b)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.14 `curve()`

```c
void Fl_PostScript_Graphics_Driver::curve (double X0,
double Y0,
double X1,
double Y1,
double X2,
double Y2,
double X3,
double Y3) [virtual]
```

see `fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.15 `descent()`

```c
int Fl_PostScript_Graphics_Driver::descent () [virtual]
```

see `fl_descent()`.
Reimplemented from `Fl_Graphics_Driver`.
9.101.2.16 draw() [1/5]

```cpp
void Fl_PostScript_Graphics_Driver::draw (  
    const char * str,  
    int n,  
    int x,  
    int y ) [inline], [virtual]
```

see \_\_fl_draw(const char *str, int n, int x, int y).

Reimplemented from Fl_Graphics_Driver.

9.101.2.17 draw() [2/5]

```cpp
void Fl_PostScript_Graphics_Driver::draw (  
    Fl_Bitmap * bm,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy ) [virtual]
```

Draws an Fl_Bitmap object to the device.

Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.

Reimplemented from Fl_Graphics_Driver.

9.101.2.18 draw() [3/5]

```cpp
void Fl_PostScript_Graphics_Driver::draw (  
    Fl_Pixmap * pxm,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy ) [virtual]
```

Draws an Fl_Pixmap object to the device.

Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.

Reimplemented from Fl_Graphics_Driver.

9.101.2.19 draw() [4/5]

```cpp
void Fl_PostScript_Graphics_Driver::draw (  
    Fl_RGB_Images * rgb,  
    int XP,  
    int YP,  
    int WP,  
    int HP,  
    int cx,  
    int cy ) [virtual]
```

Draws an Fl_RGB_Images object to the device.

Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.

Reimplemented from Fl_Graphics_Driver.

9.101.2.20 draw() [5/5]

```cpp
void Fl_PostScript_Graphics_Driver::draw (  
    int angle,
```
const char * str,
  int n,
  int x,
  int y) [virtual]

see fl_draw(int angle, const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.101.2.21 draw_image() [1/2]

void Fl_PostScript_Graphics_Driver::draw_image (  
  const uchar * buf,
  int X,
  int Y,
  int W,
  int H,
  int D = 3,
  int L = 0) [virtual]

see fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.101.2.22 draw_image() [2/2]

void Fl_PostScript_Graphics_Driver::draw_image (  
  Fl_Draw_Image_Cb cb,
  void * data,
  int X,
  int Y,
  int W,
  int H,
  int D = 3) [virtual]

see fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D).
Reimplemented from Fl_Graphics_Driver.

9.101.2.23 draw_image_mono() [1/2]

void Fl_PostScript_Graphics_Driver::draw_image_mono (  
  const uchar * buf,
  int X,
  int Y,
  int W,
  int H,
  int D = 1,
  int L = 0) [virtual]

see fl_draw_image_mono(const uchar * buf, int X, int Y, int W, int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.101.2.24 draw_image_mono() [2/2]

void Fl_PostScript_Graphics_Driver::draw_image_mono (  
  Fl_Draw_Image_Cb cb,
  void * data,
  int X,
  int Y,
  int W,
  int H,
  int D = 1) [virtual]

see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D).
Reimplemented from Fl_Graphics_Driver.
9.101.2.25 draw_scaled()

int Fl_PostScript_Graphics_Driver::draw_scaled (  
    Fl_Image * img,  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]

Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.

Returns
    zero when the graphics driver doesn't implement scaled drawing, non-zero if it does implement it.

Reimplemented from Fl_Graphics_Driver.

9.101.2.26 end_complex_polygon()

void Fl_PostScript_Graphics_Driver::end_complex_polygon ( ) [inline], [virtual]
see fl_end_complex_polygon().
Reimplemented from Fl_Graphics_Driver.

9.101.2.27 end_line()

void Fl_PostScript_Graphics_Driver::end_line ( ) [virtual]
see fl_end_line().
Reimplemented from Fl_Graphics_Driver.

9.101.2.28 end_loop()

void Fl_PostScript_Graphics_Driver::end_loop ( ) [virtual]
see fl_end_loop().
Reimplemented from Fl_Graphics_Driver.

9.101.2.29 end_points()

void Fl_PostScript_Graphics_Driver::end_points ( ) [virtual]
see fl_end_points().
Reimplemented from Fl_Graphics_Driver.

9.101.2.30 end_polygon()

void Fl_PostScript_Graphics_Driver::end_polygon ( ) [virtual]
see fl_end_polygon().
Reimplemented from Fl_Graphics_Driver.

9.101.2.31 font()

void Fl_PostScript_Graphics_Driver::font (  
    int face,  
    int fsize ) [virtual]
see fl_font(Fl_Font face, Fl_Fontsize size).
Reimplemented from Fl_Graphics_Driver.

9.101.2.32 gap()

void Fl_PostScript_Graphics_Driver::gap ( ) [inline], [virtual]
see fl_gap().
Reimplemented from Fl_Graphics_Driver.
9.101.2.33  height()

int Fl_PostScript_Graphics_Driver::height ( ) [virtual]
see fl_height().
Reimplemented from Fl_Graphics_Driver.

9.101.2.34  line() [1/2]

void Fl_PostScript_Graphics_Driver::line (  
    int x,
    int y,
    int x1,
    int y1  ) [virtual]
see fl_line(int x, int y, int x1, int y1).
Reimplemented from Fl_Graphics_Driver.

9.101.2.35  line() [2/2]

void Fl_PostScript_Graphics_Driver::line (  
    int x,
    int y,
    int x1,
    int y1,
    int x2,
    int y2  ) [virtual]
see fl_line(int x, int y, int x1, int y1, int x2, int y2).
Reimplemented from Fl_Graphics_Driver.

9.101.2.36  line_style()

void Fl_PostScript_Graphics_Driver::line_style (  
    int style,
    int width = 0,
    char * dashes = 0  ) [virtual]
see fl_line_style(int style, int width, char* dashes).
Reimplemented from Fl_Graphics_Driver.

9.101.2.37  loop() [1/2]

void Fl_PostScript_Graphics_Driver::loop (  
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2  ) [virtual]
see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2).
Reimplemented from Fl_Graphics_Driver.

9.101.2.38  loop() [2/2]

void Fl_PostScript_Graphics_Driver::loop (  
    int x0,
    int y0,
    int x1,
    int y1,
    int x2,
    int y2,
    int x3,
    int y3  ) [virtual]
see `fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`. Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.39 `not_clipped()`

```cpp
template <class int Fl_PostScript_Graphics_Driver::not_clipped (int x, int y, int w, int h) [virtual]
```

see `fl_not_clipped(int x, int y, int w, int h)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.40 `pie()`

```cpp
void Fl_PostScript_Graphics_Driver::pie (int x, int y, int w, int h, double a1, double a2) [virtual]
```

see `fl_pie(int x, int y, int w, int h, double a1, double a2)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.41 `point()`

```cpp
void Fl_PostScript_Graphics_Driver::point (int x, int y) [virtual]
```

see `fl_point(int x, int y)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.42 `polygon()` [1/2]

```cpp
void Fl_PostScript_Graphics_Driver::polygon (int x0, int y0, int x1, int y1, int x2, int y2) [virtual]
```

see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2)`.
Reimplemented from `Fl_Graphics_Driver`.

### 9.101.2.43 `polygon()` [2/2]

```cpp
void Fl_PostScript_Graphics_Driver::polygon (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3) [virtual]
```

see `fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)`.
Reimplemented from `Fl_Graphics_Driver`.
9.101.2.44  pop_clip()

void Fl_PostScript_Graphics_Driver::pop_clip ( ) [virtual]
see fl_pop_clip().
Reimplemented from Fl_Graphics_Driver.

9.101.2.45  push_clip()

void Fl_PostScript_Graphics_Driver::push_clip ( int x, int y, int w, int h ) [virtual]
see fl_push_clip(int x, int y, int w, int h).
Reimplemented from Fl_Graphics_Driver.

9.101.2.46  push_no_clip()

void Fl_PostScript_Graphics_Driver::push_no_clip ( ) [virtual]
see fl_push_no_clip().
Reimplemented from Fl_Graphics_Driver.

9.101.2.47  rect()

void Fl_PostScript_Graphics_Driver::rect ( int x, int y, int w, int h ) [virtual]
see fl_rect(int x, int y, int w, int h).
Reimplemented from Fl_Graphics_Driver.

9.101.2.48  rectf()

void Fl_PostScript_Graphics_Driver::rectf ( int x, int y, int w, int h ) [virtual]
see fl_rectf(int x, int y, int w, int h).
Reimplemented from Fl_Graphics_Driver.

9.101.2.49  rtl_draw()

void Fl_PostScript_Graphics_Driver::rtl_draw ( const char * str, int n, int x, int y ) [virtual]
see fl_rtl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.101.2.50  text_extents()

void Fl_PostScript_Graphics_Driver::text_extents ( const char * t, int n, int & dx, int & dy,
int & w,
int & h ) [virtual]
see fl_text_extents(const char *, int n, int & dx, int & dy, int & w, int & h).
Reimplemented from Fl_Graphics_Driver.

9.101.2.51 transformed_vertex()

void Fl_PostScript_Graphics_Driver::transformed_vertex ( double xf,
double yf ) [virtual]
see fl_transformed_vertex(double xf, double yf).
Reimplemented from Fl_Graphics_Driver.

9.101.2.52 vertex()

void Fl_PostScript_Graphics_Driver::vertex ( double x,
double y ) [virtual]
see fl_vertex(double x, double y).
Reimplemented from Fl_Graphics_Driver.

9.101.2.53 width() [1/2]

double Fl_PostScript_Graphics_Driver::width ( const char * str,
int n ) [virtual]
see fl_width(const char *str, int n).
Reimplemented from Fl_Graphics_Driver.

9.101.2.54 width() [2/2]

double Fl_PostScript_Graphics_Driver::width ( unsigned int c ) [virtual]
see fl_width(unsigned int n).
Reimplemented from Fl_Graphics_Driver.

9.101.2.55 xyline() [1/3]

void Fl_PostScript_Graphics_Driver::xyline ( int x,
int y,
int x1 ) [virtual]
see fl_xyline(int x, int y, int x1).
Reimplemented from Fl_Graphics_Driver.

9.101.2.56 xyline() [2/3]

void Fl_PostScript_Graphics_Driver::xyline ( int x,
int y,
int x1,
int y2 ) [virtual]
see fl_xyline(int x, int y, int x1, int y2).
Reimplemented from Fl_Graphics_Driver.

9.101.2.57 xyline() [3/3]

void Fl_PostScript_Graphics_Driver::xyline ( int x,
int y,
int x1,
int y2,
int x3 ) [virtual]

see fl_xyline(int x, int y, int x1, int y2, int x3).
Reimplemented from Fl_Graphics_Driver.

9.101.258 yxline() [1/3]

void Fl_PostScript_Graphics_Driver::yxline (  
    int x,
    int y,
    int y1 ) [virtual]

see fl_yxline(int x, int y, int y1).
Reimplemented from Fl_Graphics_Driver.

9.101.259 yxline() [2/3]

void Fl_PostScript_Graphics_Driver::yxline (  
    int x,
    int y,
    int y1,
    int x2 ) [virtual]

see fl_yxline(int x, int y, int y1, int x2).
Reimplemented from Fl_Graphics_Driver.

9.101.260 yxline() [3/3]

void Fl_PostScript_Graphics_Driver::yxline (  
    int x,
    int y,
    int y1,
    int x2,
    int y3 ) [virtual]

see fl_yxline(int x, int y, int y1, int x2, int y3).
Reimplemented from Fl_Graphics_Driver.

The documentation for this class was generated from the following files:

- Fl_PoScript.H
- Fl_PoScript.cxx

9.102 Fl_PostScript_Printer Class Reference

Print support under Unix/Linux.
#include <Fl_Printer.H>

Inheritance diagram for Fl_PostScript_Printer:
Public Member Functions

- **const char ∗ class_name ()**
  Returns the name of the class of this object.
- **int start_job (int pages, int ∗firstpage=NULL, int ∗lastpage=NULL)**
  Starts a print job.

Public Member Functions inherited from Fl_PostScript_File_Device

- **void end_job (void)**
  To be called at the end of a print job.
- **int end_page (void)**
  To be called at the end of each page.
- **Fl_PostScript_File_Device ()**
  The constructor.
- **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 origin (int ∗x, int ∗y)**
  Computes the page coordinates of the current origin of graphics functions.
- **void origin (int x, int y)**
  Sets the position in page coordinates of the origin of graphics functions.
- **int printable_rect (int ∗w, int ∗h)**
  Computes the width and height of the printable area of the page.
- **void rotate (float angle)**
  Rotates the graphics operations relatively to paper.
- **void scale (float scale_x, float scale_y=0.)**
  Changes the scaling of page coordinates.
- **int start_job (FILE ∗ps_output, int pagecount, 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 start_job (int pagecount, 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.
- **int start_page (void)**
  Starts a new printed page.
- **void translate (int x, int y)**
  Translates the current graphics origin accounting for the current rotation.
- **void untranslate (void)**
  Undoes the effect of a previous translate() call.
- **∼Fl_PostScript_File_Device ()**
  The destructor.
Public Member Functions inherited from Fl_Paged_Device

- virtual void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  
  Draws the widget on the printed page.

- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Prints a window with its title bar and frame if any.

- virtual void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)
  
  Prints a rectangular part of an on-screen window.

- virtual ~Fl_Paged_Device ()
  
  The destructor.

Public Member Functions inherited from Fl_Surface_Device

- const char *class_name ()
  
  Returns the name of the class of this object.

- Fl_Graphics_Driver *driver ()
  
  Returns the graphics driver of this drawing surface.

- void driver (Fl_Graphics_Driver *graphics_driver)
  
  Sets the graphics driver of this drawing surface.

- virtual void set_current (void)
  
  Make this surface the current drawing surface.

- virtual ~Fl_Surface_Device ()
  
  The destructor.

Public Member Functions inherited from Fl_Device

- virtual ~Fl_Device ()
  
  Virtual destructor.

Static Public Attributes

- static const char *class_id = Fl_Printer::class_id

Static Public Attributes inherited from Fl_PostScript_File_Device

- static const char *class_id = "Fl_PostScript_File_Device"

- static const char *file_chooser_title = "Select a .ps file"

  Label of the PostScript file chooser window.

Static Public Attributes inherited from Fl_Paged_Device

- static const char *class_id = "Fl_Paged_Device"

- static const page_format page_formats [NO_PAGE_FORMATS]

  width, height and name of all elements of the enum Page_Format.

Static Public Attributes inherited from Fl_Surface_Device

- static const char *class_id = "Fl_Surface_Device"

Static Public Attributes inherited from Fl_Device

- static const char *class_id = "Fl_Device"

  A string that identifies each subclass of Fl_Device.
Protected Member Functions

- FL_PostScript_Printer (void)
  
  The constructor.

Protected Member Functions inherited from FL_PostScript_File_Device

- 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_Surface_Device

- FL_Surface_Device (FL_Graphics_Driver *graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

Friends

- class FL_Printer

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 * surface ()
  
  The current drawing surface.

Protected Attributes inherited from FL_Paged_Device

- int x_offset
  
  horizontal offset to the origin of graphics coordinates

- int y_offset
  
  vertical offset to the origin of graphics coordinates
9.102.1 Detailed Description

Print support under Unix/Linux.
Class Fl_PostScript_Printer is implemented only on the Unix/Linux platform. It has no public constructor. Use Fl_Printer instead that is cross-platform and has the same API.

9.102.2 Member Function Documentation

9.102.2.1 class_name()

const char * Fl_PostScript_Printer::class_name ( ) [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
Reimplemented from Fl_PostScript_File_Device.

9.102.2.2 start_job()

int Fl_PostScript_Printer::start_job ( 
   int pages,
   int * firstpage = NULL,
   int * lastpage = NULL ) [virtual]
Starts a print job.
Reimplemented from Fl_PostScript_File_Device.
The documentation for this class was generated from the following files:

  • Fl_Printer.H
  • Fl_PostScript.cxx
  • Fl_Printer.cxx

9.103 Fl_Preferences Class Reference

Fl_Preferences provides methods to store user settings between application starts.
#include <Fl_Preferences.H>
Inheritance diagram for Fl_Preferences:

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 { SYSTEM =0 , USER }
  Define the scope of the preferences.
Public Member Functions

- `char clear ()`
  "Delete all groups and all entries."

- `char deleteAllEntries ()`
  "Delete all entries."

- `char deleteAllGroups ()`
  "Delete all groups."

- `char deleteEntry (const char * entry)`
  "Deletes a single name/value pair."

- `char deleteGroup (const char * group)`
  "Deletes a group."

- `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 entryExists (const char * key)`
  "Returns non-zero if an entry with this name exists."

- `FL_PREFERENCES (const char * path, const char * vendor, const char * application)`
  "Use this constructor to create or read a preferences 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 name/value pairs and child groups."

- `void flush ()`
  "Writes all preferences to disk."

- `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 maxSize)`
  "Reads an entry from the group."

- `char getUserdataPath (char *&path, int pathlen)`
  "Creates a path that is related to the preferences 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 groupExists (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 const char * newUUID ()
  Returns a UUID as generated by the system.
• static char remove (ID id_)
  Remove the group with this ID from a database.

Protected Attributes

• Node * node
• RootNode * rootNode

Friends

• class Node
• class RootNode
9.103.1 Detailed Description

**Fl_Preferences** provides methods to store user settings between application starts. It is similar to the Registry on WIN32 and Preferences on MacOS, and provides a simple configuration mechanism for UNIX. **Fl_Preferences** uses a hierarchy to store data. It bundles similar data into groups and manages entries into those groups as name/value pairs.

Preferences are stored in text files that can be edited manually. The file format is easy to read and relatively forgiving. Preferences 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.

Entries can be of any length. However, the size of each preferences file should be kept small for performance reasons. One application can have multiple preferences files. Extensive binary data however should be stored in separate files: see `getUserdataPath()`.

**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.

9.103.2 Member Typedef Documentation

9.103.2.1 ID

```c
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.

9.103.3 Member Enumeration Documentation

9.103.3.1 Root

```c
enum Fl_Preferences::Root
```

Define the scope of the preferences.

**Enumerator**

| SYSTEM | Preferences are used system-wide. |
| USER   | Preferences apply only to the current user. |

9.103.4 Constructor & Destructor Documentation

9.103.4.1 **Fl_Preferences()** [1/7]

```c
Fl_Preferences::Fl_Preferences (  
    Root root,  
    const char * vendor,  
    const char * application )
```

The constructor creates a group that manages name/value pairs and child groups. Groups are ready for reading and writing at any time. The root argument is either **Fl_Preferences::USER** or **Fl_Preferences::SYSTEM**.

This constructor creates the base instance for all following entries and reads existing databases into memory. The vendor argument is a unique text string identifying the development team or vendor of an application. A domain name or an EMail address are great unique names, e.g. "researchATmatthiasm.com" or "fltk.org". The application argument can be the working title or final name of your application. Both vendor and application must be valid relative UNIX pathnames and may contain '/'s to create deeper file structures.

A set of Preferences marked “run-time” exists exactly one per application and only as long as the application runs.

---

Generated by Doxygen
It can be used as a database for volatile information. FLTK uses it to register plugins at run-time.

### Parameters

- **root**: can be `USER` or `SYSTEM` for user specific or system wide preferences
- **vendor**: unique text describing the company or author of this file
- **application**: unique text describing the application

#### 9.103.2 Fl_Preferences() [2/7]

```cpp
Fl_Preferences::Fl_Preferences (const char * path,
                               const char * vendor,
                               const char * application)
```

Use this constructor to create or read a preferences 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` must contain the full file name.

### Parameters

- **path**: path to the directory that contains the preferences file
- **vendor**: unique text describing the company or author of this file
- **application**: unique text describing the application

#### 9.103.3 Fl_Preferences() [3/7]

```cpp
Fl_Preferences::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

- **parent**: reference object for the new group
- **group**: name of the group to access (may contain `/`s)

#### 9.103.4 Fl_Preferences() [4/7]

```cpp
Fl_Preferences::Fl_Preferences (Fl_Preferences * parent,
                               const char * group)
```

Create or access a group of preferences using a name.

### Parameters

- **parent**: the parameter parent is a pointer to the parent group. Parent may be NULL. It then refers to an application internal database 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.
- **group**: a group name that is used as a key into the database
See also

Fl_Preferences( Fl_Preferences&, const char *group )

9.103.4.5  Fl_Preferences() [5/7]

Fl_Preferences::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

| in   | parent                  | reference object for the new group          |
|      | in groupIndex           | zero based index into child groups         |

9.103.4.6  Fl_Preferences() [6/7]

Fl_Preferences::Fl_Preferences ( Fl_Preferences * parent, int groupIndex )

See also

Fl_Preferences( Fl_Preferences&, int groupIndex )

9.103.4.7  Fl_Preferences() [7/7]

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.

9.103.4.8  ~Fl_Preferences()

Fl_Preferences::~Fl_Preferences ( ) [virtual]
The destructor removes allocated resources.
When used on the base preferences group, the destructor flushes all changes to the preferences file and deletes all
internal databases.
The destructor does not remove any data from the database. It merely deletes your reference to the database.

9.103.5  Member Function Documentation

9.103.5.1  deleteEntry()

char Fl_Preferences::deleteEntry ( const char * key )

Deletes a single name/value pair.
This function removes the entry key from the database.

Parameters

| in  | key          | name of entry to delete          |
9.103.5.2 deleteGroup()

```cpp
char Fl_Preferences::deleteGroup ( 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

---

9.103.5.3 entries()

```cpp
int Fl_Preferences::entries ( )
```

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

---

9.103.5.4 entry()

```cpp
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().

Parameters

- **index** number indexing the requested entry

Returns

pointer to value cstring

---

9.103.5.5 entryExists()

```cpp
char Fl_Preferences::entryExists ( const char * key )
```

Returns non-zero if an entry with this name exists.

Parameters

- **key** name of entry that is searched for
Returns

0 if entry was not found

9.103.5.6 flush()

void Fl_Preferences::flush ( )

Writes all preferences to disk.
This function works only with the base preferences group. This function is rarely used as deleting the base preferences flushes automatically.

9.103.5.7 get() [1/7]

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

9.103.5.8 get() [2/7]

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

9.103.5.9  get() [3/7]

```cpp
cchar 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

9.103.5.10  get() [4/7]

```cpp
cchar 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

9.103.5.11  get() [5/7]

```cpp
cchar 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

9.103.5.12 get() [6/7]

```cpp
cchar 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>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>
</tbody>
</table>

Returns

0 if the default value was used

9.103.5.13 get() [7/7]

```cpp
cchar Fl_Preferences::get (  
    const char * key,  
    void * & data,  
    const void * defaultValue,  
    int defaultSize,  
    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.

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</td>
<td>maxSize</td>
<td>maximum length of value</td>
</tr>
</tbody>
</table>
Returns

0 if the default value was used

Todo maxSize should receive the number of bytes that were read.

9.103.5.14 getUserdataPath()

char Fl_Preferences::getUserdataPath (  
  char * path,  
  int pathlen )

Creates a path that is related to the preferences 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.

Example:

```cpp
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( path );
```

.. creates the preferences database in (MS Windows):

```
c:/Documents and Settings/matt/Application Data/matthiasm.com/test.prefs
```

.. and returns the userdata path:

```
c:/Documents and Settings/matt/Application Data/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

0 if path was not created or pathname can't fit into buffer

9.103.5.15 group()

const char * Fl_Preferences::group (  
  int num_group )

Returns the name of the Nth (num_group) group.

There is no guaranteed order of group names. The index must be within the range given by groups().

Parameters

| in   | num_group | number indexing the requested group |

Returns

‘C’ string pointer to the group name

9.103.5.16 groupExists()

char Fl_Preferences::groupExists (  
  const char * key )

Returns non-zero if a group with this name exists. Group names are relative to the Preferences node and can contain a path. "." describes the current node, "./" describes the topmost node. By preceding a groupname with a "./", its path becomes relative to the topmost node.

Parameters

| in | key | name of group that is searched for |
9.103.5.17  groups()

int Fl_Preferences::groups ( )
Returns the number of groups that are contained within a group.
Returns
0 for no groups at all

9.103.5.18  newUUID()

const char * Fl_Preferences::newUUID ( ) [static]
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!

9.103.5.19  set() [1/7]

char Fl_Preferences::set ( const char * key, const char * text )
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 preferences 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

9.103.5.20  set() [2/7]

char 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 preferences 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

9.103.5.21 set() [3/7]

char 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 preferences file.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns
0 if setting the value failed

9.103.5.22 set() [4/7]

char 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 preferences file.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>set this entry to value</td>
</tr>
<tr>
<td></td>
<td>precision</td>
<td>number of decimal digits to represent value</td>
</tr>
</tbody>
</table>

Returns
0 if setting the value failed

9.103.5.23 set() [5/7]

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 preferences file.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>
Returns
0 if setting the value failed

**9.103.24 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 preferences 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

**9.103.25 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 preferences 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

**9.103.26 size()**

```cpp
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

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

9.104 Fl_Printer Class Reference

OS-independent print support.

#include <Fl_Printer.H>

Inheritance diagram for Fl_Printer:

```
Fl_Device
    |
    Fl_Surface_Device
    |
    Fl_Paged_Device
    |
Fl_Printer
```

Public Member Functions

- **const char ∗ class_name ()**
  Returns the name of the class of this object.
- **Fl_Graphics_Driver ∗ driver (void)**
- **void end_job (void)**
  To be called at the end of a print job.
- **int end_page (void)**
  To be called at the end of each page.
- **Fl_Printer ∗ (void)**
  The constructor.
- **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 origin (int ∗ x, int ∗ y)**
  Computes the page coordinates of the current origin of graphics functions.
- **void origin (int x, int y)**
  Sets the position in page coordinates of the origin of graphics functions.
- **void print_widget (Fl_Widget ∗ widget, int delta_x=0, int delta_y=0)**
  Draws the widget on the printed page.
- **void print_window_part (Fl_Window ∗ win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  Prints a rectangular part of an on-screen window.
- **int printable_rect (int ∗ w, int ∗ h)**
  Computes the width and height of the printable area of the page.
- **void rotate (float angle)**
  Rotates the graphics operations relatively to paper.
- **void scale (float scale_x, float scale_y=0.)**
  Changes the scaling of page coordinates.
- **void set_current (void)**
Make this surface the current drawing surface.
- int start_job (int pagecount, int *frompage=NULL, int *topage=NULL)
  Starts a print job.
- int start_page (void)
  Starts a new printed page.
- void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.
- void untranslate (void)
  Undoes the effect of a previous translate() call.
- ~Fl_Printer (void)
  The destructor.

Public Member Functions inherited from Fl_Paged_Device
- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Prints a window with its title bar and frame if any.
- virtual ~Fl_Paged_Device ()
  The destructor.

Public Member Functions inherited from Fl_Surface_Device
- const char *class_name ()
  Returns the name of the class of this object.
- Fl_Graphics_Driver *driver ()
  Returns the graphics driver of this drawing surface.
- void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual ~Fl_Surface_Device ()
  The destructor.

Public Member Functions inherited from Fl_Device
- virtual ~Fl_Device ()
  Virtual destructor.

Static Public Attributes
- static const char *class_id = "Fl_Printer"

These attributes are effective under the Xlib 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:"

Generated by Doxygen
[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"
[this text may be customized at run-time]
- 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 char * class_id = "Fl_Paged_Device"
- static const page_format page_formats [NO_PAGE_FORMATS]
  width, height and name of all elements of the enum Page_Format.

Static Public Attributes inherited from Fl_Surface_Device

- static const char * class_id = "Fl_Surface_Device"

Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.

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 * surface ()
  The current drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

• Fl_Paged_Device ()
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

• 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_Paged_Device

• int x_offset
  horizontal offset to the origin of graphics coordinates
• int y_offset
  vertical offset to the origin of graphics coordinates

9.104.1 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 start_job(), start_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_Graphics_Driver *graphics_driver = ... // a graphics driver
Fl_Printer *printer = new Fl_Printer();
if (printer->start_job(1) == 0) {
  printer->start_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;
```

Platform specifics

• Unix/Linux platforms: 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::start_job() call. This opens a "Printer Properties" window where it's possible to select the adequate paper size. Finally press the "Save" button thereafter to assign the chosen paper size to the chosen printer for this and all further print operations.

Class Fl_RGB_Image prints but loses its transparency if it has one. See class Fl_PostScript_Graphics_Driver for a description of how UTF-8 strings appear in print. 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:
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.

- MSWindows platform: Transparent Fl_RGB_Image’s don’t print with exact transparency on most printers. Fl_RGB_Image’s don’t rotate() well. A workaround is to use the print_window_part() call.
- Mac OS X platform: all graphics requests print as on display.

9.104.2 Member Function Documentation

9.104.2.1 class_name()

const char * Fl_Printer::class_name ( ) [inline], [virtual]

Returns the name of the class of this object.

Use of the class_name() function is discouraged because it will be removed from future FLTK versions.

The class of an instance of an Fl_Device subclass can be checked with code such as:

```c
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
```

Reimplemented from Fl_Paged_Device.

9.104.2.2 end_job()

void Fl_Printer::end_job ( ) [virtual]

To be called at the end of a print job.

Reimplemented from Fl_Paged_Device.

9.104.2.3 end_page()

int Fl_Printer::end_page ( ) [virtual]

To be called at the end of each page.

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

9.104.2.4 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>int</td>
<td>If non-null, *left is set to the left margin size.</td>
</tr>
<tr>
<td>top</td>
<td>int</td>
<td>If non-null, *top is set to the top margin size.</td>
</tr>
<tr>
<td>right</td>
<td>int</td>
<td>If non-null, *right is set to the right margin size.</td>
</tr>
<tr>
<td>bottom</td>
<td>int</td>
<td>If non-null, *bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.
9.104.2.5 origin () [1/2]

```c
void Fl_Printer::origin (  
    int * x,  
    int * y ) [virtual]
```

Computes the page coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x</th>
<th>If non-null, *x is set to the horizontal page offset of graphics origin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

9.104.2.6 origin () [2/2]

```c
void Fl_Printer::origin (  
    int x,  
    int y ) [virtual]
```

Sets the position in page coordinates of the origin of graphics functions.

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 page printable area. Origin() calls are not affected by rotate() calls. Successive origin() calls don't combine their effects.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>Horizontal position in page coordinates of the desired origin of graphics functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

9.104.2.7 print_widget ()

```c
void Fl_Printer::print_widget (  
    Fl_Widget * widget,  
    int delta_x = 0,  
    int delta_y = 0 ) [virtual]
```

Draws the widget on the printed page.

The widget's position on the printed page 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</td>
<td>Optional horizontal offset for positioning the widget relatively to the current origin of graphics functions.</td>
</tr>
<tr>
<td>in</td>
<td>delta_y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

9.104.2.8 print_window_part ()

```c
void Fl_Printer::print_window_part (  
    Fl_Window * win,  
    int x,  
    int y,  
```
Prints a rectangular part of an on-screen window.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>The window from where to capture.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>The rectangle left</td>
</tr>
<tr>
<td><code>y</code></td>
<td>The rectangle top</td>
</tr>
<tr>
<td><code>w</code></td>
<td>The rectangle width</td>
</tr>
<tr>
<td><code>h</code></td>
<td>The rectangle height</td>
</tr>
<tr>
<td><code>delta_x</code></td>
<td>Optional horizontal offset from current graphics origin where to print the captured rectangle.</td>
</tr>
<tr>
<td><code>delta_y</code></td>
<td>As above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Paged_Device`.

### 9.104.2.9 printable_rect()

```c
int Fl_Printer::printable_rect ( int * w, int * h ) [virtual]
```

Computes the width and height of the printable area of the page.
Values are in the same unit as that used by FLTK drawing functions, are unchanged by calls to `origin()`, but are changed by `scale()` calls. Values account for the user-selected paper type and print orientation.

Returns

0 if OK, non-zero if any error

Reimplemented from `Fl_Paged_Device`.

### 9.104.2.10 rotate()

```c
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

<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 from `Fl_Paged_Device`.

### 9.104.2.11 scale()

```c
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.

### 9.104.2.12 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. Reimplemented from Fl_Surface_Device.

### 9.104.2.13 start_job()

```cpp
int Fl_Printer::start_job (
    int pagecount,
    int * frompage = NULL,
    int * topage = NULL  // [virtual]
)
```

Starts a print job.

Opens a platform-specific dialog window allowing the user to set several options including the desired printer and the page orientation. Optionally, the user can also select a range of pages to be printed. This range is returned to the caller that is in charge of sending only these pages for printing.

Parameters

| in | pagecount | the total number of pages of the job (or 0 if you don’t know the number of pages) |
| out | frompage | if non-null, *frompage is set to the first page the user wants printed |
| out | topage | if non-null, *topage is set to the last page the user wants printed |

Returns

0 if OK, non-zero if any error occurred or if the user cancelled the print request.

Reimplemented from Fl_Paged_Device.

### 9.104.2.14 start_page()

```cpp
int Fl_Printer::start_page ()
{
    void  // [virtual]
}
```

Starts 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.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

### 9.104.2.15 translate()

```cpp
void Fl_Printer::translate (
    int x,
    int y  // [virtual]
)
```

Translates the current graphics origin accounting for the current rotation.
This function is only useful after a rotate() call. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects. Reimplemented from Fl_Paged_Device.

9.104.2.16  untranslate()

void Fl_Printer::untranslate (  
    void                              ) [virtual]

Undoes the effect of a previous translate() call. Reimplemented from Fl_Paged_Device.

The documentation for this class was generated from the following files:

- Fl_Printer.H
- Fl_Printer.cxx

9.105 Fl_Progress Class Reference

Displays a progress bar for the user.
#include <Fl_Progress.H>

Inheritance diagram for Fl_Progress:

\[ \begin{array}{c}
\text{Fl_Widget} \\
\downarrow \\
\text{Fl_Progress}
\end{array} \]

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 Fl_Group * as_group ()
  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.

• 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, 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)
  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.
- **const Fl_Image *deimage () const**
  void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.
- **void do_callback ()**
  Calls the widget callback.
- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.
- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.
- **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.
- **const Fl_Image *image () const**
  void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.
- **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.
• 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.
• 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 (void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible () const**
  
  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 () 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 void draw ()
  
  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 ()
  
  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_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 ∗cb, void ∗d)
  
  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**

```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, COPY_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, COPY_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACENTS_MENU = 1<<19,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
```

flags possible values enumeration.

**9.105.1 Detailed Description**

Displays a progress bar for the user.

**9.105.2 Constructor & Destructor Documentation**

**9.105.2.1 Fl_Progress()**

```cpp
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.

**9.105.3 Member Function Documentation**

**9.105.3.1 draw()**

```cpp
void Fl_Progress::draw {  
    void ) [protected], [virtual]
```

Draws the progress bar. Implements `Fl_Widget`.

**9.105.3.2 maximum() [1/2]**

```cpp
float Fl_Progress::maximum ( ) const [inline]
```

Gets the maximum value in the progress widget.

**9.105.3.3 maximum() [2/2]**

```cpp
void Fl_Progress::maximum (  
    float v ) [inline]
```

Sets the maximum value in the progress widget.
9.105.3.4 minimum() [1/2]

float Fl_Progress::minimum ( ) const [inline]
Gets the minimum value in the progress widget.

9.105.3.5 minimum() [2/2]

void Fl_Progress::minimum ( float v ) [inline]
Sets the minimum value in the progress widget.

9.105.3.6 value() [1/2]

float Fl_Progress::value ( ) const [inline]
Gets the current value in the progress widget.

9.105.3.7 value() [2/2]

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.cxx

9.106 Fl_Quartz_Graphics_Driver Class Reference

The Mac OS X-specific graphics class.

#include <Fl_Device.H>
Inheritance diagram for Fl_Quartz_Graphics_Driver:

```
Fl_Device

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Quartz_Graphics_Driver</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- const char * class_name ()
  
  Returns the name of the class of this object.

- void color (Fl_Color c)
  
  see fl_color(Fl_Color c).

- void color (uchar r, uchar g, uchar b)
  
  see fl_color(uchar r, uchar g, uchar b).

- void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)

Generated by Doxygen
see \texttt{fl\_copy\_offscren()}

- int \texttt{descent()}
  see \texttt{fl\_descent()}

- void \texttt{draw(const char \ast str, int n, int x, int y)}
  see \texttt{fl\_draw(const char \ast str, int n, int x, int y)}

- void \texttt{draw(Fl\_Bitmap \ast pxm, int XP, int YP, int WP, int HP, int cx, int cy)}
  Draws an \texttt{Fl\_Bitmap} object to the device.

- void \texttt{draw(Fl\_Pixmap \ast pxm, int XP, int YP, int WP, int HP, int cx, int cy)}
  Draws an \texttt{Fl\_Pixmap} object to the device.

- void \texttt{draw(Fl\_RGB\_Image \ast img, int XP, int YP, int WP, int HP, int cx, int cy)}
  Draws an \texttt{Fl\_RGB\_Image} object to the device.

- void \texttt{draw(int angle, const char \ast str, int n, int x, int y)}
  see \texttt{fl\_draw(int angle, const char \ast str, int n, int x, int y)}

- void \texttt{draw_image(const uchar \ast buf, int X, int Y, int W, int H, int D=3, int L=0)}
  see \texttt{fl\_draw\_image(const uchar \ast buf, int X, int Y, int W, int H, int D, int L)}

- void \texttt{draw\_image(Fl\_Draw\_Image\_Cb cb, void \ast data, int X, int Y, int W, int H, int D=3)}
  see \texttt{fl\_draw\_image(Fl\_Draw\_Image\_Cb cb, void \ast data, int X, int Y, int W, int H, int D)}

- void \texttt{draw\_image\_mono(const uchar \ast buf, int X, int Y, int W, int H, int D=1, int L=0)}
  see \texttt{fl\_draw\_image\_mono(const uchar \ast buf, int X, int Y, int W, int H, int D, int L)}

- void \texttt{draw\_image\_mono(Fl\_Draw\_Image\_Cb cb, void \ast data, int X, int Y, int W, int H, int D=1)}
  see \texttt{fl\_draw\_image\_mono(Fl\_Draw\_Image\_Cb cb, void \ast data, int X, int Y, int W, int H, int D)}

- void \texttt{draw\_scaled(Fl\_Image \ast img, int XP, int YP, int WP, int HP)}
  Draws an \texttt{Fl\_Image} scaled to width \texttt{W} & height \texttt{H} with top-left corner at \texttt{X,Y}.

- void \texttt{font(Fl\_Font face, Fl\_Fontsize size)}
  see \texttt{fl\_font(Fl\_Font face, Fl\_Fontsize size)}

- int \texttt{height()}
  see \texttt{fl\_height()}

- void \texttt{rtl\_draw(const char \ast str, int n, int x, int y)}
  see \texttt{fl\_rtl\_draw(const char \ast str, int n, int x, int y)}

- void \texttt{text\_extents(const char \ast, int n, \&dx, \&dy, int \&w, int \&h)}
  see \texttt{fl\_text\_extents(const char \ast, int n, \&dx, \&dy, int \&w, int \&h)}

- double \texttt{width(const char \ast str, int n)}
  see \texttt{fl\_width(const char \ast str, int n)}

- double \texttt{width(unsigned int c)}
  see \texttt{fl\_width(unsigned int c)}

### Public Member Functions inherited from \texttt{FI\_Graphics\_Driver}

- \texttt{Fl\_Color color()}
  see \texttt{fl\_color(void)}

- \texttt{Fl\_Font font()}
  see \texttt{fl\_font(void)}

- \texttt{Fl\_Font\_Descriptor \ast font\_descriptor()}
  Returns a pointer to the current \texttt{Fl\_Font\_Descriptor} for the graphics driver.

- void \texttt{font\_descriptor(Fl\_Font\_Descriptor \ast d)}
  Sets the current \texttt{Fl\_Font\_Descriptor} for the graphics driver.

- \texttt{Fl\_Fontsize size()}
  see \texttt{fl\_size()}.

- virtual \texttt{\~Fl\_Graphics\_Driver()}
  The destructor.
Public Member Functions inherited from Fl_Device

- virtual \sim Fl_Device ()

Virtual destructor.

Static Public Attributes

- static const char * class_id = "Fl_Quartz_Graphics_Driver"

Static Public Attributes inherited from Fl_Graphics_Driver

- static const char * class_id = "Fl_Graphics_Driver"

Static Public Attributes inherited from Fl_Device

- static const char * class_id = "Fl_Device"

A string that identifies each subclass of Fl_Device.

Additional Inherited Members

Protected Member Functions inherited from Fl_Graphics_Driver

- virtual void arc (double x, double y, double r, double start, double end)

see fl_arc(double x, double y, double r, double start, double end).

- virtual void arc (int x, int y, int w, int h, double a1, double a2)

see fl_arc(int x, int y, int w, int h, double a1, double a2).

- virtual void begin_complex_polygon ()

see fl_begin_complex_polygon().

- virtual void begin_line ()

see fl_begin_line().

- virtual void begin_loop ()

see fl_begin_loop().

- virtual void begin_points ()

see fl_begin_points().

- virtual void begin_polygon ()

see fl_begin_polygon().

- virtual void circle (double x, double y, double r)

see fl_circle(double x, double y, double r).

- virtual int clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).

- Fl_Region clip_region ()

see fl_clip_region().

- void clip_region (Fl_Region r)

see fl_clip_region(Fl_Region r).

- virtual void curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).

- virtual void end_complex_polygon ()

see fl_end_complex_polygon().

- virtual void end_line ()

see fl_end_line().

- virtual void end_loop ()

see fl_end_loop().

- virtual void end_points ()

see fl_end_points().
• virtual void **end_polygon** ()
  
  **see fl_end_polygon().**

• **Fl_Graphics_Driver** ()
  
  *The constructor.*

• virtual void **gap** ()
  
  **see fl_gap().**

• virtual void **line** (int x, int y, int x1, int y1)
  
  **see fl_line(int x, int y, int x1, int y1).**

• virtual void **line** (int x, int y, int x1, int y1, int x2, int y2)
  
  **see fl_line(int x, int y, int x1, int y1, int x2, int y2).**

• virtual void **line_style** (int style, int width=0, char *dashes=0)
  
  **see fl_line_style(int style, int width, char * dashes).**

• virtual void **loop** (int x0, int y0, int x1, int y1, int x2, int y2)
  
  **see fl_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)
  
  **see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).**

• void **mult_matrix** (double a, double b, double c, double d, double x, double y)
  
  **see fl_mult_matrix(double a, double b, double c, double d, double x, double y).**

• virtual int **not_clipped** (int x, int y, int w, int h)
  
  **see fl_not_clipped(int x, int y, int w, int h).**

• virtual void **pie** (int x, int y, int w, int h, double a1, double a2)
  
  **see fl_pie(int x, int y, int w, int h, double a1, double a2).**

• virtual void **point** (int x, int y)
  
  **see fl_point(int x, int y).**

• virtual void **polygon** (int x0, int y0, int x1, int y1, int x2, int y2)
  
  **see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2).**

• virtual void **polygon** (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  
  **see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).**

• virtual void **pop_clip** ()
  
  **see fl_pop_clip().**

• void **pop_matrix** ()
  
  **see fl_pop_matrix().**

• virtual void **push_clip** (int x, int y, int w, int h)
  
  **see fl_push_clip(int x, int y, int w, int h).**

• void **push_matrix** ()
  
  **see fl_push_matrix().**

• virtual void **push_no_clip** ()
  
  **see fl_push_no_clip().**

• virtual void **rect** (int x, int y, int w, int h)
  
  **see fl_rect(int x, int y, int w, int h).**

• virtual void **rectf** (int x, int y, int w, int h)
  
  **see fl_rectf(int x, int y, int w, int h).**

• void **rotate** (double d)
  
  **see fl_rotate(double d).**

• void **scale** (double x)
  
  **see fl_scale(double x).**

• void **scale** (double x, double y)
  
  **see fl_scale(double x, double y).**

• double **transform_dx** (double x, double y)
• double transform_dx(double x, double y)
  see fl_transform_dx(double x, double y).
• double transform_dy(double x, double y)
  see fl_transform_dy(double x, double y).
• double transform_x(double x, double y)
  see fl_transform_x(double x, double y).
• double transform_y(double x, double y)
  see fl_transform_y(double x, double y).
• virtual void transformed_vertex(double xf, double yf)
  see fl_transformed_vertex(double xf, double yf).
• void translate(double x, double y)
  see fl_translate(double x, double y).
• virtual void vertex(double x, double y)
  see fl_vertex(double x, double y).
• virtual void xyline(int x, int y, int x1)
  see fl_xyline(int x, int y, int x1).
• virtual void xyline(int x, int y, int x1, int y2)
  see fl_xyline(int x, int y, int x1, int y2).
• virtual void xyline(int x, int y, int x1, int y2, int x3)
  see fl_xyline(int x, int y, int x1, int y2, int x3).
• virtual void yxline(int x, int y, int y1)
  see fl_yxline(int x, int y, int y1).
• virtual void yxline(int x, int y, int y1, int x2)
  see fl_yxline(int x, int y, int y1, int x2).
• virtual void yxline(int x, int y, int y1, int x2, int y3)
  see fl_yxline(int x, int y, int y1, int x2, int y3).

Protected Attributes inherited from Fl_Graphics_Driver

• matrix * fl_matrix
  Points to the current coordinate transformation matrix.

9.106.1 Detailed Description

The Mac OS X-specific graphics class.
This class is implemented only on the Mac OS X platform.

9.106.2 Member Function Documentation

9.106.2.1 class_name()

const char * Fl_Quartz_Graphics_Driver::class_name ( ) [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
Reimplemented from Fl_Graphics_Driver.

9.106.2.2 color() [1/2]

void Fl_Quartz_Graphics_Driver::color ( Fl_Color c ) [virtual]
see fl_color(Fl_Color c).
Reimplemented from Fl_Graphics_Driver.
9.106.2.3 color() [2/2]

void Fl_Quartz_Graphics_Driver::color (uchar r, uchar g, uchar b) [virtual]
see fl_color(uchar r, uchar g, uchar b).
Reimplemented from Fl_Graphics_Driver.

9.106.2.4 descent()

int Fl_Quartz_Graphics_Driver::descent () [virtual]
see fl_descent().
Reimplemented from Fl_Graphics_Driver.

9.106.2.5 draw() [1/5]

void Fl_Quartz_Graphics_Driver::draw (const char *str, int n, int x, int y) [virtual]
see fl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.106.2.6 draw() [2/5]

void Fl_Quartz_Graphics_Driver::draw (Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
Draws an Fl_Bitmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.106.2.7 draw() [3/5]

void Fl_Quartz_Graphics_Driver::draw (Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
Draws an Fl_Pixmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.106.2.8 draw() [4/5]

void Fl_Quartz_Graphics_Driver::draw (Fl_RGB_Image *rgb,
Draws an Fl_RGB_Image object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.106.2.9 draw() [5/5]

void Fl_Quartz_Graphics_Driver::draw (  
    int angle,  
    const char ∗str,  
    int n,  
    int x,  
    int y ) [virtual]

see fl_draw(int angle, const char ∗str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.106.2.10 draw_image() [1/2]

void Fl_Quartz_Graphics_Driver::draw_image (  
    const uchar ∗buf,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3,  
    int L = 0 ) [virtual]

see fl_draw_image(const uchar ∗buf, int X,int Y,int W,int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.106.2.11 draw_image() [2/2]

void Fl_Quartz_Graphics_Driver::draw_image (  
    Fl_Draw_Image_Cb cb,  
    void ∗data,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3 ) [virtual]

see fl_draw_image(Fl_Draw_Image_Cb cb, void ∗data, int X,int Y,int W,int H, int D).
Reimplemented from Fl_Graphics_Driver.

9.106.2.12 draw_image_mono() [1/2]

void Fl_Quartz_Graphics_Driver::draw_image_mono (  
    const uchar ∗buf,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 1,  
    int L = 0 ) [virtual]

see fl_draw_image_mono(const uchar ∗buf, int X,int Y,int W,int H, int D, int L).
Reimplemented from Fl_Graphics_Driver.

9.106.2.13 draw_image_mono() [2/2]

```c
void Fl_Quartz_Graphics_Driver::draw_image_mono (  
    Fl_Draw_Image_Cb cb,  
    void * data,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 1 ) [virtual]

see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).
Reimplemented from Fl_Graphics_Driver.
```

9.106.2.14 draw_scaled()

```c
int Fl_Quartz_Graphics_Driver::draw_scaled (  
    Fl_Image * img,  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]

Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.

Returns
    zero when the graphics driver doesn’t implement scaled drawing, non-zero if it does implement it.

Reimplemented from Fl_Graphics_Driver.
```

9.106.2.15 font()

```c
void Fl_Quartz_Graphics_Driver::font (  
    Fl_Font face,  
    Fl_Fontsize size ) [virtual]

see fl_font(Fl_Font face, Fl_Fontsize size).
Reimplemented from Fl_Graphics_Driver.
```

9.106.2.16 height()

```c
int Fl_Quartz_Graphics_Driver::height ( ) [virtual]

see fl_height() .
Reimplemented from Fl_Graphics_Driver.
```

9.106.2.17 rtl_draw()

```c
void Fl_Quartz_Graphics_Driver::rtl_draw (  
    const char * str,  
    int n,  
    int x,  
    int y ) [virtual]

see fl_rtl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.
```

9.106.2.18 text_extents()

```c
void Fl_Quartz_Graphics_Driver::text_extents (  
    const char * t,  
    int n,  
```
9.107 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:

```
Fl_Radio_Button
    ↓
Fl_Button
    ↓
Fl_Widget
```

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).
- 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)

The documentation for this class was generated from the following files:

- Fl_Device.H
- fl_color_mac.cxx
- Fl_Device.cxx
- Fl_Double_Window.cxx
- fl_draw_image_mac.cxx

9.106.19 width() [1/2]

double Fl_Quartz_Graphics_Driver::width {
    const char * str,
    int n ) [virtual]

see fl_width(const char *str, int n).
Reimplemented from Fl_Graphics_Driver.

9.106.20 width() [2/2]

double Fl_Quartz_Graphics_Driver::width {
    unsigned int c ) [virtual]

see fl_width(unsigned int n).
Reimplemented from Fl_Graphics_Driver.

Generated by Doxygen
• 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.

• virtual int **handle**(int)
  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 **Fl_Group** * as_group ()
  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.

• **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, 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)
  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_Imge *deimage ()
  Gets the image that is used as part of the widget label.
• const Fl_Imge *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
  Sets the image to use as part of the widget label.
• void image (Fl_Image &img)
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.
• 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(void *)
  Sets the user data for this widget.
- unsigned int visible() const
  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() 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 *cb, void *d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
Protected Member Functions inherited from Fl_Button

- virtual void draw ()
  
  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 ()
  
  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_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
9.107.1 Constructor & Destructor Documentation

9.107.1.1 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

| 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_Button.H
- Fl_Button.cxx

9.108 Fl_Radio_Light_Button Class Reference

Inheritance diagram for Fl_Radio_Light_Button:

```
Fl_Widget
   ↓
Fl_Button
   ↓
Fl_Light_Button
   ↓
Fl_Radio_Light_Button
```

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.

- virtual int handle (int)  
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()  
  Same as value(0).

- 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 Fl_Group * as_group()`  
  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`.
- `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, 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)
  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.
• const Fl_Image ∗ deimage () const
  
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label.
• void do_callback ()
  
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)
  
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  
  Calls the widget callback.
• 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.
• const Fl_Image ∗ image () const
• void image (Fl_Image &img)

  Sets the image to use as part of the widget label.
• void image (Fl_Image ∗img)

  Sets the image to use as part of the widget label.
• 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.

- unsigned int \texttt{output} () const
  Returns if a widget is used for output only.

- \texttt{Fl\_Group\ *} \texttt{parent} () const
  Returns a pointer to the parent widget.

- void \texttt{parent} (\texttt{Fl\_Group\ *}\texttt{p})
  Internal use only - "for hacks only".

- void \texttt{position} (int \texttt{X}, int \texttt{Y})
  Repositions the window or widget.

- void \texttt{redraw} ()
  Schedules the drawing of the widget.

- void \texttt{redraw\_label} ()
  Schedules the drawing of the label.

- virtual void \texttt{resize} (int \texttt{x}, int \texttt{y}, int \texttt{w}, int \texttt{h})
  Changes the size or position of the widget.

- \texttt{Fl\_Color\ *} \texttt{selection\_color} () const
  Gets the selection color.

- void \texttt{selection\_color} (\texttt{Fl\_Color} \texttt{a})
  Sets the selection color.

- void \texttt{set\_active} ()
  Marks the widget as active without sending events or changing focus.

- void \texttt{set\_changed} ()
  Marks the value of the widget as changed.

- void \texttt{set\_output} ()
  Sets a widget to output only.

- void \texttt{set\_visible} ()
  Makes the widget visible.

- void \texttt{set\_visible\_focus} ()
  Enables keyboard focus navigation with this widget.

- virtual void \texttt{show} ()
  Makes a widget visible.

- void \texttt{size} (int \texttt{W}, int \texttt{H})
  Changes the size of the widget.

- int \texttt{take\_focus} ()
  Gives the widget the keyboard focus.

- unsigned int \texttt{takesevents} () const
  Returns if the widget is able to take events.

- int \texttt{test\_shortcut} ()
  Returns true if the widget's label contains the entered '\&x' shortcut.

- const char * \texttt{tooltip} () const
  Gets the current tooltip text.

- void \texttt{tooltip} (const char \*\texttt{text})
  Sets the current tooltip text.

- \texttt{Fl\_Window\ *} \texttt{top\_window} () const
  Returns a pointer to the top-level window for the widget.

- \texttt{Fl\_Window\ *} \texttt{top\_window\_offset} (int \texttt{xoff}, int \texttt{yoff}) const
  Finds the x/y offset of the current widget relative to the top-level window.

- uchar \texttt{type} () const
  Gets the widget type.

- void \texttt{type} (uchar \texttt{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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 *cb, void *d)
  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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Generated by Doxygen
Protected Member Functions inherited from `Fl_Light_Button`

- virtual void `draw ()`
  
  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 ()`
  
  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_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`

The documentation for this class was generated from the following files:

- `Fl_Radio_Light_Button.H`
- `Fl_Light_Button.cxx`
9.109   **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.
- virtual int handle (int)
  Handles the specified event.

Public Member Functions inherited from **Fl_Button**

- int clear ()
  Same as **value(0)**.
- 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.

Generated by Doxygen
• 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 Fl_Group * as_group ()
  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.
• 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, 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)
  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.
- const Fl_Image ∗deimage () const
- void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
- void do_callback()
  Calls the widget callback.
- void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.
- void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
- 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.
• const Fl_Image * image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image * img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

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 (void *v)
  Sets the user data for this widget.

unsigned int visible () const
  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 () 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 *cb, void *d)**
  
  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,
  
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }**

  flags possible values enumeration.

**Protected Member Functions inherited from Fl_Light_Button**

- **virtual void draw ()**
  
  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_box ()
  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_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

9.109.1 Constructor & Destructor Documentation

9.109.1.1 Fl_Radio_Round_Button()

Fl_Radio_Round_Button::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.
The button type() is set to FL_RADIO_BUTTON.
Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in X,Y,W,H</td>
<td>position and size of the widget</td>
</tr>
<tr>
<td>in 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_Radio_Round_Button.H
- Fl_Round_Button.cxx

9.110 Fl_Scroll::Fl_Region_LRTB Struct Reference

A local struct to manage a region defined by left/right/top/bottom.

#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

9.110.1 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

9.111 Fl_Scroll::Fl_Region_XYWH Struct Reference

A local struct to manage a region defined by xywh.

#include <Fl_Scroll.H>

Public Attributes

- int h
- int w
- int x
- int y

9.111.1 Detailed Description

A local struct to manage a region defined by xywh.
The documentation for this struct was generated from the following file:

- Fl_Scroll.H
9.112  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)
  - Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()
  - Same as value(0).
- 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 Fl_Group ∗ as_group ()
  
  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.
- 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, 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)
  
  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.

- **const FL_Image * deimage () const**
  void deimage (FL_Image &img)
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (FL_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (FL_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const FL_Image * image () const**
  void image (FL_Image &img)
  Sets the image to use as part of the widget label.
• void `image (Fl_Image ∗img)`
  Sets the image to use as part of the widget label.
• 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 `labels size (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.
• 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.

- 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(void *v)`
  
  Sets the user data for this widget.

- unsigned int `visible()` const
  
  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()` 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 *cb, void *d)  
  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 ,  
    USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

• virtual void draw ()  
  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 ()  
  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_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

### 9.112.1 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.

### 9.112.2 Constructor & Destructor Documentation

#### 9.112.2.1 Fl_Repeat_Button()

**Fl_Repeat_Button::Fl_Repeat_Button**

```cpp
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.

### 9.112.3 Member Function Documentation

#### 9.112.3.1 handle()

```cpp
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.
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.cxx

9.113 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)
  
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- **int clear** ()
  
  Same as value(0).

- **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 Fl_Group * as_group()**
  
  *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.*

- **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, 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_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_changes ()**
  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.

- **const Fl_Image ∗deimage () const**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Generated by Doxygen
Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.
- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.
- **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.
- **const Fl_Image * image () const**
  Sets the image to use as part of the widget label.
- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.
- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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 measure_label (int &ww, int &hh) const**
  Sets width ww and height hh accordingly with the label size.

- **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)**
9.113 Fl_Return_Button Class Reference

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.

- 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.

- 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 (void ∗v)
  Sets the user data for this widget.
- `unsigned int visible () const`
  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 () 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 ()`
  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 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 ()`
  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_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_Widget`

• static void default_callback (Fl_Widget *cb, void *d)
  
  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<<26, INVISIBLE = 1<<27, OUTPUT = 1<<28, NOBORDER = 1<<29,
    FORCEPOSITION = 1<<30, NON_MODAL = 1<<31, SHORTCUT_LABEL = 1<<32,
    CHANGED = 1<<33,
    OVERRIDE = 1<<34, VISIBLE_FOCUS = 1<<35, COPIED_LABEL = 1<<36, CLIP_CHILDREN = 1<<37,
    MENU_WINDOW = 1<<104,
    FORCE_MODAL = 1<<105,
    TOOLTIP_WINDOW = 1<<106,
    MODAL = 1<<107, NO_OVERLAY = 1<<108,
    GROUP_RELATIVE = 1<<109,
    MAC_USE_ACCENTS_MENU = 1<<110,
    USERFLAG3 = 1<<111, USERFLAG2 = 1<<112, USERFLAG1 = 1<<113
  }

  `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

9.113.1 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.
9.113.2 Constructor & Destructor Documentation

9.113.2.1 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.

9.113.3 Member Function Documentation

9.113.3.1 draw()

```cpp
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:
```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw();             // calls Fl_Scrollbar::draw()
```
Reimplemented from Fl_Button.

9.113.3.2 handle()

```cpp
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.

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_Button.
The documentation for this class was generated from the following files:
9.114 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
```

### Public Member Functions

- virtual void color_average (Fl_Color c, float i)
  
  The color_average() method averages the colors in the image with the FLTK color value c.

- Fl_Image *copy ()
  
  The copy() method creates a copy of the specified image.

- virtual Fl_RGB_Image *copy (int W, int H)
  
  The copy() method creates a copy of the specified image.

- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)
  
  The draw() method draws the image with a bounding box.

- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  The draw() method draws the image 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 W, int H, int D=3, int LD=0)
  
  The constructor creates a new image from the specified data.

- virtual void label (Fl_Menu_Item *m)
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void label (Fl_Widget *w)
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- virtual void uncache ()
  
  If the image has been cached for display, delete the cache data.

- virtual ~Fl_RGB_Image ()
  
  The destructor frees all memory and server resources that are used by the image.

### Public Member Functions inherited from Fl_Image

- Fl_Image *copy ()
  
  The copy() method creates a copy of the specified image.

- int count () const
  
  The count() method returns the number of data values associated with the image.

- int d () const
  
  Returns the current image depth.

- const char *const *data () const

Generated by Doxygen
Returns a pointer to the current image data array.

- **void** draw (int X, int Y)
  Draws the image.

- **int** fail ()
  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 height in pixels.

- **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.

- **int** w () const
  Returns the current image width in pixels.

- **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_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).

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_GDI_Graphics_Driver**
- class **Fl_GDI_Printer_Graphics_Driver**
- class **Fl_Quartz_Graphics_Driver**
- class **Fl_Xlib_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_NO_IMAGE = -1
9.114 Fl_RGB_Image Class Reference

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 array 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.
- void h (int H)
  Sets the current image height in pixels.
- void ld (int LD)
  Sets the current line data size in bytes.
- void w (int W)
  Sets the current image width in pixels.

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)

9.114.1 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.

9.114.2 Constructor & Destructor Documentation

9.114.2.1 Fl_RGB_Image() [1/2]

Fl_RGB_Image::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.
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)

9.114.2.2 Fl_RGB_Image() [2/2]

Fl_RGB_Image::Fl_RGB_Image (  
    const Fl_Pixmap * pxm,  
    Fl_Color bg = FL_GRAY )

The constructor creates a new RGBA image from the specified Fl_Pixmap.
The RGBA image is built fully opaque except for the transparent area of the pixmap that is assigned the bg color
with full transparency.
This constructor creates a new internal data array and sets Fl_RGB_Image::alloc_array to 1 so the data array is
deleted when the image is destroyed.

9.114.3 Member Function Documentation

9.114.3.1 color_average()

void Fl_RGB_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]

The color_average() method averages the colors in the image with the FLTK color value c.
The i 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 before changes are applied, to avoid modifying the original image.
Reimplemented from Fl_Image.

9.114.3.2 copy()

Fl_Image * Fl_RGB_Image::copy (  
    int W,  
    int H ) [virtual]

The copy() method creates a copy of the specified image.
If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in
the case of Fl_Shared_Image, released) when you are done with it.
Reimplemented from Fl_Image.

9.114.3.3 desaturate()

void Fl_RGB_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 before changes are applied, to avoid modifying the original image.
Reimplemented from Fl_Image.

9.114.3.4 draw()

void Fl_RGB_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]
int Y,
int W,
int H,
int cx = 0,
int cy = 0 ) [virtual]

Draws the image 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.

9.114.3.5 label() [1/2]

void Fl_RGB_Image::label ( Fl_Menu_Item * m ) [virtual]
The label() methods are an obsolete way to set the image attribute of a widget or menu item.
Use the image() or deimage() methods of the Fl_Widget and Fl_Menu_Item classes instead.
Reimplemented from Fl_Image.

9.114.3.6 label() [2/2]

void Fl_RGB_Image::label ( Fl_Widget * widget ) [virtual]
The label() methods are an obsolete way to set the image attribute of a widget or menu item.
Use the image() or deimage() methods of the Fl_Widget and Fl_Menu_Item classes instead.
Reimplemented from Fl_Image.

9.114.3.7 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)

9.114.3.8 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 ap-
ply 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.

9.114.3.9 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.
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)
  
  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 Fl_Group ∗ as_group ()
  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.
- 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, 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)
  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.
• const Fl_Image ∗deimage () const  
  For back compatibility only.
• void deimage (Fl_Image &img)  
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image ∗img)  
  Sets the image to use as part of the widget label.
• void do_callback ()  
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)  
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)  
  Calls the widget callback.
• 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.
• const Fl_Image * image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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 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.

• **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 (void *v)**
  Sets the user data for this widget.

• **unsigned int visible () const**
  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 () 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 ()`
  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 ()
  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_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 ∗cb, void ∗d)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.
9.115.1 Detailed Description

The Fl_Roller widget is a "dolly" control commonly used to move 3D objects.

![Fl_Roller Widget](image)

Figure 9.26 Fl_Roller

9.115.2 Constructor & Destructor Documentation

9.115.2.1 Fl_Roller()

```cpp
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.

9.115.3 Member Function Documentation

9.115.3.1 draw()

```cpp
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:
```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

9.115.3.2 handle()

```cpp
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.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Generated by Doxygen
Return values

<table>
<thead>
<tr>
<th></th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></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_Roller.H
- Fl_Roller.cxx

9.116 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
```

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.
- virtual int handle (int)
  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()
  Same as value(0).
- 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 Fl_Group * **as_group** ()
  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.

- 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, 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)`
  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.
- `const Fl_Image * deimage () const`
- `void deimage (Fl_Image &img)`
Sets the image to use as part of the widget label.

- void deimage (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label.

- void do_callback ()
  
  Calls the widget callback.

- void do_callback (Fl_Widget ∗o, long arg)
  
  Calls the widget callback.

- void do_callback (Fl_Widget ∗o, void ∗arg=0)
  
  Calls the widget callback.

- 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.

- const Fl_Image ∗image () const
  
  Sets the image to use as part of the widget label.

- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- Fl_Image ∗image (Fl_Image ∗img)
  
  Sets the image to use as part of the widget label.

- 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.

- 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.

- **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(void *v)
  Sets the user data for this widget.

• unsigned int visible() const
  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() 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* cb, void *d)
  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_ACCECTS_MENU = 1<<19, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31}
  flags possible values enumeration.
Protected Member Functions inherited from Fl_Light_Button

• virtual void draw ()

  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 ()

  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_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
9.116.1 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()`.

![Fl_Round_Button](image)

Figure 9.27 Fl_Round_Button

The `Fl_Round_Button` subclass display 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`.

9.116.2 Constructor & Destructor Documentation

9.116.2.1 Fl_Round_Button()

```
Fl_Round_Button::Fl_Round_Button (  
ext, 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.

![Fl_Round_Button](image)

Figure 9.28 Fl_Round_Button

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>Type</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>

The documentation for this class was generated from the following files:

- `Fl_Round_Button.H`
- `Fl_Round_Button.cxx`

9.117 Fl_Round_Clock Class Reference

A clock widget of type `FL_ROUND_CLOCK`.

```cpp
#include <Fl_Round_Clock.H>
```

Inheritance diagram for `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 boxtype, position, size, and label string.

• int handle (int)

  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).

• int minute () const

  Returns the displayed minute (0 to 59).

• int second () const

  Returns the displayed second (0 to 60, 60=leap second).

• 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 Fl_Group * as_group ()`
  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`.

- `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, 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)`
  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.
- const Fl_Image * deimage () const
- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.
- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
- void do_callback ()
  
  Calls the widget callback.
- void do_callback (Fl_Widget *o, long arg)
  
  Calls the widget callback.
- void do_callback (Fl_Widget *o, void *arg=0)
  
  Calls the widget callback.
- 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.
- const Fl_Image * image () const
- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.
- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
- 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.

- 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.

- 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** (void *v)
  Sets the user data for this widget.

• unsigned int **visible** () const
  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** () 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 ∗cb, void ∗d)
  
  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,
  USERFLAG3 = 1<<29,
  USERFLAG2 = 1<<30,
  USERFLAG1 = 1<<31
} flags possible values enumeration.

Protected Member Functions inherited from Fl_Clock_Output

- void draw ()
  
  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 (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 ()
  
  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_label () const
  
  Draws the widget's label at the default 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.*

### 9.117.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.
Has no box.
The documentation for this class was generated from the following files:

- Fl_Round_Clock.H
- Fl_Scroll.cxx

### 9.118 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 scrollbar's xywh region and tab values.

- **struct ScrollInfo**
  Structure to manage scrollbar and widget interior sizes.

#### Public Types

```cpp
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...

- **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)**
  
  Handles the specified event.

- **void resize (int X, int Y, int W, int H)**
  
  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.

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.

- **virtual Fl_Group ∗as_group ()**
  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  
  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 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**

Generated by Doxygen
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
  See void Fl_Group::resizable(Fl_Widget *box)

- void **resizable** (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)

- void **resizable** (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.

- 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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- 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, 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)
  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.

- **const FL_Image * deimage () const**
  
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const FL_Image * image () const**
  
    Sets the image to use as part of the widget label.

- **void image (FL_Image &img)**
  
    Sets the image to use as part of the widget label.

- **void image (FL_Image *img)**
  
    Sets the image to use as part of the widget label.

- **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.
• 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.
• 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 (void *v)`
  Sets the user data for this widget.

- **unsigned int** `visible () const`
  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 () 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`
- **Fl_Scrollbar** `scrollbar`

### Protected Member Functions

- **void** `bbox (int &, int &, int &, int &)`
  Returns the bounding box for the interior of the scrolling area, inside the scrollbars.

- **void** `draw ()`
  Draws the widget.

- **void** `recalc_scrollbars (ScrollInfo &si)`
  Calculate visibility/size/position of scrollbars, find children's bounding box.

### Protected Member Functions inherited from **Fl_Group**

- **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 ()
  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_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 \texttt{Fl_Widget}

\begin{itemize}
\item static void \texttt{default\_callback (Fl\_Widget} *\texttt{cb}, void *\texttt{d})
\emph{The default callback for all widgets that don't set a callback.}
\item static unsigned int \texttt{label\_shortcut (const char} *\texttt{t})
\emph{Returns the Unicode value of the 'x' shortcut in a given text.}
\item static int \texttt{test\_shortcut (const char} *\texttt{t}, const bool require\_alt=false)
\emph{Returns true if the given text} \texttt{t} \emph{contains the entered 'x' shortcut.}
\end{itemize}

Protected Types inherited from \texttt{Fl\_Widget}

\begin{itemize}
\item 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{USERFLAG3 = 1<<29}, \texttt{USERFLAG2 = 1<<30}, \texttt{USERFLAG1 = 1<<31}}\emph{flags possible values enumeration.}
\end{itemize}

9.118.1 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 9.29 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 \texttt{Fl\_Double\_Window}.

By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll. Use \texttt{Fl\_Scroll::type()} to change this as follows:

\begin{itemize}
\item 0 - No scrollbars
\item \texttt{Fl\_Scroll::HORIZONTAL - Only a horizontal scrollbar.}
\item \texttt{Fl\_Scroll::VERTICAL - Only a vertical scrollbar.}
\item \texttt{Fl\_Scroll::BOTH - The default is both scrollbars.}
\item \texttt{Fl\_Scroll::HORIZONTAL\_ALWAYS - Horizontal scrollbar always on, vertical always off.}
\item \texttt{Fl\_Scroll::VERTICAL\_ALWAYS - Vertical scrollbar always on, horizontal always off.}
\item \texttt{Fl\_Scroll::BOTH\_ALWAYS - Both always on.}
\end{itemize}
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 scrollbars 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.

### 9.118.2 Constructor & Destructor Documentation

#### 9.118.2.1 Fl_Scroll()

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.

### 9.118.3 Member Function Documentation

#### 9.118.3.1 bbox()

void Fl_Scroll::bbox (int & X, int & Y, int & W, int & H) [protected]

Returns the bounding box for the interior of the scrolling area, inside the scrollbars.
Currently this is only reliable after `draw()`, and before any resizing of the Fl_Scroll or any child widgets occur.

**Todo** The visibility of the scrollbars ought to be checked/calculated outside of the `draw()` method (STR #1895).

#### 9.118.3.2 draw()

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:

```cpp
Fl_Widget ∗s = ∗scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.
9.118.3.3 handle()

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.

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.

9.118.3.4 recalc_scrollbars()

void Fl_Scroll::recalc_scrollbars (  
  ScrollInfo & si ) [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.

Parameters

| in,out | si | – ScrollInfo structure |

9.118.3.5 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.

9.118.3.6 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:

```c
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.

9.118.3.7 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)

9.118.3.8 scrollbar_size() [2/2]

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

| in      | newSize    | Sets the scrollbar size in pixels.
|---------|------------|--------------------------------------
|         |            | If 0 (default), scrollbar size tracks the global Fl::scrollbar_size() |
See also

\texttt{Fl::scrollbar\_size()}

\section*{9.118.3.9 \texttt{xposition()}}

\begin{verbatim}
int Fl_Scroll::xposition ( ) const [inline]
\end{verbatim}

Gets the current horizontal scrolling position.

\section*{9.118.3.10 \texttt{yposition()}}

\begin{verbatim}
int Fl_Scroll::yposition ( ) const [inline]
\end{verbatim}

Gets the current vertical scrolling position.

The documentation for this class was generated from the following files:

- \texttt{Fl\_Scroll.H}
- \texttt{Fl\_Scroll.cxx}

\section*{9.119 \texttt{Fl\_Scrollbar} Class Reference}

The \texttt{Fl\_Scrollbar} widget displays a slider with arrow buttons at the ends of the scrollbar.

\begin{verbatim}
#ifndef Fl_Scrollbar
#include <Fl_Scrollbar.H>
#endif
\end{verbatim}

Inheritance diagram for \texttt{Fl\_Scrollbar}:

```
Fl_Widget
\downarrow
Fl_Valuator
\downarrow
Fl_Slider
\downarrow
Fl_Scrollbar
```

\subsection*{Public Member Functions}

- \texttt{\texttt{Fl\_Scrollbar} (int X, int Y, int W, int H, const char *L=0)}
  
  \textit{Creates a new \texttt{Fl\_Scrollbar} widget with given position, size, and label.}

- \texttt{int handle (int)}

  \textit{Handles the specified event.}

- \texttt{int linesize ( ) const}

  \textit{Get the size of step, in lines, that the arrow keys move.}

- \texttt{void linesize (int i)}

  \textit{This number controls how big the steps are that the arrow keys do.}

- \texttt{int value ( ) const}

  \textit{Gets the integer value (position) of the slider in the scrollbar.}

- \texttt{int value (int p)}

  \textit{Sets the value (position) of the slider in the scrollbar.}

- \texttt{int value (int pos, int windowSize, int first, int total)}

  \textit{Sets the position, size and range of the slider in the scrollbar.}

- \texttt{\texttt{\sim}Fl\_Scrollbar ( )}

  \textit{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)
  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^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_Valulator::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** ()

- **void activate** ()
  
  Returns whether the widget is active.

- **int active** () const
  
  Returns whether the widget and all of its parents are active.

- **FI_Align align** () const
  
  Gets the label alignment.

- **void align (FI_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 FI_Gl_Window * as_gl_window** ()
  
  Returns an FI_Gl_Window pointer if this widget is an FI_Gl_Window.

- **virtual FI_Group * as_group** ()
  
  Returns an FI_Group pointer if this widget is an FI_Gl_Window.

- **virtual FI_Window * as_window** ()
  
  Returns an FI_Window pointer if this widget is an FI_Window.

- **FI_Boxtype box** () const
  
  Gets the box type of the widget.

- **void box (FI_Boxtype new_box)**
  
  Sets the box type for the widget.

- **FI_Callback_p callback** () const
  
  Gets the current callback function for the widget.

- **void callback (FI_Callback *cb)**
  
  Sets the current callback function for the widget.

- **void callback (FI_Callback *cb, void *p)**
  
  Sets the current callback function for the widget.

- **void callback (FI_Callback0 *cb)**
  
  Sets the current callback function for the widget.

- **void callback (FI_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** ()
• 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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• void draw_label (int, int, int, Fl_Align) const
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
• int h () const
• virtual void hide ()
  Makes a widget invisible.
• Fl_Image * image ()
  Gets the image that is used as part of the widget label.
• const Fl_Image * image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.
- `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 (void ∗ v)`
  Sets the user data for this widget.
- `unsigned int visible () const`
  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 () 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 ()
  
  Draws the widget.

Protected Member Functions inherited from Fl_Slider

• void draw ()
  
  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** ()
  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_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 *cb, void *d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

9.119.1 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.

![Figure 9.30 Fl_Scrollbar](image)

9.119.2 Constructor & Destructor Documentation

9.119.2.1 Fl_Scrollbar()

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.

9.119.3 Member Function Documentation

9.119.3.1 draw()

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:

---

Generated by Doxygen
FL_Widget *s = &scroll;  // scroll is an embedded Fl_Scrollbar
s->draw();            // calls Fl_Scrollbar::draw()
Implements Fl_Widget.

9.119.3.2 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.

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.

9.119.3.3 linesize()

```cpp
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.

9.119.3.4 value() [1/3]

```cpp
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)

9.119.3.5 value() [2/3]

```cpp
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)
9.119.3.6  value()  [3/3]

int Fl_Scrollbar::value (  
   int pos,
   int windowSize,
   int first,
   int total ) [inline]

Sets the position, size and range of the slider in the scrollbar.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>position, first line displayed</td>
</tr>
<tr>
<td>windowSize</td>
<td>number of lines displayed</td>
</tr>
<tr>
<td>first</td>
<td>number of first line</td>
</tr>
<tr>
<td>total</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

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

9.120  FL_Scroll::FL_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

9.120.1  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

9.121  FL_Secret_Input Class Reference

The `FL_Secret_Input` class is a subclass of `FL_Input` that displays its input as a string of placeholders.

```cpp
#include <FL_Secret_Input.H>
```

Inheritance diagram for `FL_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)
  
  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 **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.

- **Fl_Input_** (int, int, int, int, const char ∗=0)
  
  Creates a new Fl_Input_ widget.

- Fl_Char **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 **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 position of the text cursor.

• int position (int p)
  
  Sets the cursor position and mark.

• int position (int p, int m)
  
  Sets the index for the cursor and mark.

• 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 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)
  
  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 textszie () const
  
  Gets the size of the text in the input field.

• void textszie (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 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 Fl_Group * as_group ()`
  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`.

- `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, 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)`
  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.

- **const Fl_Image ∗deimage () const**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image =img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget ∗o, void ∗arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image * image () const**
  Sets the image to use as part of the widget label.

- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **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.

- **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 ()**
  Generated by Doxygen
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.

- **virtual void show ()**
  Makes a widget visible.

- **int 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 (void *v)**
  Sets the user data for this widget.

- **unsigned int visible () const**
  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 () 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 *cb, void *d)**

  *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,\
  *USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }**

  *flags possible values enumeration.*

#### Protected Member Functions inherited from Fl_Input

• **void draw ()**

  *Draws the widget.*

#### Protected Member Functions inherited from Fl_Input_

• **void drawtext (int, int, int, int)**

  *Draws the text in the passed bounding box.*

• **void handle_mouse (int, int, int, int keepmark=0)**

  *Handles mouse clicks and mouse moves.*

• **int handletextarea (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 ()
  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
  Draws the widget box around the widget.
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
• void w (int v)
  Internal use only.
• void xscroll (int yOffset)
• 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 drawBackdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
• void drawBackDrop () 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 ()
  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_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.
9.121.1 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.

9.121.2 Constructor & Destructor Documentation

9.121.2.1 Fl_Secret_Input()

Fl_Secret_Input::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. The default boxtype is FL_DOWN_BOX. Inherited destructor destroys the widget and any value associated with it.

9.121.3 Member Function Documentation

9.121.3.1 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.

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_Input.

The documentation for this class was generated from the following files:

- Fl_Secret_Input.H
- Fl_Input.cxx

9.122 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:
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** ()

  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 ()**
  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)  
  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 position () const  
  Gets the vertical scroll position of the list as a pixel position pos.

• void position (int pos)  
  Sets the vertical scroll position of the list to pixel position pos.

• void resize (int X, int Y, int W, int H)  
  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  
  This method has been deprecated, existing for backwards compatibility only.

• void scrollbar_width (int width)  
  This method has been deprecated, existing for backwards compatibility only.

• 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`.

### 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** `*array () const`
  
  Returns a pointer to the array of children.

- **virtual** `Fl_Group * as_group ()`
  
  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.

- **int** `handle (int)`
  
  Handles the specified event.

- **void** `init_sizes ()`
  
  Resets the internal array of widget sizes and positions.

- **void** `insert (Fl_Widget &o, Fl_Widget *before)`
  
  The widget is removed from its current group (if any) and then inserted into this group.
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
  See void Fl_Group::resizable(Fl_Widget *box)
- void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)
- void resizable (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.
- void resize (int, int, int, int)
  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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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_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.

- 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.

- const Fl_Image ∗ deimage () const

- void deimage (Fl_Image & img)
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image img)
  Sets the image to use as part of the widget label.

- void do_callback ()
Calls the widget callback.

- void **do_callback** (Fl_Widget ∗o, long arg)
  
  Calls the widget callback.

- void **do_callback** (Fl_Widget ∗o, void ∗arg=0)
  
  Calls the widget callback.

- 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_Imagex **image** ()
  
  Gets the image that is used as part of the widget label.

- const Fl_Imagex **image** () const
  
  void **image** (Fl_Imagex &img)
  
  Sets the image to use as part of the widget label.

- void **image** (Fl_Imagex ∗img)
  
  Sets the image to use as part of the widget label.

- 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 **labelsze** () const
  
  Gets the font size in pixels.

- void **labelsze** (Fl_Fontsize pix)
  
  Sets the font size in pixels.

- Fl_Labeltype **labeltpe** () const
  
  Gets the label type.

- void **labeltpe** (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.

- 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.

- **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 (void ∗v)
  Sets the user data for this widget.

- **unsigned int** visible () const
  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 () 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 *cb, void *d)
  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 , 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 
  The height of the entire list of all visible() items in pixels.
• int incr_height () const 
  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 
  Return the item at specified line.
• void item_draw (void * item, int X, int Y , int W, int H) const 
  Draws item at the position specified by XYWH.
• void * item_first () const 
  Returns the very first item in the list.
• int item_height (void * item) const 
  Returns height of item in pixels.
• void * item_last () const 
  Returns the very last item in the list.
• void * item_next (void * item) const 
  Returns the next item after item.
• void * item_prev (void * item) const 
  Returns the previous item before item.
• void item_select (void * item, int val) 
  Change the selection state of item to the value val.
• int item_selected (void * item) const 
  See if item is selected.
• void item_swap (void *a, void *b) 
  Swap the items a and b.
• const char * item_text (void *item) const 
  Returns the label text for item.
• int item_width (void *item) const 
  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** ()
  
  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

• void **draw** ()
  
  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** ()
  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_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 ∗L = 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.

9.122.1 Detailed Description

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.
As long as the mouse button is held down on an unselected item it is highlighted. Normally the callback is done when the user presses the mouse, but you can change this with when().
See Fl_Browser for methods to add and remove lines from the browser.

9.122.2 Constructor & Destructor Documentation

9.122.2.1 Fl_Select_Browser()

Fl_Select_Browser::Fl_Select_Browser (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗L = 0L )

Creates a new Fl_Select_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_SELECT_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_Select_Browser.H
- Fl_Browser.cxx

### 9.123 Fl_Shared_Image Class Reference

This class supports caching, loading, scaling, and drawing of image files.

```cpp
#include <Fl_Shared_Image.H>
```

Inheritance diagram for Fl_Shared_Image:

```
Fl_Shared_Image
    Fl_Image
```

**Public Member Functions**

- `virtual void color_average (Fl_Color c, float i)`
  
  The `color_average()` method averages the colors in the image with the FLTK color value `c`.

- `Fl_Image * copy ()`

  The `copy()` method creates a copy of the specified image.

- `virtual Fl_Image * copy (int W, int H)`

  The `copy()` method creates a copy of the specified image.

- `virtual void desaturate ()`

  The `desaturate()` method converts an image to grayscale.

- `void draw (int X, int Y)`

- `virtual void draw (int X, int Y, int W, int H, int cx, int cy)`

  Draws the image with a bounding box.

- `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 ()`

  Releases and possibly destroys (if refcount <= 0) a shared image.

- `void reload ()`

  Reloads the shared image from disk.

- `void scale (int width, int height, int proportional=1, int can_expand=0)`

  Sets the drawing size of the shared image.

- `virtual void uncache ()`

  If the image has been cached for display, delete the cache data.

**Public Member Functions inherited from Fl_Image**

- `Fl_Image * copy ()`

  The `copy()` method creates a copy of the specified image.

- `int count () const`

  The `count()` method returns the number of data values associated with the image.

- `int d () const`
Returns the current image depth.

• const char * const * data () const
  Returns a pointer to the current image data array.

• void draw (int X, int Y)
  Draws the image.

• int fail ()
  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 height in pixels.

• 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)
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

• virtual void label (Fl_Widget * w)
  The label() methods are 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.

• int w () const
  Returns the current image width in pixels.

• 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 formats.

• static Fl_Shared_Image * find (const char * name, int W=0, int H=0)
  Finds a shared image from its name and size specifications.

• static Fl_Shared_Image * 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_Image * get (Fl_RGB_Image * rgb, int own_it=1)
  Builds a shared image from a pre-existing Fl_RGB_Image.

• static Fl_Shared_Image ** images ()
  Returns the Fl_Shared_Image array.

• static int num_images ()
  Returns the total number of shared images in the array.

• static void remove_handler (Fl_Shared_Handler f)
  Removes a shared image handler.

• static void scaling_algorithm (Fl_RGB_Scaling algorithm)
  Sets what algorithm is used when resizing a source image.

Static Public Member Functions inherited from Fl_Image

• static Fl_RGB_Scaling RGB_scaling ()
  Returns the currently used RGB scaling method.

• static void RGB_scaling (Fl_RGB_Scaling)
  Sets the RGB image scaling method used for copy(int, int).
Protected Member Functions

• void add ()
   
   Adds a shared image to the image cache.

• 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 ()
• 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 array 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.

• void h (int H)
   
   Sets the current image height in pixels.

• void ld (int LD)
   
   Sets the current line data size in bytes.

• void w (int W)
   
   Sets the current image width in pixels.

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

Generated by Doxygen
9.123 Fl_Shared_Image Class Reference

Friends

- class Fl_JPEG_Image
- class Fl_PNG_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_NO_IMAGE = -1

9.123.1 Detailed Description

This class supports caching, loading, scaling, 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, and PNG. 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_Shared_Image::get()
- Fl_Shared_Image::find()
- Fl_Shared_Image::release()

9.123.2 Constructor & Destructor Documentation

9.123.2.1 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.

9.123.2.2 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.

9.123.2.3 ~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.

9.123.3 Member Function Documentation

9.123.3.1 add()

void Fl_Shared_Image::add ( ) [protected]

Adds a shared image to the image cache.
This **protected** method adds an image to the cache, an ordered list of shared images. The cache is searched for a matching image whenever one is requested, for instance with `Fl_Shared_Image::get()` or `Fl_Shared_Image::find()`.

### 9.123.3.2 color_average()

```c
void Fl_Shared_Image::color_average (Fl_Color c, float i) [virtual]
```

The `color_average()` method averages the colors in the image with the FLTK color value `c`. The `i` 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 before changes are applied, to avoid modifying the original image. Reimplemented from `Fl_Image`.

### 9.123.3.3 compare()

```c
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()`.

**Returns**

Whether the images match or their relative sort order (see text).

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the images match</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Image i0 is less than image i1</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>Image i0 is greater than image i1</td>
</tr>
</tbody>
</table>

### 9.123.3.4 copy()

```c
Fl_Image * Fl_Shared_Image::copy (int W, int H) [virtual]
```

The `copy()` method creates a copy of the specified image. If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in the case of `Fl_Shared_Image`, released) when you are done with it. Reimplemented from `Fl_Image`.

### 9.123.3.5 desaturate()

```c
void Fl_Shared_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 before changes are applied, to avoid modifying the original image. Reimplemented from Fl_Image.

### 9.123.3.6 draw()

```c
void Fl_Shared_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx,  
    int cy  
) [virtual]
```

Draws the image 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.

### 9.123.3.7 find()

```c
Fl_Shared_Image * Fl_Shared_Image::find (  
    const char * name,  
    int W = 0,  
    int H = 0  
) [static]
```

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.

### 9.123.3.8 get() [1/2]

```c
Fl_Shared_Image * Fl_Shared_Image::get (  
    const char * name,  
    int W = 0,  
    int H = 0  
) [static]
```

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>

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)

9.123.3.9 get() [2/2]

Fl_Shared_Image * Fl_Shared_Image::get ( Fl_RGB_Image * rgb,
                                          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>__it</td>
<td>own</td>
<td>1 if the shared image should delete rgb when it is itself deleted, 0 otherwise</td>
</tr>
</tbody>
</table>

Version

1.3.4

9.123.3.10 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

9.123.3.11 refcount()

int Fl_Shared_Image::refcount ( ) [inline]

Returns the number of references of this shared image. When reference is below 1, the image is deleted.
9.123.3.12 release()

void Fl_Shared_Image::release ( )
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.

9.123.3.13 scale()

void Fl_Shared_Image::scale ( int width,
int height,
int proportional = 1,
int can_expand = 0 )
Sets the drawing size of the shared image.
This function gives the shared image its own size, independently from the size of the original image that is typically larger. This can be useful to draw a shared image on a drawing surface whose resolution is higher than the drawing unit for this surface: all pixels of the original image become available to fill an area of the drawing surface sized at width,height. Examples of such drawing surfaces: laser printers, PostScript files, PDF printers, retina displays on Apple hardware.

Parameters

| width,height | maximum width and height (in drawing units) to use when drawing the shared image |
| proportional | if not null, keep the width and height of the shared image proportional to those of its original image |
| can_expand | if null, the width and height of the shared image will not exceed those of the original image |

Version

1.3.4 and requires compiling with FLTK_ABI_VERSION = 10304

Example code: scale an image to fit in a box

FL_Box *b = ... // a box
Fl_Shared_Image *shared = Fl_Shared_Image::get("/path/to/picture.jpeg"); // read a picture file
shared->resize(b->w(), b->h(), 1); // set the drawing size of the shared image to the size of the box
b->image(shared); // use the shared image as the box image
b->align(FL_ALIGN_INSIDE | FL_ALIGN_CENTER | FL_ALIGN_CLIP); // the image is to be drawn centered in the box

9.123.3.14 scaling_algorithm()

static void Fl_Shared_Image::scaling_algorithm ( Fl_RGB_Scaling algorithm ) [inline], [static]
Sets what algorithm is used when resizing a source image.
The default algorithm is FL_RGB_SCALING_BILINEAR. Drawing an Fl_Shared_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 Linux or MSWindows after having called Fl_Shared_Image::scale(). This function controls what method is used when the image to be resized is an Fl_RGB_image.

Version

1.3.4 and requires compiling with FLTK_ABI_VERSION = 10304

9.123.3.15 uncache()

void Fl_Shared_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.
The documentation for this class was generated from the following files:

- Fl_Shared_Image.H
- Fl_Shared_Image.cxx
9.124 Fl_Simple_Counter Class Reference

This widget creates a counter with only 2 arrow buttons.

```c
#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)
  
  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 Fl_Group * as_group ()`
  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`.

- `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, 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)`
  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.
- const Fl_Image * **deimage** () const
  void **deimage** (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void **deimage** (Fl_Image *img)
  Sets the image to use as part of the widget label.
- void **do_callback** ()
  Calls the widget callback.
- void **do_callback** (Fl_Widget *o, long arg)
  Calls the widget callback.
- void **do_callback** (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
- 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.
- const Fl_Image * **image** () const
  void **image** (Fl_Image &img)
  Sets the image to use as part of the widget label.
- void **image** (Fl_Image *img)
  Sets the image to use as part of the widget label.
- 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.
  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.
• 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 (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  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 () 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 ∗cb, void ∗d)
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Counter

• void draw ()
  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 ()
  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_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.

9.124.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

![Figure 9.31 Fl_Simple.Counter](image)

The documentation for this class was generated from the following files:

- Fl_Simple_Counter.H
- Fl_Counter.cxx
9.125 Fl_Single_Window Class Reference

This is the same as Fl_Window.

#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 flush ()
  
  Forces the window to be drawn, this window is also made current and calls draw().

• int make_current ()
  
  Puts the window on the screen.

• void show ()
  
  Puts the window on the screen.

• void show (int a, char ∗∗b)

Public Member Functions inherited from Fl_Window

• virtual Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• unsigned int border () const
  
  See void Fl_Window::border(int)

• 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.

• 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 ()

Generated by Doxygen
Returns the window height including any window title bar and any frame added by the window manager.

- int decorated_w()
  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.

- Fl_Window(int w, int h, const char∗ title=0)
  Creates a window from the given size and title.

- Fl_Window(int x, int y, int w, int h, const char∗ title=0)
  Creates a window from the given position, size 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.

- virtual int handle(int)
  Handles the specified event.

- virtual void hide()
  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)
  Sets the current icon window target dependent data.

- void iconize()
  Iconifies the window.

- const char∗ iconlabel() const
  See void Fl_Window::iconlabel(const char∗)

- void iconlabel(const char ∗) (const)
  Sets the icon label.

- void icons(const Fl_RGB_Image ∗[], int)
  Sets the window icons.

- const char∗ label() const
• 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.

• 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.

• unsigned int **override**() const
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

• virtual void **resize**(int X, int Y, int W, int H)
  Changes the size and position of 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|FL_OVERRIDE.

• void **set_tooltip_window**()
  Marks the window as a tooltip window.

• 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**(int argc, char **argv)
  Puts the window on the screen 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 **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 &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.
- virtual \Fl_Group * as_group ()
  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 *)&
  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
  See void Fl_Group::resizable(Fl_Widget *box)
• **void resizable (Fl_Widget &o)**
  
  See void Fl_Group::resizable(Fl_Widget *box)

• **void resizable (Fl_Widget *o)**
  
  The resizable widget defines the resizing box for the group.

• 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.

• 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, 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)
  
  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.
• const FL_IMAGE ∗deimage () const
• void deimage (FL_IMAGE &img)
  Sets the image to use as part of the widget label.
• void deimage (FL_IMAGE ∗img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (FL_Widget ∗o, long arg)
  Calls the widget callback.
• void do_callback (FL_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.
• const FL_IMAGE ∗image () const
• void image (FL_IMAGE &img)
Sets the image to use as part of the widget label.

- void image (Fl_Image *img)
  Sets the image to use as part of the widget label.

- 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.

- 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.
• 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 (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_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 const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *)
  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, 
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

- virtual void draw ()
  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.

Protected Member Functions inherited from Fl_Group

• 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 ()
  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_label () const
  Draws the widget's label at the given position and size.
• 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_Window`

- `shape_data_type * shape_data_`
  non-null means the window has a non-rectangular shape

Static Protected Attributes inherited from `Fl_Window`

- `static Fl_Window * current_`
  Stores the last window that was made current.

### 9.125.1 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.

### 9.125.2 Member Function Documentation

#### 9.125.2.1 flush()

```cpp
void Fl_Single_Window::flush ( ) [virtual]
```

Forces the window to be drawn, this window is also made current and calls `draw()`.
Reimplemented from `Fl_Window`.

#### 9.125.2.2 show()

```cpp
void Fl_Single_Window::show ( ) [virtual]
```

Puts the window on the screen. Usually (on X) this has the side effect of opening the display.
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

```cpp
Fl_Window::show(int argc, char **argv)
```

Reimplemented from `Fl_Window`.
The documentation for this class was generated from the following files:

- `Fl_Single_Window.H`
- `Fl_Single_Window.cxx`

### 9.126 `Fl_Slider` Class Reference

The `Fl_Slider` widget contains a sliding knob inside a box.

```cpp
#include <Fl_Slider.H>
```
Inheritance diagram for Fl_Slider:

<table>
<thead>
<tr>
<th>Fl_Widget</th>
<th>Fl_Valuator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Slider</td>
<td></td>
</tr>
</tbody>
</table>

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)
  
  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 _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 Fl_Group * as_group ()
  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.

- 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, 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)
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image ∗image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label.
• 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.
• 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.
• 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 (void ∗v)
Sets the user data for this widget.

- **unsigned int visible () const**
  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 () 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 ()**
  Draws the widget.

- **void draw (int, int, int, int)**

- **int handle (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 ()
  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_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 ∗ cb, void ∗ d)
  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.
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }

flags possible values enumeration.

9.126.1 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 9.32 Fl_Slider

9.126.2 Constructor & Destructor Documentation

9.126.2.1 Fl_Slider()

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.
9.126.3 Member Function Documentation

9.126.3.1 bounds()

```c
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.

9.126.3.2 draw()

```c
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:
```c
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```
Implements Fl_Widget.
Reimplemented in Fl_Value_Slider.

9.126.3.3 handle()

```c
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.

Parameters

| in  | event | the kind of event received |

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.
Reimplemented in Fl_Value_Slider.

9.126.3.4 scrollvalue()

```c
int Fl_Slider::scrollvalue ( int pos,
    int size,
    int first,
    int total )
```
Sets the size and position of the sliding knob in the box.
### 9.126.3.5 slider_size()

```cpp
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

### 9.127 Fl_Spinner Class Reference

This widget is a combination of the input widget and repeat buttons.

```cpp
#include <Fl_Spinner.H>
```

Inheritance diagram for Fl_Spinner:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Spinner
```

#### Public Member Functions

- **Fl_Color color () const**
  
  *Return the background color of the spinner widget's input field.*

- **void color (Fl_Color v)**
  
  *Change 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 ()**
  
  *Sets or returns the format string for the value.*

- **void format (const char *f)**
  
  *Sets or returns the format string for the value.*

- **int handle (int event)**
  
  *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.*
• double maximum () const
  Spelling mistakes retained for source compatibility.
• double minimum () const
  Gets the minimum value of the widget.
• void minimum (double m)
  Sets the minimum value of the widget.
• double minimum () const
  Spelling mistakes retained for source compatibility.
• 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)
  Resizes the Fl_Group widget and all of its children.
• Fl_Color selection_color () const
  Return the selection color of the spinner widget's input field.
• void selection_color (Fl_Color val)
  Change the selection color of the spinner widget's input field.
• double step () const
  Sets or returns the amount to change the value when the user clicks a button.
• void step (double s)
  See double Fl_Spinner::step() const.
• 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 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.

• virtual **Fl_Group** *as_group ()
  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 &, 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
  See void **Fl_Group::resizable(Fl_Widget *box)**

• void **resizable (Fl_Widget &o)**
  See void **Fl_Group::resizable(Fl_Widget *box)**

• void **resizable (Fl_Widget *)**
  The resizable widget defines the resizing box for the group.

• 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 Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• 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, 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)
  
  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.

- const Fl_Image *deimage () const
  
  void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label.

- void do_callback()
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, long arg)
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, void *arg=0)
  
  Calls the widget callback.

- 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.

- const Fl_Image *image () const
  
  void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
• 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.
• 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.

- 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` (void ∗v)
  Sets the user data for this widget.

- unsigned int `visible` () const
  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` () 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 call back 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 Attributes

- `Fl_REPEAT_BUTTON down_button_`
- `Fl_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 *cb, void *d)`
  
  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`,
  `USERFLAG3 = 1<<29`, `USERFLAG2 = 1<<30`, `USERFLAG1 = 1<<31` }

  flags possible values enumeration.

**Protected Member Functions inherited from Fl_Group**

- void `draw ()`
  
  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 `drawOutsideLabel (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 ()
  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_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.

### 9.127.1 Detailed Description

This widget is a combination of the input widget and repeat buttons. The user can either type into the input area or use the buttons to change the value.

![Figure 9.33 Fl_Spinner widget](image)

**Figure 9.33** Fl_Spinner widget
9.127.2 Constructor & Destructor Documentation

9.127.2.1 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.  
Inherited destructor Destroys the widget and any value associated with it.

9.127.3 Member Function Documentation

9.127.3.1 handle()

int Fl_Spinner::handle (  
  int event ) [inline], [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.

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.

9.127.3.2 maximum()

double Fl_Spinner::maximum ( ) const [inline]

Speling mistakes retained for source compatibility.

Deprecated

9.127.3.3 minimum()

double Fl_Spinner::minimum ( ) const [inline]

Speling mistakes retained for source compatibility.

Deprecated

9.127.3.4 resize()

void Fl_Spinner::resize (  
  int X,
int Y, int W, int H ) [inline], [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.

9.127.3.5  step()

double Fl_Spinner::step() const [inline]

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.

9.127.3.6  type() [1/2]

uchar Fl_Spinner::type() const [inline]

Gets the numeric representation in the input field.

See also
   Fl_Spinner::type(uchar)

9.127.3.7  type() [2/2]

void Fl_Spinner::type(uchar v) [inline]

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.

9.127.3.8  value()

void Fl_Spinner::value(double v) [inline]

Sets the current value of the widget.
Before setting value to a non-integer value, the spinner type() should be changed to floating point.
The documentation for this class was generated from the following files:
   • Fl_Spinner.H
   • Fl_Group.cxx

9.128  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:
Public Member Functions

- **const char ∗ class_name ()**
  Returns the name of the class of this object.
- **Fl_Graphics_Driver ∗ driver ()**
  Returns the graphics driver of this drawing surface.
- **void driver (Fl_Graphics_Driver ∗ graphics_driver)**
  Sets the graphics driver of this drawing surface.
- **virtual void set_current (void)**
  Make this surface the current drawing surface.
- **virtual ~Fl_Surface_Device ()**
  The destructor.

Public Member Functions inherited from Fl_Device

- **virtual ~Fl_Device ()**
  Virtual destructor.

Static Public Member Functions

- **static Fl_Surface_Device ∗ surface ()**
  The current drawing surface.

Static Public Attributes

- **static const char ∗ class_id = "Fl_Surface_Device"**

Static Public Attributes inherited from Fl_Device

- **static const char ∗ class_id = "Fl_Device"**
  A string that identifies each subclass of Fl_Device.

Protected Member Functions

- **Fl_Surface_Device (Fl_Graphics_Driver ∗ graphics_driver)**
  Constructor that sets the graphics driver to use for the created surface.

9.128.1 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. Memorize what is the current drawing surface with Fl_Surface_Device *old_current = Fl_Surface_Device::surface();

3. Call surface->set_current() to redirect all graphics requests to surface which becomes the new current drawing surface (not necessary with class Fl_Printer because it is done by Fl_Printer::start_job()).

4. At this point any of the Drawing functions (e.g., fl_rect()) or the Color & Font functions or Drawing Images functions (e.g., Fl_draw_image(), Fl_Image::draw()) operates on the new current drawing surface. Certain drawing surfaces allow additional ways to draw to them (e.g., Fl_Printer::print_widget(), Fl_Image_Surface::draw()).

5. After all drawing requests have been performed, redirect graphics requests back to their previous destination with old_current->set_current();

6. Delete surface.

9.128.2 Member Function Documentation

9.128.2.1 class_name()

const char * Fl_Surface_Device::class_name ( ) [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if (instance->class_name() == Fl_Printer::class_id) { ... }
Reimplemented from Fl_Device.
Reimplemented in Fl_System_Printer.

9.128.2.2 set_current()

void Fl_Surface_Device::set_current ( 
    void ) [virtual]
Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, and Fl_Printer.

9.128.2.3 surface()

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 output
The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.hxx

9.129 Fl_Sys_Menu_Bar Class Reference

A class to create, modify and delete menus that appear on Mac OS X in the menu bar at the top of the screen.
#include <Fl_Sys_Menu_Bar.H>
Inheritance diagram for Fl_Sys_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.
- 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 *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 ()
  Updates the system menu after any change to its items.
- ~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)
  
  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 ∗=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)
  
  See Fl_Boxtype Fl_Menu_::down_box() const

- 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 ∗ **menu**( ) const
  
  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 ∗=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.

- `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.

- `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.

- `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 Fl_Group * as_group ()
  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.
- 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, 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)
  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.

- **const FL_Image * deimage () const**

- **void deimage (FL_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (FL_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (FL_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const FL_Image * image () const**

- **void image (FL_Image &img)**
  Sets the image to use as part of the widget label.
• **void image (Fl_Ima**ge ∗img)**
  
  *Sets the image to use as part of the widget label.*

• **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.*

• **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.

- 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 (void ∗v)**
  
  Sets the user data for this widget.

- unsigned int **visible () const**
  
  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 () 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 ()
  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 ()
  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_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_Widget

- static void default_callback (Fl_Widget ∗cb, void ∗d)
  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 ,
  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_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

9.129.1 Detailed Description

A class to create, modify and delete menus that appear on Mac OS X in the menu bar at the top of the screen.
On other than Mac OS X platforms, Fl_Sys_Menu_Bar is a synonym of class Fl_Menu_Bar.
To use this class, just replace Fl_Menu_Bar by Fl_Sys_Menu_Bar, and, on the Mac platform, 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 with standard FLTK menus.
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 Fl_Sys_Menu_Bar::update() to be visible in the menubar across all platforms.
A few FLTK features are not supported by the Mac System menu:

- no symbolic labels
- no embossed labels
- no font sizes

You can configure a callback for the 'About' menu item to invoke your own code with fl_mac_set_about().
9.129.2 Constructor & Destructor Documentation

9.129.2.1 Fl_Sys_Menu_Bar()

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().

9.129.3 Member Function Documentation

9.129.3.1 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)

9.129.3.2 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.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>- new menu item's label</td>
</tr>
<tr>
<td>shortcut</td>
<td>- new menu item's integer shortcut (can be 0 for none, or e.g. FL_ALT+'x')</td>
</tr>
<tr>
<td>cb</td>
<td>- callback to be invoked when item selected (can be 0 for none, in which case the menubar's callback() can be used instead)</td>
</tr>
<tr>
<td>user_data</td>
<td>- argument to the callback</td>
</tr>
<tr>
<td>flags</td>
<td>- item's flags, e.g. FL_MENU_TOGGLE, etc.</td>
</tr>
</tbody>
</table>

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)

9.129.3.3 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)

9.129.3.4 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()

9.129.3.5 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)

9.129.3.6 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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Menu_Bar.

9.129.3.7 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)
9.129.3.8 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)

9.129.3.9 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 list of Fl_Menu_Item

9.129.3.10 mode()

void Fl_Sys_Menu_Bar::mode (  
    int i,  
    int fl ) [inline]

Sets the flags of item i.

See also  
Fl_Menu_::mode(int i, int fl)

9.129.3.11 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
9.129.3.12 replace()

```cpp
void Fl_Sys_Menu_Bar::replace (  
   int index,  
   const char * name )
```

rename an item from the system menu bar

Parameters

<table>
<thead>
<tr>
<th>index</th>
<th>the index of the item to rename</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>the new item name as a UTF8 string</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Sys_Menu_Bar.H
- Fl_Sys_Menu_Bar.mm

9.130 Fl_System_Printer Class Reference

Print support under MSWindows and Mac OS.

```cpp
#include <Fl_Printer.H>
```

Inheritance diagram for Fl_System_Printer:

```
Fl_System_Printer
   |________________|
   |                |
   | Fl_Paged_Device
   |                |
   | Fl_Surface_Device
   |                |
   | Fl_Device
```

Public Member Functions

- `const char * class_name ()`
  
  *Returns the name of the class of this object.*

- `void end_job (void)`

  *To be called at the end of a print job.*

- `int end_page (void)`

  *To be called at the end of each page.*

- `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 origin (int *x, int *y)`

  *Computes the page coordinates of the current origin of graphics functions.*

- `void origin (int x, int y)`

  *Sets the position in page coordinates of the origin of graphics functions.*

- `int printable_rect (int *w, int *h)`

  *Computes the width and height of the printable area of the page.*

- `void rotate (float angle)`

  *Rotates the graphics operations relatively to paper.*

- `void scale (float scale_x, float scale_y=0.)`

  *Changes the scaling of page coordinates.*

- `int start_job (int pagecount, int *frompage=NULL, int *topage=NULL)`
Starts a print job.

- **int start_page (void)**
  Starts a new printed page.

- **void translate (int x, int y)**
  Translates the current graphics origin accounting for the current rotation.

- **void untranslate (void)**
  Undoes the effect of a previous translate() call.

- **∼Fl_System_Printer (void)**
  The destructor.

### Public Member Functions inherited from Fl_Paged_Device

- **virtual void print_widget (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)**
  Draws the widget on the printed page.

- **void print_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)**
  Prints a window with its title bar and frame if any.

- **virtual void print_window_part (Fl_Window ∗win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  Prints a rectangular part of an on-screen window.

- **virtual ∼Fl_Paged_Device ()**
  The destructor.

### Public Member Functions inherited from Fl_Surface_Device

- **Fl_Graphics_Driver ∗driver ()**
  Returns the graphics driver of this drawing surface.

- **void driver (Fl_Graphics_Driver ∗graphics_driver)**
  Sets the graphics driver of this drawing surface.

- **virtual ∼Fl_Surface_Device ()**
  The destructor.

### Public Member Functions inherited from Fl_Device

- **virtual ∼Fl_Device ()**
  Virtual destructor.

### Static Public Attributes

- **static const char ∗class_id = Fl_Printer::class_id**

### Static Public Attributes inherited from Fl_Paged_Device

- **static const char ∗class_id = "Fl_Paged_Device"**
- **static const page_format page_formats [NO_PAGE_FORMATS]**
  width, height and name of all elements of the enum Page_Format.

### Static Public Attributes inherited from Fl_Surface_Device

- **static const char ∗class_id = "Fl_Surface_Device"**

### Static Public Attributes inherited from Fl_Device

- **static const char ∗class_id = "Fl_Device"**
  A string that identifies each subclass of Fl_Device.
Protected Member Functions

- Fl_System_Printer (void)
  
  The constructor.

Protected Member Functions inherited from Fl_Paged_Device

- Fl_Paged_Device ()
  
  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

Friends

- class Fl_Printer

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_Layou { PORTRAIT = 0, LANDSCAPE = 0x100, REVERSED = 0x200, ORIENTATION = 0x300 }

  Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Paged_Device

- int x_offset
  
  Horizontal offset to the origin of graphics coordinates

- int y_offset
  
  Vertical offset to the origin of graphics coordinates

9.130.1 Detailed Description

Print support under MSWindows and Mac OS. Class Fl_System_Printer is implemented only on the MSWindows and Mac OS platforms. It has no public constructor. Use Fl_Printer instead that is cross-platform and has the same API.
9.130.2 Member Function Documentation

9.130.2.1 class_name()

const char * Fl_System_Printer::class_name ( ) [inline], [virtual]
Returns the name of the class of this object.
Use of the class_name() function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an Fl_Device subclass can be checked with code such as:
if ( instance->class_name() == Fl_Printer::class_id ) { ... }
Reimplemented from Fl_Paged_Device.

9.130.2.2 end_job()

void Fl_System_Printer::end_job ( 
    void ) [virtual]
To be called at the end of a print job.
Reimplemented from Fl_Paged_Device.

9.130.2.3 end_page()

int Fl_System_Printer::end_page ( 
    void ) [virtual]
To be called at the end of each page.
Returns
    0 if OK, non-zero if any error.
Reimplemented from Fl_Paged_Device.

9.130.2.4 margins()

void Fl_System_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>out</th>
<th>left</th>
<th>If non-null, *left is set to the left margin size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>top</td>
<td>If non-null, *top is set to the top margin size.</td>
</tr>
<tr>
<td>out</td>
<td>right</td>
<td>If non-null, *right is set to the right margin size.</td>
</tr>
<tr>
<td>out</td>
<td>bottom</td>
<td>If non-null, *bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

9.130.2.5 origin () [1/2]

void Fl_System_Printer::origin ( 
    int * x, 
    int * y ) [virtual]
Computes the page coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x</th>
<th>If non-null, *x is set to the horizontal page offset of graphics origin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>
Reimplemented from Fl_Paged_Device.

**9.130.2.6 origin() [2/2]**

```cpp
void Fl_System_Printer::origin ( 
    int x, 
    int y ) [virtual]
```

Sets the position in page coordinates of the origin of graphics functions.
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 page printable area. Origin() calls are not affected by `rotate()` calls. Successive `origin()` calls don't combine their effects.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>Horizontal position in page coordinates of the desired origin of graphics functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>y</td>
<td>Same as above, vertically.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

**9.130.2.7 printable_rect()**

```cpp
int Fl_System_Printer::printable_rect ( 
    int * w, 
    int * h ) [virtual]
```

Computes the width and height of the printable area of the page. Values are in the same unit as that used by FLTK drawing functions, are unchanged by calls to `origin()`, but are changed by `scale()` calls. Values account for the user-selected paper type and print orientation.

Returns

- 0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

**9.130.2.8 rotate()**

```cpp
void Fl_System_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. |

Reimplemented from Fl_Paged_Device.

**9.130.2.9 scale()**

```cpp
void Fl_System_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.

**9.130.2.10 start_job()**

```cpp
int Fl_System_Printer::start_job (  
  int pagecount,  
  int * frompage = NULL,  
  int * topage = NULL ); [virtual]
```

Starts a print job.

**Parameters**

| in  | pagecount | the total number of pages of the job (or 0 if you don’t know the number of pages) |
| out | frompage  | if non-null, *frompage is set to the first page the user wants printed |
| out | topage    | if non-null, *topage is set to the last page the user wants printed |

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

**9.130.2.11 start_page()**

```cpp
int Fl_System_Printer::start_page (  
  void ) [virtual]
```

Starts 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.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

**9.130.2.12 translate()**

```cpp
void Fl_System_Printer::translate (  
  int x,  
  int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.

This function is only useful after a rotate() call. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Paged_Device.

**9.130.2.13 untranslate()**

```cpp
void Fl_System_Printer::untranslate (  
  void ) [virtual]
```

Undoes the effect of a previous translate() call.

Reimplemented from Fl_Paged_Device.

The documentation for this class was generated from the following files:

- FL_Printer.H
- FL_Printer.cxx
9.131 Fl_Table Class Reference

A table of widgets or other content.
```cpp
#include <Fl_Table.H>
```

Inheritance diagram for Fl_Table:
```
Fl_Widget
   |   
   v   
Fl_Group
   |   
   v   
Fl_Table
   |   
   v   
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)
- void add (Fl_Widget *wgt)
- Fl_Widget *const *array ()
- 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.
- 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.
- 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 column scroll position to column 'col', 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)

- void **draw** (void)
  Draws the widget.

- void **end** ()

- int **find** (const FL_Widget &wgt) const
- int **find** (const FL_Widget *wgt) const
  The constructor for the 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** ()

- void **insert** (FL_Widget &wgt, FL_Widget *w2)
- void **insert** (FL_Widget &wgt, int 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)
- int **move_cursor** (int R, int C, int shiftselect)

- void **remove** (FL_Widget &wgt)

- void **resize** (int X, int Y, int W, int H)
  Changes the size of the Fl_Table, causing it to redraw.

- 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 row scroll position to 'row', 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()`
  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()`
  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.
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 the 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 addResizable (Fl_Widget &o)
  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget * & ddifice_kludge ()
  Returns a pointer to the array of children.
- virtual Fl_Group * & asGroup ()
  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 clipChildren ()
  Returns the current clipping mode.
- void clipChildren (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 *o) 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 formsEnd ()
  This is for forms compatibility only.
- void initSizes ()
  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &w, 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
  See void Fl_Group::resizable(Fl_Widget ∗box)
• void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget ∗box)
• void resizable (Fl_Widget ∗o)
  The resizable widget defines the resizing box for the group.
• 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 Fl_Window ∗as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
• 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, 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)
  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.
• const Fl_IMAGE *deimage () const
  Sets the image to use as part of the widget label.
• void deimage (Fl_IMAGE &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_IMAGE *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
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.

const Fl Image * image () const

void image (Fl Image &img)

Sets the image to use as part of the widget label.

void image (Fl Image *img)

Sets the image to use as part of the widget label.

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.

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.
• 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** (void ∗ v)
  Sets the user data for this widget.
• unsigned int **visible** () const
  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** () const
Returns whether a widget and all its parents are visible.

- int \texttt{w()} \texttt{const}
  
  Gets the widget width.

- \texttt{FL\_When when()} \texttt{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.

- \texttt{FL\_Window \star \texttt{window()} \texttt{const}}
  
  Returns a pointer to the nearest parent window up the widget hierarchy.

- int \texttt{x()} \texttt{const}
  
  Gets the widget position in its window.

- int \texttt{y()} \texttt{const}
  
  Gets the widget position in its window.

- virtual \texttt{\sim FL\_Widget()}
  
  Destroys the widget.

**Protected Types**

- enum \texttt{ResizeFlag}
  
  \begin{verbatim}
  RESIZE\_NONE = 0, RESIZE\_COL\_LEFT = 1, RESIZE\_COL\_RIGHT = 2, RESIZE\_ROW\_ABOVE = 3, RESIZE\_ROW\_BELOW = 4
  \end{verbatim}

**Protected Types inherited from FL\_Widget**

- enum
  
  \begin{verbatim}
  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, USERFLAG\_G3 = 1<<29, USERFLAG\_G2 = 1<<30, USERFLAG\_G1 = 1<<31
  \end{verbatim}

  \texttt{flags possible values enumeration.}

**Protected Member Functions**

- void \texttt{change\_cursor(FL\_Cursor newcursor)}

- long \texttt{col\_scroll\_position(int col)}

- TableContext \texttt{cursor2rowcol} (int R, int \&C, ResizeFlag \&resizeflag)

- void \texttt{damage\_zone(TableContext context, int X=0, int Y=0, int W=0, int H=0)}

  \texttt{Subclass should override this method to handle drawing the cells.}

- int \texttt{find\_cell(TableContext context, int R, int C, int \&X, int \&Y, int \&W, int \&H)}

- void \texttt{get\_bounds(TableContext context, int \&X, int \&Y, int \&W, int \&H)}

- int \texttt{handle(int e)}

  \texttt{Handles the specified event.}

- int \texttt{is\_fltk\_container()}

- void \texttt{recalc\_dimensions()}

- void \texttt{redraw\_range(int topRow, int botRow, int leftCol, int rightCol)}

- int \texttt{row\_col\_clamp(TableContext context, int R, int \&C)}

- long \texttt{row\_scroll\_position(int row)}

- void \texttt{table\_resized()}

- void \texttt{table\_scrolled()}

Generated by Doxygen
Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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.
- 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 *)
Protected Attributes

- int botrow
- int current_col
- int current_row
- Fl_Scrollbar * hscrollbar
- int leftcol
- int leftcol_scrollpos
- int rightcol
- int select_col
- int select_row
- Fl_Scroll * table
- int table_h
- int table_w
- int th
- int tw
- int tx
- int ty
- int toh
- int toprow
- int toprow_scrollpos
- int tow
- int tox
- int toy
- Fl_Scrollbar * vscrollbar
- int wih
- int wiw
- int wix
- int wiy

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 *cb, void *d)
  
  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.
9.131.1 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.

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>x()/y()/w()/h()</th>
<th>Fl_Table widget's outer dimension. The outer edge of the border of the Fl_Table. (Red in the diagram above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wix/wiy/wiw/wh</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/th</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 added the following license to the original distribution of Fl_Table. He 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. If used on its own, this is the license that applies:

Fl_Table License
December 16, 2002

The Fl_Table 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 Fl_Table 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 Fl_Table project - send all contributions to "erco at seriss dot com".

2. Widgets that are subclassed from Fl_Table widgets do not constitute a derivative work.

3. Static linking of applications and widgets to the Fl_Table 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 Fl_Table libraries, or link their applications or widgets against a user-supplied version of Fl_Table.

If you link the application or widget to a modified version of Fl_Table, then the changes to Fl_Table 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 Fl_Table license with programs that are linked to the Fl_Table library, nor do you have to identify the Fl_Table license in your program or documentation as required by section 6 of the LGPL.

However, programs must still identify their use of Fl_Table. 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 Fl_Table project http://seriss.com/people/erco/fltk/Fl_Table/

### 9.131.2 Member Enumeration Documentation

#### 9.131.2.1 TableContext

Enum Fl_Table::TableContext

The context bit flags for Fl_Table related callbacks.

Used in `draw_cell()`, `callback()`, etc.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</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 a page is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ENDPAGE</td>
<td>after a page is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ROW_HEADER</td>
<td>in the row header</td>
</tr>
<tr>
<td>CONTEXT_COL_HEADER</td>
<td>in the col header</td>
</tr>
<tr>
<td>CONTEXT_CELL</td>
<td>in one of the cells</td>
</tr>
<tr>
<td>CONTEXT_TABLE</td>
<td>in a dead zone of table</td>
</tr>
<tr>
<td>CONTEXT_RC_RESIZE</td>
<td>column or row being resized</td>
</tr>
</tbody>
</table>

### 9.131.3 Constructor & Destructor Documentation

#### 9.131.3.1 Fl_Table()

Fl_Table::Fl_Table ( 
    int X,    
    int Y,    

Generated by Doxygen
The constructor for the Fl_Table.
The constructor for the Fl_Table.
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

9.131.3.2 ~Fl_Table()
Fl_Table::~Fl_Table();
The destructor for the Fl_Table.
Destroys the table and its associated widgets.

9.131.4 Member Function Documentation

9.131.4.1 callback()

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, col will be 0.
If someone clicked on a column header, row will be 0.
callback_context() will return one of the following:

| Fl_Table::CONTEXT_ROW_HEADER | Someone clicked on a row header. Excludes resizing. |
| Fl_Table::CONTEXT_COL_HEADER | Someone clicked on a column header. Excludes resizing. |
| Fl_Table::CONTEXT_CELL       | Someone clicked on a cell. To receive callbacks for FL_RELEASE events, you must set when(FL_WHEN_RELEASE). |
| Fl_Table::CONTEXT_RC_RESIZE  | Someone is resizing rows/columns either interactively, or via the col_width() or row_height() API. Use is_interactive_resize() 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). |

class MyTable : public Fl_Table {
    [...] 
private:
    // 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

Generated by Doxygen
MyTable() {
    ...
    table.callback(event_callback, {void}*this); // setup callback
    table.when(FL_WHEN_CHANGED|FL_WHEN_RELEASE); // when to call it
}

9.131.4.2 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.

9.131.4.3 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.

9.131.4.4 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.

9.131.4.5 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);
    ...
}

9.131.4.6 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)

9.131.4.7 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.
9.131.4.8 col_header()

void Fl_Table::col_header ( 
    int flag ) [inline]
Enable or disable column headers.  
If changed, the table is redrawn.

9.131.4.9 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.

9.131.4.10 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.

9.131.4.11 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.

9.131.4.12 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.

9.131.4.13 draw()

void Fl_Table::draw ( 
    void ) [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 = &scroll; // scroll is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()  
Reimplemented from Fl_Group.

9.131.4.14 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. |

row and col will be set to the row and column number of the cell being drawn. In the case of row headers, col will be 0. In the case of column headers, row 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'.

```cpp
// 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);
        {  // draw box
            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);
        // draw cell
    }
    return;
```

// Generated by Doxygen


```c

default:
    return;

// NOTREACHED
}

9.131.4.15  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>in</th>
<th>row_top</th>
<th>Returns the top row of selection area</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>col_left</td>
<td>Returns the left column of selection area</td>
</tr>
<tr>
<td>in</td>
<td>row_bot</td>
<td>Returns the bottom row of selection area</td>
</tr>
<tr>
<td>in</td>
<td>col_right</td>
<td>Returns the right column of selection area</td>
</tr>
</tbody>
</table>

9.131.4.16  handle()

int Fl_Table::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.

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.
Reimplemented in Fl_Table_Row.
```
is_interactive_resize()

```cpp
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().

is_selected()

```cpp
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.

resize()

```cpp
void Fl_Table::resize ( int X, int Y, int W, int H ) [virtual]
```

Changes the size of the Fl_Table, causing it to redraw. Reimplemented from Fl_Group.

row_header()

```cpp
void Fl_Table::row_header ( int flag ) [inline]
```

Enables/disables showing the row headers. 1=enabled, 0=disabled. If changed, the table is redrawn.

row_height()

```cpp
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.

row_height_all()

```cpp
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.

row_resize()

```cpp
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.
9.131.4.24  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.

9.131.4.25  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.

9.131.4.26  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)

9.131.4.27  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

<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()

9.131.4.28  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.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>row_top</th>
<th>Top row of selection area</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>col_left</td>
<td>Left column of selection area</td>
</tr>
<tr>
<td>in</td>
<td>row_bot</td>
<td>Bottom row of selection area</td>
</tr>
<tr>
<td>in</td>
<td>col_right</td>
<td>Right column of selection area</td>
</tr>
</tbody>
</table>

**9.13.1.4.29 tab_cell_nav() [1/2]**

```c
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)

**9.13.1.4.30 tab_cell_nav() [2/2]**

```c
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. |

**9.13.1.4.31 table_box()**

```c
void Fl_Table::table_box (Fl_Boxtyle 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.

**9.13.1.4.32 top_row() [1/2]**

```c
int Fl_Table::top_row ( ) [inline]
```

Returns the current top row shown in the table.

This row may be partially obscured.

**9.13.1.4.33 top_row() [2/2]**

```c
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.

### 9.131.4.34 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.

```plaintext
leftcol : rightcol
toprow .. ------------------.
| VISIBLE | Table |
|         |      |
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`

### 9.131.4.35 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:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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:

- Fl_Table.H
- Fl_Table.hxx

### 9.132 Fl_Table_Row Class Reference

A table with row selection capabilities.

```cpp
#include <Fl_Table_H.hxx>
```

Inheritance diagram for `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 ()
  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)
  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
  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)
- void add (Fl_Widget ∗wgt)
- Fl_Widget ∗const ∗ array ()
- 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 column scroll position to column 'col', 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)**

- **void draw (void)**
  Draws the widget.

- **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 the 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 ()
• void insert (Fl_Widget &wgt, Fl_Widget *w2)
• void insert (Fl_Widget &wgt, int 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)
• int move_cursor (int R, int C, int shiftselect)
• void remove (Fl_Widget &wgt)
• void resize (int X, int Y, int W, int H)
  
  Changes the size of the Fl_Table, causing it to redraw.
• 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 row scroll position to 'row', 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 the `FL_Table`.

### Public Member Functions inherited from `FL_Group`

- `FL_Widget *& _ddfdesign_kludge()` (void)
  
  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.

- virtual `FL_Group * as_group()`
  
  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(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
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)

- **void** resizable (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.

- **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** Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

- **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, 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)
  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.

- **const Fl_Image** `deimage()` const
- **void deimage (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image** `image () const`
- **void image (Fl_Image &img)**
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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.
• 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.
• 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(void *v)**
  
  Sets the user data for this widget.

- **unsigned int visible()** const
  
  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()** 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.

Protected Member Functions

- **int find_cell(TableContext context, int R, int C, int &X, int &Y, int &W, int &H)**
- **int handle(int event)**
  
  Handles the specified event.

Protected Member Functions inherited from Fl_Table

- **void change_cursor(Fl_Cursor newcursor)**
- **long col_scroll_position(int col)**
- **TableContext cursor2rowcol(int &R, int &C, ResizeFlag &resizeflag)**
- **void damage_zone(int r1, int c1, int r2, int c2, int r3=0, int c3=0)**
- **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)**
- **void get_bounds(TableContext context, int &X, int &Y, int &W, int &H)**
- **int is_fltk_container()**
- **void recalc_dimensions()**
- **void redraw_range(int topRow, int botRow, int leftCol, int rightCol)**
- **int row_col_clamp(TableContext context, int &R, int &C)**
- **long row_scroll_position(int row)**
- **void table_resized()**
- **void table_scrolled()**
Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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 *cb, void *d)
  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_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_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  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 *)

Protected Attributes inherited from Fl_Table

- int botrow
- int current_col
- int current_row
- Fl_Scrollbar * hscrollbar
- int leftcol
- int leftcol_scrollpos
- int rightcol
- int select_col
- int select_row
- Fl_Scroll * table
- int table_h

Generated by Doxygen
9.132 Fl_Table_Row Class Reference

• int table_w
• int tih
• int tiw
• int tix
• int tiy
• int toh
• int toprow
• int toprow_scrollpos
• int tow
• int tox
• int toy
• Fl_Scrollbar * vscrollbar
• int wih
• int wiw
• int wix
• int wiy

9.132.1 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 Fl_Browser
with columns. Most methods of importance will be found in the Fl_Table widget, such as Fl_Table::rows() and
Fl_Table::cols().
To be useful it must be subclassed and at minimum the draw_cell() method must be overridden to provide the
content of the cells. This widget does not manage the cell's data content; it is up to the parent class's 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 Fl_Table::when().

9.132.2 Constructor & Destructor Documentation

9.132.2.1 Fl_Table_Row()

Fl_Table_Row::Fl_Table_Row ( int X, int Y, int W, int H, const char * l = 0 ) [inline]
The constructor for the Fl_Table_Row.
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

9.132.2.2 Fl_Table_Row()

Fl_Table_Row::~Fl_Table_Row ( ) [inline]
The destructor for the Fl_Table_Row.
Destroys the table and its associated widgets.

9.132.3 Member Function Documentation

9.132.3.1 clear()

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.

Generated by Doxygen
9.132.3.2 handle()

```cpp
int Fl_Table_Row::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.

**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_Table.

9.132.3.3 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)

9.132.3.4 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.

9.132.3.5 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.

9.132.3.6 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.

9.132.3.7 type()

```cpp
void Fl_Table_Row::type ( TableRowSelectMode val )
```
Sets the table selection mode.

- \texttt{Fl_Table_Row::SELECT_NONE} - \textit{No selection allowed}
- \texttt{Fl_Table_Row::SELECT_SINGLE} - \textit{Only single rows can be selected}
- \texttt{Fl_Table_Row::SELECT_MULTI} - \textit{Multiple rows can be selected}

The documentation for this class was generated from the following files:

- \texttt{Fl_Table_Row.H}
- \texttt{Fl_Table_Row.cxx}

\section*{9.133 Fl_Tabs Class Reference}

The \texttt{Fl_Tabs} widget is the "file card tabs" interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits.

\begin{verbatim}
#include <Fl_Tabs.H>
\end{verbatim}

Inheritance diagram for \texttt{Fl_Tabs}:

```
Fl_Widget
   
Fl_Group
   
Fl_Tabs
```

### Public Member Functions

- \texttt{void client_area (int &rx, int &ry, int &rw, int &rh, int tabh=0)}
  
  \textit{Returns the position and size available to be used by its children.}

- \texttt{Fl_Tabs (int, int, int, const char ∗=0)}

  \textit{Creates a new \texttt{Fl_Tabs} widget using the given position, size, and label string.}

- \texttt{int handle (int)}

  \textit{Handles the specified event.}

- \texttt{Fl_Widget ∗ push () const}

  \textit{Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.}

- \texttt{int push (Fl_Widget ∗)}

  \textit{This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.}

- \texttt{Fl_Widget ∗ value ()}

  \textit{GETS the currently visible widget/tab.}

- \texttt{int value (Fl_Widget ∗)}

  \textit{Sets the widget to become the current visible widget/tab.}

- \texttt{Fl_Widget ∗ which (int event_x, int event_y)}

  \textit{Return the widget of the tab the user clicked on at event_x/event_y.}

### Public Member Functions inherited from \texttt{Fl_Group}

- \texttt{Fl_Widget ∗ √ dddfdesign_kludge ()}

  \textit{This is for forms compatibility only.}

- \texttt{void add (Fl_Widget ∗)}

  \textit{The widget is removed from its current group (if any) and then added to the end of this group.}

- \texttt{void add (Fl_Widget ∗)}

  \textit{See void \texttt{Fl_Group::add(Fl_Widget ∗w)}}

\[\text{Generated by Doxygen}\]
• **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.

• virtual **Fl_Group** ∗ as_group ()
  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 ∗)
  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
  See void Fl_Group::resizable(Fl_Widget ∗box)

• **void** resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget ∗box)

• **void** resizable (Fl_Widget ∗o)
  The resizable widget defines the resizing box for the group.

• **void** resize (int, int, int, int)
  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 Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- 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, 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)
  
  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.

- const Fl_Image *deimage () const
  
  void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label.

- void do_callback ()
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, long arg)
  
  Calls the widget callback.

- void do_callback (Fl_Widget *o, void *arg=0)
  
  Calls the widget callback.

- 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.

- void hide ()
  
  Makes a widget invisible.

- virtual void hide ()
  
  virtual do_callback ()

  Calls the widget callback.

- Fl_Image *image ()
  
  Gets the image that is used as part of the widget label.

- const Fl_Image *image () const
  
  void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.

- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
• 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.
• 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.
Makes the widget visible.

- `void set_visible_focus ()`
  Enables keyboard focus navigation with this widget.

- `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 (void * v)`
  Sets the user data for this widget.

- `unsigned int visible () const`
  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 () 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** ()
  
  Draws the widget.

- void **redraw_tabs** ()

Protected Member Functions inherited from Fl_Group

- 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** ()
  
  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_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 *cb, void *d)
  
  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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }

  flags possible values enumeration.

9.133.1 Detailed Description

The Fl_Tabs widget is the “file card tabs” interface that allows you to put lots and lots of buttons and switches in a
panel, as popularized by many toolkits.

![Figure 9.37 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.
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:

```c++
// 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 tab#1..
    }
    grp1->end();
    Fl_Group *grp2 = new Fl_Group(20,30,280,170,"Tab2");
    { ..widgets that go in tab#2..
    }
    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_Tab, 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 9.38 Fl_Tabs Default Appearance](image)

**Highlighting The Selected Tab**

The selected "tab" can be highlighted further by setting the selection_color() of the Fl_Tab itself, e.g.

```c++
..
tabs = new Fl_Tabs(..);
tabs->selection_color(FL_DARK3);
..```

The result of the above looks like:

![Figure 9.39 Highlighting the selected tab](image)

**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*)**

```c
Fl_Tabs *tabs = (Fl_Tabs*)w;
// When tab changed, make sure it has same color as its group
tabs->selection_color( (tab->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:

![Figure 9.40 Fl_Tabs with uniform colors](image)

**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.

```c
tabs = new Fl_Tabs(...);
grp1 = new Fl_Group(...);
  ...
  grp2 = new Fl_Group(...);
  ...
tabs->end();
tabs->resizable(grp1); // keeps tab height constant
```

**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:**

Generated by Doxygen
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.

9.133.2 Constructor & Destructor Documentation

9.133.2.1 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.

9.133.3 Member Function Documentation

9.133.3.1 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>

Since

FLTK 1.3.0

9.133.3.2 draw()

```cpp
void Fl_Tabs::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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

9.133.3.3 handle()

```cpp
int Fl_Tabs::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.

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>

See also

`Fl_Event`

Reimplemented from `Fl_Group`.

9.133.3.4 push() [1/2]

```cpp
Fl_Widget * Fl_Tabs::push ( ) const [inline]
```

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*)`.
9.133.3.5 push() [2/2]

```c
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

click() 

9.133.3.6 value() [1/2]

```c
Fl_Widget * Fl_Tabs::value ( )
```
Gets the currently visible widget/tab.  
The value() is the first visible child (or the last child if none are visible) and this also hides any other children. This  
allows the tabs to be deleted, moved to other groups, and show()/hide() called without it screwing up.

9.133.3.7 value() [2/2]

```c
int Fl_Tabs::value (Fl_Widget * newvalue)
```
Sets the widget to become the current visible widget/tab.  
Setting the value hides all other children, and makes this one visible, if it is really a child.

Returns

1 if there was a change (new value different from previous),  
0 if there was no change (new value already set)

9.133.3.8 which()

```c
Fl_Widget * Fl_Tabs::which (int event_x, int event_y)
```
Return the widget of the tab the user clicked on at event_x/event_y.  
This is used for event handling (clicks) and by fluid to pick tabs.

Returns

The child widget of the tab the user clicked on, or  
0 if there are no children or if the event is outside of the tabs area.

The documentation for this class was generated from the following files:

- Fl_Tabs.H
- Fl_Tabs.cxx

9.134 Fl_Text_Buffer Class Reference

This class manages Unicode text displayed in one or more Fl_Text_Display widgets.  
#include <Fl_Text_Buffer.H>

Public Member Functions

- void add_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)  
  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.

- **char ∗ address (int pos)**
  
  Convert a byte offset in buffer into a memory address.

- **const char ∗ address (int pos) const**
  
  Convert a byte offset in buffer into a memory address.

- **void append (const char ∗) const**
  
  Appends the text string to the end of the buffer.

- **int appendfile (const char ∗file, int buflen=128 ∗1024) const**
  
  Appends the named file to the end of the buffer.

- **char byte_at (int pos) const**
  
  Returns the raw byte at the specified position pos in the buffer.

- **void call_modify_callbacks () const**
  
  Calls all modify callbacks that have been registered using the add_modify_callback() method.

- **void call_predelete_callbacks () 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.

- **void canUndo (char flag=1)**
  
  Lets the undo system know if we can undo changes.

- **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 the highlighted text.

- **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)**
  
  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.

- **int length () const**
  
  Returns the number of bytes in the buffer.

- **int line_end (int pos) const**
  
  Returns the raw byte at the specified position pos in the buffer.
Finds and returns the position of the end of the line containing position \texttt{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).

\begin{itemize}
\item \texttt{int line_start (int pos) const}
\hspace{1em}Returns the position of the start of the line containing position \texttt{pos}.
\item \texttt{char \* line_text (int pos) const}
\hspace{1em}Returns the text from the entire line containing the specified character position.
\item \texttt{int loadfile (const char \* file, int buflen=128 \*1024)}
\hspace{1em}Loads a text file into the buffer.
\item \texttt{int next_char (int ix) const}
\hspace{1em}Returns the index of the next character.
\item \texttt{int next_char \_clipped (int ix) const}
\item \texttt{int outputfile (const char \* file, int start, int end, int buflen=128 \*1024)}
\hspace{1em}Writes the specified portions of the text buffer to a file.
\item \texttt{int prev_char (int ix) const}
\hspace{1em}Returns the index of the previous character.
\item \texttt{int prev_char \_clipped (int ix) const}
\item \texttt{FL\_Text\_Selection \* primary_selection ()}
\hspace{1em}Returns the primary selection.
\item \texttt{const FL\_Text\_Selection \* primary_selection () const}
\hspace{1em}Returns the primary selection.
\item \texttt{void remove (int start, int end)}
\hspace{1em}Deletes a range of characters in the buffer.
\item \texttt{void remove\_modify\_callback (FL\_Text\_Modify\_Cb bufModifiedCB, void \* cbArg)}
\hspace{1em}Removes a modify callback.
\item \texttt{void remove\_predelete\_callback (FL\_Text\_Predelete\_Cb predelCB, void \* cbArg)}
\hspace{1em}Removes a callback routine \texttt{bufPreDeleteCB} associated with argument \texttt{cbArg} to be called before text is deleted from the buffer.
\item \texttt{void remove\_secondary\_selection ()}
\hspace{1em}Removes the text from the buffer corresponding to the secondary text selection object.
\item \texttt{void remove\_selection ()}
\hspace{1em}Removes the text in the primary selection.
\item \texttt{void replace (int start, int end, const char \* text)}
\hspace{1em}Deletes the characters between \texttt{start} and \texttt{end}, and inserts the null-terminated string \texttt{text} in their place in the buffer.
\item \texttt{void replace\_secondary\_selection (const char \* text)}
\hspace{1em}Replaces the text from the buffer corresponding to the secondary text selection object with the new string \texttt{text}.
\item \texttt{void replace\_selection (const char \* text)}
\hspace{1em}Replaces the text in the primary selection.
\item \texttt{int rewind\_lines (int startPos, int nLines)}
\hspace{1em}Finds and returns the position of the first character of the line \texttt{nLines} backwards from \texttt{startPos} (not counting the character pointed to by \texttt{startpos} if that is a newline) in the buffer.
\item \texttt{int savefile (const char \* file, int buflen=128 \*1024)}
\hspace{1em}Saves a text file from the current buffer.
\item \texttt{int search\_backward (int startPos, const char \* searchString, int \* foundPos, int matchCase=0) const}
\hspace{1em}Search backwards in buffer for string \texttt{searchString}, starting with the character at \texttt{startPos}, returning the result in \texttt{foundPos}.
\item \texttt{int search\_forward (int startPos, const char \* searchString, int \* foundPos, int matchCase=0) const}
\hspace{1em}Search forwards in buffer for string \texttt{searchString}, starting with the character \texttt{startPos}, and returning the result in \texttt{foundPos}.
\item \texttt{void secondary\_select (int start, int end)}
\hspace{1em}Selects a range of characters in the secondary selection.
\item \texttt{int secondary\_selected ()}
\end{itemize}
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)`
  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)`
  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 startPos 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.

- `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

- 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)
  
  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 mNPredeleteProcs
  number of pre-delete procs attached
• void ** mPredeleteCbArgs
  caller argument for pre-delete proc above
• Fl_Text_Predelete_CB * mPredeleteProcs
  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_Selection mSecondary
  highlighted areas
• int mTabDist
  equiv.

9.134.1 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 unsiged 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 http://www.nedit.org/.

9.134.2 Constructor & Destructor Documentation

9.134.2.1 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>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedSize</td>
<td>use this to avoid unnecessary re-allocation if you know exactly how much the buffer will need to hold</td>
</tr>
<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>

9.134.3 Member Function Documentation

9.134.3.1 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:

```c
typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,  
  int nRestyled, const char* deletedText,  
  void* cbArg);
```
9.134.3.2 address() [1/2]

```cpp
cchar * 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

9.134.3.3 address() [2/2]

```cpp
cconst 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

9.134.3.4 append()

```cpp
cvoid Fl_Text_Buffer::append (const char * t) [inline]
```

Appends the text string to the end of the buffer.

**Parameters**

| t | UTF-8 encoded and null terminated text |

9.134.3.5 appendfile()

```cpp
cint 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()`.

9.134.3.6 byte_at()

```cpp
cchar 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 |
9.134.3.7 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

9.134.3.8 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 |

9.134.3.9 countDisplayed_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.

9.134.3.10 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.

9.134.3.11 findchar_backward()

int Fl_Text_Buffer::findchar_backward ( int startPos,
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**

| startPos | byte offset to start position |
| searchChar | UCS-4 character that we want to find |
| foundPos | byte offset where the character was found |

**Returns**

1 if found, 0 if not

### 9.134.3.12 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**

| startPos | byte offset to start position |
| searchChar | UCS-4 character that we want to find |
| foundPos | byte offset where the character was found |

**Returns**

1 if found, 0 if not

### 9.134.3.13 highlight()

```cpp
int Fl_Text_Buffer::highlight ( ) [inline]
```

Returns the highlighted text.
When you are done with the text, free it using the free() function.

### 9.134.3.14 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.

### 9.134.3.15 insert()

```cpp
void Fl_Text_Buffer::insert ( 
    int pos,
    const char * text )
```
Inserts null-terminated string \texttt{text} at position \texttt{pos}.

**Parameters**

| \texttt{pos} | insertion position as byte offset (must be UTF-8 character aligned) |
| \texttt{text} | UTF-8 encoded and nul terminated text |

**9.134.3.16 insert()**

```cpp
int Fl_Text_Buffer::insert_ (int pos, const char * text) [protected]
```

Internal (non-redisplaying) version of \texttt{insert()}. Returns the length of text inserted (this is just \texttt{strlen(text)}, however this calculation can be expensive and the length will be required by any caller who will continue on to call redisplay). \texttt{pos} must be contiguous with the existing text in the buffer (i.e. not past the end).

Returns the number of bytes inserted.

**9.134.3.17 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 (\texttt{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 \texttt{Fl_Text_Buffer} widget will contain data transcoded to UTF-8. By default, the message \texttt{Fl_Text_Buffer::file_encoding_warning_message} will warn the user about this.

See also \texttt{input_file_was_transcoded} and \texttt{transcoding_warning_action}.

**9.134.3.18 length()**

```cpp
int Fl_Text_Buffer::length ( ) const [inline]
```

Returns the number of bytes in the buffer.

Returns

size of text in bytes

**9.134.3.19 line_end()**

```cpp
int Fl_Text_Buffer::line_end (int pos) const
```

Finds and returns the position of the end of the line containing position \texttt{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

\texttt{pos} \hspace{1em} \text{byte index into buffer}

Returns

byte offset to line end

\textbf{9.134.3.20} \texttt{line_start()}

\begin{verbatim}
int Fl_Text_Buffer::line_start(int pos) const
\end{verbatim}

Returns the position of the start of the line containing position \texttt{pos}.

Parameters

\texttt{pos} \hspace{1em} \text{byte index into buffer}

Returns

byte offset to line start

\textbf{9.134.3.21} \texttt{line_text()}

\begin{verbatim}
char * Fl_Text_Buffer::line_text(int pos) const
\end{verbatim}

Returns the text from the entire line containing the specified character position.
When you are done with the text, free it using the \texttt{free()} function.

Parameters

\texttt{pos} \hspace{1em} \text{byte index into buffer}

Returns

copy of UTF-8 text, must be free'd

\textbf{9.134.3.22} \texttt{loadfile()}

\begin{verbatim}
int Fl_Text_Buffer::loadfile(const char * file, int buflen = 128*1024) [inline]
\end{verbatim}

Loads a text file into the buffer.
See also \texttt{insertfile()}.

\textbf{9.134.3.23} \texttt{next_char()}

\begin{verbatim}
int Fl_Text_Buffer::next_char(int ix) const
\end{verbatim}

Returns the index of the next character.

Parameters

\texttt{ix} \hspace{1em} \text{index to the current character}
9.134.3.24  outputfile()

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)

9.134.3.25  prev_char()

int Fl_Text_Buffer::prev_char (  
    int ix ) const

Returns the index of the previous character.

Parameters

ix  |  index to the current character

9.134.3.26  remove()

void Fl_Text_Buffer::remove (  
    int start,  
    int end )

Deletes a range of characters in the buffer.

Parameters

start  |  byte offset to first character to be removed
end    |  byte offset to character after last character to be removed

9.134.3.27  remove_()

void Fl_Text_Buffer::remove_ (  
    int start,  
    int end ) [protected]

Internal (non-redisplaying) version of remove().

Removes the contents of the buffer between start and end (and moves the gap to the site of the delete).

9.134.3.28  replace()

void Fl_Text_Buffer::replace (  
    int start,  
    int end,  
    const char * text )
Deletes the characters between `start` and `end`, and inserts the null-terminated string `text` in their place in the buffer.

Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>byte offset to first character to be removed and new insert position</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>byte offset to character after last character to be removed</td>
</tr>
<tr>
<td>text</td>
<td>UTF-8 encoded and nul terminated text</td>
</tr>
</tbody>
</table>

### 9.134.3.29 rewind_lines()

```cpp
int Fl_Text_Buffer::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. `nLines == 0` means find the beginning of the line.

### 9.134.3.30 savefile()

```cpp
int Fl_Text_Buffer::savefile (const char * file, int buflen = 128*1024 ) [inline]
```

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)`

### 9.134.3.31 search_backward()

```cpp
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

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>UTF-8 string that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the string was found</td>
</tr>
<tr>
<td>matchCase</td>
<td>if set, match character case</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

1 if found, 0 if not

9.134.3.32 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

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>UTF-8 string that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the string was found</td>
</tr>
<tr>
<td>matchCase</td>
<td>if set, match character case</td>
</tr>
</tbody>
</table>

Returns

1 if found, 0 if not

9.134.3.33 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.

9.134.3.34 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.

9.134.3.35 skip_displayed_characters()

int Fl_Text_Buffer::skip_displayed_characters {
    int lineStartPos,
    int nChars )

Count forward from buffer position startPos 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
9.134.3.36  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.

9.134.3.37  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

9.134.3.38  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. |

9.134.3.39  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

| start | byte offset to first character |
| end   | byte offset after last character in range |

Returns
newly allocated text buffer - must be free'd, text is UTF-8

9.134.3.40  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 |
Returns

byte offset to word end

9.134.3.41 word_start()

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

9.134.4 Member Data Documentation

9.134.4.1 file_encoding_warning_message

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.

9.134.4.2 mTabDist

int Fl_Text_Buffer::mTabDist  [protected]  
equiv.  
number of characters in a tab

9.134.4.3 transcoding_warning_action

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

9.135 Fl_Text_Display Class Reference

Rich text display widget.

#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()

  • 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.

• 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.
• virtual int handle (int e)
  
  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 linenumber_font () const
  
  Return the font used for line numbers (if enabled).
• void linenumber_font (Fl_Font val)
  
  Set the font used for line numbers (if enabled).
• const char ∗linenumber_format () 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.
• void redisplay_range (int start, int end)
  Marks text from start to end as needing a redraw.
• virtual void resize (int X, int Y, int W, int H)
  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_width () const
  Gets the width/height of the scrollbars.
• void scrollbar_width (int W)
  Sets the width/height of the scrollbars.
• 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 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 * & dddfdesign_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.

- virtual Fl_Group * & as_group ()
  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
  See void Fl_Group::resizable(Fl_Widget *box)
• void resizable (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget *box)
• void resizable (Fl_Widget *o)
  The resizable widget defines the resizing box for the group.
• 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 Fl_Gl_Window * as_gl_window ()
  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.
• 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, 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)
  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.
• const Fl_Image ∗ deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void **deimage** (**Fl__Image** img)
  
  Sets the image to use as part of the widget label.

• void **do_callback** ()
  
  Calls the widget callback.

• void **do_callback** (**Fl__Widget** w, long arg)
  
  Calls the widget callback.

• void **do_callback** (**Fl__Widget** w, void **arg=**0)
  
  Calls the widget callback.

• 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.

• const **Fl__Image** * **image** () const
  
  Sets the image to use as part of the widget label.

• void **image** (**Fl__Image** &img)
  
  Sets the image to use as part of the widget label.

• void **image** (**Fl__Image** img)
  
  Sets the image to use as part of the widget label.

• 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.

• 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.

• 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 (void *v)
Sets the user data for this widget.

- unsigned int visible() const
  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() 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 }

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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions

- void absolute_top_line_number(int oldFirstChar)
  Line numbering stuff, currently unused.
- void calc_last_char()
  Update last display character index.
- void calc_line_starts(int startLine, int endLine)
  Update the line start arrays.
- 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.

- **virtual void draw ()**
  
  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.

- **int get_absolute_top_line_number () const**
  
  Line numbering stuff, currently unused.

- **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**
  
  Line numbering stuff, currently unused.

- **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 ()**
  
  Line numbering stuff, probably unused.
• 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 \n or by line wrapping.
• void wrapped_line_counter (Fl_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIs← LineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLast← LineMissingNewLine=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

• 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_Fboxtype 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_Fboxtype 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 ()
  draws a focus rectangle around the widget
• void draw_focus (FL_Fboxtype t, int x, int y, int w, int h) const

Generated by Doxygen
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)
  Callbacks for drag or valueChanged on horizontal scrollbar.

- static void **scroll_timer_cb** (void *cbArg)
  Timer callback for scroll events.

- static void **v_scrollbar_cb** (Fl_Scrollbar *w, Fl_Text_Display *d)
  Callbacks 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_Align** **linenumber_align_**
- **Fl_Color** **linenumberbgcolor_**
- **Fl_Color** **linenumber_fgcolor_**
- **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 * mhHighlightCBArg
• int mHorizOffset
• int mHorizOffsetHint
• Fl_Scrollbar * mHScrollBar
• int mLChar
• 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_
• int shortcut_
• struct {
    int h
    int w
    int x
    int y
} text_area

• Fl_Color textcolor_
• Fl_Font textfont_
• Fl_Fontsize textsize_

Friends

• void fl_text_drag_me (int pos, Fl_Text_Display *d)
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∗ cb, void∗ d)
  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.

9.135.1 Detailed Description

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()`, `linenumber_fgcolor()`, `linenumberbgcolor()`, `linenumber_align()`, `linenumber_format()`

Note that other features may be available via `Fl_Text_Editor` and `Fl_Text_Buffer` classes.

**Note**

Line numbers were added in 1.3.3. To avoid breaking ABI, many of its options are read only. To adjust these features in 1.3.x, you must build FLTK with FLTK_ABI_VERSION set to 10303 or higher.

### 9.135.2 Member Enumeration Documentation

#### 9.135.2.1 anonymous enum

<table>
<thead>
<tr>
<th>Enumerators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NORMAL_CURSOR</code></td>
<td>I-beam.</td>
</tr>
<tr>
<td><code>CARET_CURSOR</code></td>
<td>caret under the text</td>
</tr>
<tr>
<td><code>DIM_CURSOR</code></td>
<td>dim I-beam</td>
</tr>
<tr>
<td><code>BLOCK_CURSOR</code></td>
<td>unfilled box under the current character</td>
</tr>
<tr>
<td><code>HEAVY_CURSOR</code></td>
<td>thick I-beam</td>
</tr>
<tr>
<td><code>SIMPLE_CURSOR</code></td>
<td>as cursor as <code>Fl_Input</code> cursor</td>
</tr>
</tbody>
</table>

#### 9.135.2.2 anonymous enum

<table>
<thead>
<tr>
<th>Enumerators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WRAP_NONE</code></td>
<td>don’t wrap text at all</td>
</tr>
<tr>
<td><code>WRAP_AT_COLUMN</code></td>
<td>wrap text at the given text column</td>
</tr>
<tr>
<td><code>WRAP_AT_PIXEL</code></td>
<td>wrap text at a pixel position</td>
</tr>
<tr>
<td><code>WRAP_AT_BOUNDS</code></td>
<td>wrap text so that it fits into the widget width</td>
</tr>
</tbody>
</table>
9.135.3 Constructor & Destructor Documentation

9.135.3.1 Fl_Text_Display()

Fl_Text_Display::Fl_Text_Display (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0  )

Creates a new text display widget.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y, W, H</td>
</tr>
<tr>
<td>l</td>
</tr>
</tbody>
</table>

9.135.3.2 ~Fl_Text_Display()

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.

9.135.4 Member Function Documentation

9.135.4.1 absolute_top_line_number()

void Fl_Text_Display::absolute_top_line_number (  
    int oldFirstChar  ) [protected]

Re-calculate absolute top line number for a change in scroll position.

9.135.4.2 buffer() [1/3]

Fl_Text_Buffer * Fl_Text_Display::buffer ( ) const [inline]

Sets the current text buffer associated with the text widget.
Multiple text widgets can be associated with the same text buffer.

Returns

    current text buffer

9.135.4.3 buffer() [2/3]

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 |

9.135.4.4 buffer() [3/3]

void Fl_Text_Display::buffer (  
    Fl_Text_Buffer * buf  )
Attach a text buffer to display, replacing the current buffer (if any)

Parameters

| buf | attach this text buffer |

9.135.4.5 buffer_modified_cb()

```c
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>pos</th>
<th>starting index of modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>nInserted</td>
<td>number of bytes we inserted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td>nDeleted</td>
<td>number of bytes deleted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td>nRestyled</td>
<td>??</td>
</tr>
<tr>
<td>deletedText</td>
<td>this is what was removed, must not be NULL if nDeleted is set</td>
</tr>
<tr>
<td>cbArg</td>
<td>&quot;this&quot; pointer for static callback function</td>
</tr>
</tbody>
</table>

9.135.4.6 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.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>starting index of deletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeleted</td>
<td>number of bytes we will delete (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td>cbArg</td>
<td>&quot;this&quot; pointer for static callback function</td>
</tr>
</tbody>
</table>

9.135.4.7 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.
9.135.4.8 calc_line_starts()

void Fl_Text_Display::calc_line_starts (  
    int startLine,  
    int endLine   ) [protected]

Update the line start arrays.  
Scan through the text in the "textD"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.

Parameters

| startLine,endLine | range of lines to scan as line numbers |

9.135.4.9 clear_rect()

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>

9.135.4.10 col_to_x()

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

9.135.4.11 count_lines()

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

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to first character</th>
</tr>
</thead>
<tbody>
<tr>
<td>endPos</td>
<td>index after last character</td>
</tr>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

Returns

number of lines

9.135.4.12 cursor_color() [1/2]

Fl_Color Fl_Text_Display::cursor_color ( ) const [inline]

Gets the text cursor color.

Returns

cursor color

9.135.4.13 cursor_color() [2/2]

void Fl_Text_Display::cursor_color ( Fl_Color n ) [inline]

Sets the text cursor color.

Parameters

| n | new cursor color |

9.135.4.14 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:

- Fl_Text_Display::NORMAL_CURSOR - Shows an I beam.
- Fl_Text_Display::CARET_CURSOR - Shows a caret under the text.
- Fl_Text_Display::DIM_CURSOR - Shows a dimmed I beam.
- Fl_Text_Display::BLOCK_CURSOR - Shows an unfilled box around the current character.
- Fl_Text_Display::HEAVY_CURSOR - Shows a thick I beam.

This call also switches the cursor on and may trigger a redraw.

Parameters

| style | new cursor style |

9.135.4.15 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.

**Todo** Unicode?

### 9.135.4.16 draw()

```cpp
def draw():
    # [protected], [virtual]
    # Draw the widget.
    # This function tries to limit drawing to smaller areas if possible.
    # Reimplemented from Fl_Group.
```

### 9.135.4.17 draw_cursor()

```cpp
def draw_cursor(x, y):
    # [protected]
    # Draw a cursor with top center at X, Y.
```

**Parameters**

| X, Y | cursor position in pixels |

### 9.135.4.18 draw_line_numbers()

```cpp
def draw_line_numbers(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. |

### 9.135.4.19 draw_range()

```cpp
def draw_range(startpos, 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 |
9.135.4.20 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>style</code></td>
<td>index into style lookup table</td>
</tr>
<tr>
<td><code>X,Y</code></td>
<td>drawing origin</td>
</tr>
<tr>
<td><code>toX</code></td>
<td>rightmost position if this is a fill operation</td>
</tr>
<tr>
<td><code>string</code></td>
<td>text if this is a drawing operation</td>
</tr>
<tr>
<td><code>nChars</code></td>
<td>number of characters to draw</td>
</tr>
</tbody>
</table>

9.135.4.21 draw_text()

```cpp
void Fl_Text_Display::draw_text (  
    int left,  
    int top,  
    int width,  
    int height ) [protected]
```

Refresh a rectangle of the text display.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>left,top</code></td>
<td>are in coordinates of the text drawing window.</td>
</tr>
<tr>
<td><code>width,height</code></td>
<td>size in pixels</td>
</tr>
</tbody>
</table>

9.135.4.22 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>visLineNum</code></td>
<td>index of line in the visible line number lookup</td>
</tr>
<tr>
<td><code>leftClip,rightClip</code></td>
<td>pixel position of clipped area</td>
</tr>
<tr>
<td><code>leftCharIndex,rightCharIndex</code></td>
<td>index into line of segment that we want to draw</td>
</tr>
</tbody>
</table>
9.135.4.23 empty_vlines()

```cpp
text_list::empty_vlines ( ) const [protected]
```

Return true if there are lines visible with no corresponding buffer text.

Returns

1 if there are empty lines

9.135.4.24 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 `startpos` to `endpos`) 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?

9.135.4.25 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

| startPos | ??? |
| startPosIsLineStart | |
| lineEnd | |
| nextLineStart | |

9.135.4.26 find_wrap_range()

```cpp
void Fl_Text_Display::find_wrap_range (const char *deletedText, int pos, int nInserted, int nDeleted, int *modRangeStart, int *modRangeEnd,
```

Generated by Doxygen
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

- `deletedText`
- `pos`
- `nInserted`
- `nDeleted`
- `modRangeStart`
- `modRangeEnd`
- `linesInserted`
- `linesDeleted`

#### 9.135.4.27 find_x()

```cpp
int Fl_Text_Display::find_x (const char * s, int len, int style, int x) const [protected]
```

Find the index of the character that lies at the given x position.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s</code></td>
<td>UTF-8 text string</td>
</tr>
<tr>
<td><code>len</code></td>
<td>length of string</td>
</tr>
<tr>
<td><code>style</code></td>
<td>index into style lookup table</td>
</tr>
<tr>
<td><code>x</code></td>
<td>position in pixels</td>
</tr>
</tbody>
</table>

**Returns**

index into buffer

#### 9.135.4.28 get_absolute_top_line_number()

```cpp
int Fl_Text_Display::get_absolute_top_line_number () const [protected]
```

Line numbering stuff, currently unused.

Returns the absolute (non-wrapped) line number of the first line displayed. Returns 0 if the absolute top line number
is not being maintained.

#### 9.135.4.29 handle()

```cpp
int Fl_Text_Display::handle (int e) [virtual]
```

Event handling.

Reimplemented from `Fl_Group`.

Reimplemented in `Fl_Text_Editor`.
9.135.4.30 handle_vline()

```cpp
int Fl_Text_Display::handle_vline ( 
    int mode, 
    int lineStartPos, 
    int len, 
    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>in</th>
<th>mode</th>
<th>DRAW_LINE, GET_WIDTH, FIND_INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>lineStartPos</td>
<td>index of first character</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>size of string in bytes</td>
</tr>
<tr>
<td>in</td>
<td>leftChar, rightChar</td>
<td>drawing position</td>
</tr>
<tr>
<td>in</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>bottomClip, leftClip, rightClip</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**

| DRAW_LINE | index of last drawn character |
| GET_WIDTH | width in pixels of text segment if we would draw it |
| FIND_INDEX | index of character at given x position in window coordinates |
| FIND_INDEX_FROM_ZERO | index of character at given x position without scrolling and widget offsets |

**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

9.135.4.31 highlight_data()

```cpp
void Fl_Text_Display::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.
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 extendRangeForStyleMods 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.

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>and optional argument for the callback above, usually a pointer to the Text Display.</td>
</tr>
</tbody>
</table>

9.135.4.32 in_selection()

```cpp
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

9.135.4.33 insert()

```cpp
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 BufInsert 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. |

9.135.4.34 insert_position() [1/2]

```cpp
int Fl_Text_Display::insert_position ( ) const [inline]
```

Gets the position of the text insertion cursor for text display.

Returns

insert position index into text buffer
### 9.135.4.35 insert_position() [2/2]

```cpp
text
    void Fl_Text_Display::insert_position (int newPos)
```

Sets the position of the text insertion cursor for text display. Move the insertion cursor in front of the character at `newPos`. This function may trigger a redraw.

**Parameters**
- `newPos`: new caret position

### 9.135.4.36 line_end()

```cpp
text
    int Fl_Text_Display::line_end (int startPos, bool startPosIsLineStart) const
```

Returns the end of a line. Same as `BufEndOfLine`, 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 `true` for `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

### 9.135.4.37 line_start()

```cpp
text
    int Fl_Text_Display::line_start (int pos) const
```

Return the beginning of a line. Same as `BufStartOfLine`, but returns the character after last wrap point rather than the last newline.

**Parameters**
- `pos`: index to starting character

Returns
- new position as index

### 9.135.4.38 linenumber_align()

```cpp
text
    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 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)

9.135.4.39 linenumberbgcolor()

void Fl_Text_Display::linenumberbgcolor ( 
   Fl_Color val )

Set the background color used for line numbers (if enabled).

Version

1.3.3 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)

9.135.4.40 linenumberfgcolor()

void Fl_Text_Display::linenumberfgcolor ( 
   Fl_Color val )

Set the foreground color used for line numbers (if enabled).

Version

1.3.3 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)

9.135.4.41 linenumberfont()

void Fl_Text_Display::linenumberfont ( 
   Fl_Font val )

Set the font used for line numbers (if enabled).

Version

1.3.3 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)

9.135.4.42 linenumberformat()

void Fl_Text_Display::linenumberformat ( 
   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 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)

9.135.4.43 linenumbersize()

void Fl_Text_Display::linenumbersize ( 
   Fl_Fontsize val )

Set the font size used for line numbers (if enabled).

Version

1.3.3 ABI feature (ignored in 1.3.x unless FLTK_ABI_VERSION is 10303 or higher)
9.135.4.44 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)

9.135.4.45 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

9.135.4.46 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 TextDPosToLineAndCol). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

9.135.4.47 maintaining_absolute_top_line_number()

```cpp
int Fl_Text_Display::maintaining_absolute_top_line_number ( ) const [protected]
```

Line numbering stuff, currently unused.
Return true if a separate absolute top line number is being maintained (for displaying line numbers or showing in the statistics line).

9.135.4.48 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`
9.135.4.49 measure_proportional_character()

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

<table>
<thead>
<tr>
<th>s</th>
<th>text string</th>
</tr>
</thead>
<tbody>
<tr>
<td>xPix</td>
<td>x pixel position needed for calculating tab widths</td>
</tr>
<tr>
<td>pos</td>
<td>offset within string</td>
</tr>
</tbody>
</table>

Returns

width of character in pixels

9.135.4.50 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

9.135.4.51 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

9.135.4.52 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
9.135.4.53 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

9.135.4.54 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

9.135.4.55 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

newTopLineNum | index into buffer

9.135.4.56 overstrike()

void Fl_Text_Display::overstrike ( const char ∗ text )
Replaces text at the current insert position.

Parameters

text | new text in UTF-8 encoding

Todo  Unicode? Find out exactly what we do here and simplify.

9.135.4.57 position_style()

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.

Parameters

<table>
<thead>
<tr>
<th>lineStartPos</th>
<th>beginning of this line</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineLen</td>
<td>number of bytes in line</td>
</tr>
<tr>
<td>lineIndex</td>
<td>position of character within line</td>
</tr>
</tbody>
</table>

Returns

style for the given character

9.135.4.58 position_to_line()

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. Returns 0 if the line is not displayed.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>lineNum</td>
<td>??</td>
</tr>
</tbody>
</table>

Returns

??

Todo  What does this do?

9.135.4.59 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.
### 9.135.4.60 position_to_xy()

```cpp
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>out</td>
<td>X,Y</td>
</tr>
<tr>
<td></td>
<td>pixel position of character on screen</td>
</tr>
</tbody>
</table>

**Returns**

0 if character vertically out of view, X & Y positions otherwise

### 9.135.4.61 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>startpos</th>
<th>index of first character needing redraw</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpos</td>
<td>index after last character needing redraw</td>
</tr>
</tbody>
</table>

### 9.135.4.62 reset_absolute_top_line_number()

```cpp
void Fl_Text_Display::reset_absolute_top_line_number ( ) [protected]
```

Line numbering stuff, probably unused. 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.

### 9.135.4.63 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 lines visible and of all scrollbar sizes.

**Parameters**

| X,Y,W,H | new position and size of this widget |

Reimplemented from Fl_Group.
9.135.4.64  rewind_lines()

int Fl_Text_Display::rewind_lines (  
    int startPos,  
    int nLines )

Skip a number of lines back.  
Same as BufCountBackwardNLines, but takes into account line breaks when wrapping is turned on.

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 back</td>
</tr>
</tbody>
</table>

Returns

new position as index

9.135.4.65  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.

9.135.4.66  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

9.135.4.67  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.
9.135.4.68 scrollbar_align() [1/2]

Fl_Align Fl_Text_Display::scrollbar_align ( ) const [inline]

Gets the scrollbar alignment type.

Returns

scrollbar alignment

9.135.4.69 scrollbar_align() [2/2]

void Fl_Text_Display::scrollbar_align ( Fl_Align a ) [inline]

Sets the scrollbar alignment type.

Parameters

<table>
<thead>
<tr>
<th>a</th>
<th>new scrollbar alignment</th>
</tr>
</thead>
</table>

9.135.4.70 scrollbar_width() [1/2]

int Fl_Text_Display::scrollbar_width ( ) const [inline]

Gets the width/height of the scrollbars.

Returns

width of scrollbars

9.135.4.71 scrollbar_width() [2/2]

void Fl_Text_Display::scrollbar_width ( int W ) [inline]

Sets the width/height of the scrollbars.

Parameters

<table>
<thead>
<tr>
<th>W</th>
<th>width of scrollbars</th>
</tr>
</thead>
</table>

9.135.4.72 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

9.135.4.73 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

### 9.135.4.74 show_cursor()

```cpp
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).

### 9.135.4.75 show_insert_position()

```cpp
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()

### 9.135.4.76 skip_lines()

```cpp
int Fl_Text_Display::skip_lines ( int startPos, int nLines, bool startPosIsLineStart )
```

Skip a number of lines forward.
Same as BufCountForwardNLines, 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

### 9.135.4.77 string_width()

```cpp
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

<table>
<thead>
<tr>
<th>string</th>
<th>the text</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>number of bytes in string</td>
</tr>
<tr>
<td>style</td>
<td>index into style table</td>
</tr>
</tbody>
</table>
Returns

width of text segment in pixels

9.135.4.78 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

9.135.4.79 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

9.135.4.80 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

9.135.4.81 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

9.135.4.82 textsize() [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

9.135.4.83 textsize() [2/2]

void Fl_Text_Display::textsize ( Fl_Fontsize s ) [inline]
Sets the default size of text in the widget.
Parameters

\[ s \quad \text{new text size} \]

9.135.4.84  \texttt{update\_h\_scrollbar()}

\texttt{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.

9.135.4.85  \texttt{update\_line\_starts()}

\texttt{void Fl\_Text\_Display::update\_line\_starts (}
\begin{verbatim}
    int pos,
    int charsInserted,
    int charsDeleted,
    int linesInserted,
    int linesDeleted,
    int \* scrolled \) [protected]
\end{verbatim}

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 \texttt{pos} where the change began, and the numbers of characters and lines inserted and deleted.

Parameters

\begin{tabular}{|c|c|}
\hline
\texttt{pos} & index into buffer of recent changes \\
\hline
\texttt{charsInserted} & number of bytes(!) inserted \\
\hline
\texttt{charsDeleted} & number of bytes(!) deleted \\
\hline
\texttt{linesInserted} & number of lines \\
\hline
\texttt{linesDeleted} & number of lines \\
\hline
\texttt{scrolled} & set to 1 if the text display needs to be scrolled \\
\hline
\end{tabular}

9.135.4.86  \texttt{update\_v\_scrollbar()}

\texttt{void Fl\_Text\_Display::update\_v\_scrollbar ( ) [protected]}

Update vertical scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the vertical scrollbar.

9.135.4.87  \texttt{vline\_length()}

\texttt{int Fl\_Text\_Display::vline\_length (}
\begin{verbatim}
    int visLineNum \) const [protected]
\end{verbatim}

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

\begin{tabular}{|c|c|}
\hline
\texttt{visLineNum} & index of line in visible line array \\
\hline
\end{tabular}

Returns

number of bytes in this line
9.135.4.88 word_end()

```cpp
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**

- index of first character after the end of the word

9.135.4.89 word_start()

```cpp
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**

- beginning of the words

9.135.4.90 wrap_mode()

```cpp
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.

**Parameters**

<table>
<thead>
<tr>
<th>wrap</th>
<th>new wrap mode is 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), or WRAP_AT_BOUNDS (wrap text so that it fits into the widget width)</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>

**Todo** we need new wrap modes to wrap at the window edge and based on pixel width or average character width.

9.135.4.91 wrap_uses_character()

```cpp
int Fl_Text_Display::wrap_uses_character ( int lineEndPos ) const [protected]
```

Check if the line break is caused by a \n or by line wrapping.
Line breaks in continuous wrap mode usually happen at newlines 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 TextDEndOfLine or BufEndOfLine, 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

9.135.4.92 wrapped_column()

```cpp
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?

Todo Unicode?

9.135.4.93 wrapped_line_counter()

```cpp
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]
```

Generated by Doxygen
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 | |

9.135.4.94  wrapped_row()

```cpp
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?

9.135.4.95  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

9.135.4.96  xy_to_position()

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

9.135.4.97  xy_to_rowcol()

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

<table>
<thead>
<tr>
<th>X, Y</th>
<th>pixel coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>row, 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

9.136  Fl_Text_Editor Class Reference

This is the FLTK text editor widget.
#include <Fl_Text_Editor.H>

Inheritance diagram for 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()

- 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*="")

  The constructor creates a new text editor widget.
virtual int handle (int e)  
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  
Check 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.

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.

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 linenumbersize (Fl_Fontsize val)
  Set the font size used for line numbers (if enabled).

- **int** linenumberwidth () const
  Return the screen area width provided for line numbers.

- void linenumberwidth (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.
• void redisplay_range (int start, int end)
   Marks text from start to end as needing a redraw.
• virtual void resize (int X, int Y, int W, int H)
   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_width () const
   Gets the width/height of the scrollbars.
• void scrollbar_width (int W)
   Sets the width/height of the scrollbars.
• 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 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 *)**
  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.
- **virtual Fl_Group * as_group ()**
  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 *)**
  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**
  See void Fl_Group::resizable(Fl_Widget * box)
void resizable (Fl_Widget &o)
See void Fl_Group::resizable(Fl_Widget *box)

void resizable (Fl_Widget *o)
The resizable widget defines the resizing box for the group.

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 Fl_Window * as_window ()
Returns an Fl_Window pointer if this widget is an Fl_Window.

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, 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)
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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_IMAGE *img)
  Sets the image to use as part of the widget label.
• void do_callback ()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.

• **const Fl_Image *image () const**

• **void image (Fl_Image &img)**

  Sets the image to use as part of the widget label.

• **void image (Fl_Image *img)**

  Sets the image to use as part of the widget label.

• **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.

• **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.
• 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** (void ∗v)
  Sets the user data for this widget.
• unsigned int **visible** () const
  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** () 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, 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_m_s_move (int c, Fl_Text_Editor ∗e)
  Extends the current selection in the direction indicated by meta key 'c' in editor 'e'.
- static int kf_meta_move (int c, Fl_Text_Editor ∗e)
  Moves the current text cursor in the direction indicated by meta key 'c' 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_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)
  Undo 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 *cb, void *d)
  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

- int handle_key ()
  Handles a key press in the editor.

- void maybe_do_callback ()
  does or does not a callback according to changed() and when() settings.

Protected Member Functions inherited from Fl_Text_Display

- void absolute_top_line_number (int oldFirstChar)
  Line numbering stuff, currently unused.

- void calc_last_char ()
  Update last display character index.

- void calc_line_starts (int startLine, int endLine)
  Update the line start arrays.

- 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.

- virtual void draw ()
  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.

• int **get_absolute_top_line_number** () const
  Line numbering stuff, currently unused.

• 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
  Line numbering stuff, currently unused.

• 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** ()
  Line numbering stuff, probably unused.

• 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 \n or by line wrapping.

• **void wrapped_line_counter** (Fl_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIs←LineStart, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLast←LineMissingNewLine=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**

• **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** ()
  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_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 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 }`

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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31 }
flags possible values enumeration.

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)`
  Callbacks for drag or valueChanged on horizontal scrollbar.
- `static void scroll_timer_cb(void * cbArg)`
  Timer callback for scroll events.
- `static void v_scrollbar_cb(Fl_Scrollbar * w, Fl_Text_Display * d)`
  Callbacks 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_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 mVisibleLines
- 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_
- int shortcut_
9.136 Fl_Text_Editor Class Reference

- struct {
  int h
  int w
  int x
  int y
} text_area

- Fl_Color textcolor_
- Fl_Font textfont_
- Fl_Fontsize textsize_

9.136.1 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.

9.136.2 Member Function Documentation

9.136.2.1 add_key_binding()

```cpp
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_keyBindings.

9.136.2.2 handle()

```cpp
int Fl_Text_Editor::handle ( 
  int e ) [virtual]
```

Event handling.
Reimplemented from Fl_Text_Display.

9.136.2.3 insert_mode() [1/2]

```cpp
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.

9.136.2.4 insert_mode() [2/2]

```cpp
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.

9.136.2.5 kf_backspace()

```cpp
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.
9.136.2.6  \texttt{kf\_c\_s\_move()}

\begin{verbatim}
int Fl_Text_Editor::kf_c_s_move (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Extends the current selection in the direction indicated by control key \texttt{c} in editor \texttt{e}.

See also \texttt{kf\_ctrl\_move()}.

9.136.2.7  \texttt{kf\_copy()}

\begin{verbatim}
int Fl_Text_Editor::kf_copy (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Does a copy of selected text or the current character in the current buffer of editor \texttt{e}.

The key value \texttt{c} is currently unused.

9.136.2.8  \texttt{kf\_ctrl\_move()}

\begin{verbatim}
int Fl_Text_Editor::kf_ctrl_move (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Moves the current text cursor in the direction indicated by control key \texttt{c} in editor \texttt{e}.

Supported values for \texttt{c} are currently:
\begin{itemize}
    \item \texttt{FL\_Home} -- moves the cursor to the beginning of the document
    \item \texttt{FL\_End} -- moves the cursor to the end of the document
    \item \texttt{FL\_Left} -- moves the cursor left one word
    \item \texttt{FL\_Right} -- moves the cursor right one word
    \item \texttt{FL\_Up} -- scrolls up one line, without moving cursor
    \item \texttt{FL\_Down} -- scrolls down one line, without moving cursor
    \item \texttt{FL\_Page\_Up} -- moves the cursor to the beginning of the top line on the current page
    \item \texttt{FL\_Page\_Down} -- moves the cursor to the beginning of the last line on the current page
\end{itemize}

9.136.2.9  \texttt{kf\_cut()}

\begin{verbatim}
int Fl_Text_Editor::kf_cut (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Does a cut of selected text in the current buffer of editor \texttt{e}.

The key value \texttt{c} is currently unused.

9.136.2.10  \texttt{kf\_default()}

\begin{verbatim}
int Fl_Text_Editor::kf_default (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Inserts the text associated with key \texttt{c} in editor \texttt{e}.

Honors the current selection and insert/overstrike mode.

9.136.2.11  \texttt{kf\_delete()}

\begin{verbatim}
int Fl_Text_Editor::kf_delete (  
    int c,  
    Fl_Text_Editor * e) [static]
\end{verbatim}

Does a delete of selected text or the current character in the current buffer of editor \texttt{e}.

The key value \texttt{c} is currently unused.
9.136.2.12 `kf_down()`

```c
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.

9.136.2.13 `kf_end()`

```c
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.

9.136.2.14 `kf_enter()`

```c
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.

9.136.2.15 `kf_home()`

```c
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.

9.136.2.16 `kf_ignore()`

```c
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.

9.136.2.17 `kf_insert()`

```c
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.

9.136.2.18 `kf_left()`

```c
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.
9.136.2.19  

**kf_m_s_move()**

```c
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().**

9.136.2.20  

**kf_meta_move()**

```c
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

9.136.2.21  

**kf_move()**

```c
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

9.136.2.22  

**kf_page_down()**

```c
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.

9.136.2.23  

**kf_page_up()**

```c
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.

9.136.2.24  

**kf_paste()**

```c
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.
9.136.2.25 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.

9.136.2.26 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.

9.136.2.27 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()
```

9.136.2.28 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.

9.136.2.29 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.

9.136.2.30 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.

9.136.2.31 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.

---

Generated by Doxygen
Returns

if Tab inserts tab characters or moves the focus

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tab inserts tab characters (default)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Tab navigation is enabled.</td>
<td></td>
</tr>
</tbody>
</table>

See also

tab_nav(int), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

9.136.32  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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| in | val | If val is 0, Tab inserts a tab character (default).  
If val is 1, Tab navigates widget focus. |   |

See also

tab_nav(), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

9.136.3  Member Data Documentation

9.136.3.1  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

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.cxx

Generated by Doxygen
9.137 Fl_Text_Selection Class Reference

This is an internal class for Fl_Text_Buffer to manage text selections.  
#include <Fl_Text_Buffer.H>

Public Member Functions

- int end () const
  Return the byte offset to the character after the last selected character.

- int includes (int pos) const
  Return true if position pos with indentation dispIndex is in the Fl_Text_Selection.

- int position (int *start, int *end) const
  Return the positions of this selection.

- bool selected () const
  Returns true if any text is selected.

- void selected (bool b)
  Modify the 'selected' flag.

- void set (int start, int end)
  Set the selection range.

- int start () const
  Return 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

9.137.1 Detailed Description

This is an internal class for Fl_Text_Buffer to manage text selections.  
This class works correctly with UTF-8 strings assuming that the parameters for all calls are on character boundaries.

9.137.2 Member Function Documentation

9.137.2.1 end()

int Fl_Text_Selection::end ( ) const [inline]
Return the byte offset to the character after the last selected character.

Returns

  byte offset
9.137.2.2 position()

```cpp
int Fl_Text_Selection::position (  
    int * start,  
    int * end ) const
```

Return the positions of this selection.
Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>return byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>return byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

Returns

true if selected

9.137.2.3 selected() [1/2]

bool Fl_Text_Selection::selected ( ) const [inline]

Returns true if any text is selected.

Returns

a non-zero number if any text has been selected, or 0 if no text is selected.

9.137.2.4 selected() [2/2]

void Fl_Text_Selection::selected ( bool b ) [inline]

Modify the 'selected' flag.

Parameters

| b          | new flag |

9.137.2.5 set()

void Fl_Text_Selection::set ( int start, int end )

Set the selection range.

Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

9.137.2.6 start()

int Fl_Text_Selection::start ( ) const [inline]

Return the byte offset to the first selected character.

Returns

byte offset

9.137.2.7 update()

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

9.138 FlTile Class Reference

The FlTile class lets you resize its children by dragging the border between them.

Inheritance diagram for FlTile:

```
Fl_Widget
   ↓
Fl_Group
   ↓
Fl_Tile
```

Public Member Functions

- **FlTile** (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new FlTile widget using the given position, size, and label string.
- **int handle** (int event)
  Handles the specified event.
- **void position** (int oldx, int oldy, int newx, int newy)
  Drags the intersection at (oldx, oldy) to (newx, newy).
- **void resize** (int X, int Y, int W, int H)
  Resizes the FlTile widget and its children.

Public Member Functions inherited from FlGroup

- **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 FlGroup::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.
- **virtual FlGroup ∗as_group** ()
  Returns an FlGroup pointer if this widget is an FlGroup.
- **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 *)`
- **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 *)`
  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
  See void Fl_Group::resizable(Fl_Widget *box)
- **void** `resizable(Fl_Widget &o)`
  See void Fl_Group::resizable(Fl_Widget *box)
- **void** `resizable(Fl_Widget *o)`
  The resizable widget defines the resizing box for the group.
- **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 Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• 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, 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)
  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.

- **const Fl_Image *deimage () const**
  void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.

- **void deimage (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, long arg)**
  Calls the widget callback.

- **void do_callback (Fl_Widget *o, void *arg=0)**
  Calls the widget callback.

- **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.

- **const Fl_Image *image () const**
  void image (Fl_Image &img)
  Sets the image to use as part of the widget label.

- **void image (Fl_Image *img)**
  Sets the image to use as part of the widget label.

- **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...
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 (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 ∗cb, void ∗d)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

- void draw ()
  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 () 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 ()
  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const
9.138 Fl_Tile Class Reference

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.

9.138.1 Detailed Description

The Fl_Tile class lets you resize its children by dragging the border between them.

![Figure 9.43 Fl_Tile](image)

For the tiling to work correctly, the children of an Fl_Tile must cover the entire area of the widget, but 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.

Fl_Tile allows objects to be resized to zero dimensions. To prevent this you can use the resizable() 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 resizable(FL_Widget &w):** The "resizable" child widget (which should be invisible) limits where the borders can be dragged to. All dragging will be limited inside the resizable 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 resizable() 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 resizable() 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
resizable() widget so that they are lost for widgets covered (overlapped) by the resizable() widget.

**Note**

You can still resize widgets **inside** the resizable() to zero width and/or height, i.e. box **2b** above to zero width
and box **3a** to zero height.

See also

void Fl_Group::resizable(Fl_Widget &w)

Example for resizable with 20 pixel border distance:

```c
int dx = 20, dy = dx;
Fl_Tile tile{50,50,300,300};
// create resizable() box first
Fl_Box r(tile.x()+dx,tile.y()+dy,tile.w()-2*dx,tile.h()-2*dy);
tile.resizable(r);
// ... create widgets inside tile (see test/tile.cxx) ...
tile.end();
```

See also the complete example program in test/tile.cxx.

### 9.138.2 Constructor & Destructor Documentation

#### 9.138.2.1 Fl_Tile()

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

### 9.138.3 Member Function Documentation

#### 9.138.3.1 handle()

```c
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.

**Parameters**

| in | event | the kind of event received |

**Return values**

| 0 | if the event was not used or understood |
Return values

| t | if the event was used and can be deleted |

See also

- Fl_Event

Reimplemented from Fl_Group.

### 9.138.3.2 position()

```cpp
void Fl_Tile::position (int oldx, int oldy, int newx, int newy)
```

Drags the intersection at (oldx, oldy) to (newx, newy).

This redraws all the necessary children.

Pass zero as oldx or oldy to disable drag in that direction.

### 9.138.3.3 resize()

```cpp
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.

Enlarging works by just 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.

The documentation for this class was generated from the following files:

- Fl_Tile.H
- Fl_Tile.cxx

### 9.139 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_Tile
   |   Fl_Image
```

**Public Member Functions**

- virtual void `color_average (Fl_Color c, float i)`

  The `color_average()` method averages the colors in the image with the FLTK color value `c`. 
• **Fl_Image **\* copy ()

  *virtual Fl_Image \* copy (int W, int H)*

  The copy() method creates a copy of the specified image.

• virtual void desaturate ()

  *The desaturate() method converts an image to grayscale.*

• void draw (int X, int Y)

  *virtual void draw (int X, int Y, int W, int H, int cx, int cy)*

  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**

• **Fl_Image \* copy ()**

  *The copy() method creates a copy of the specified image.*

• int count () const

  *The count() method returns the number of data values associated with the image.*

• int d () const

  *Returns the current image depth.*

• const char \* const \* data () const

  *Returns a pointer to the current image data array.*

• void draw (int X, int Y)

  *Draws the image.*

• int fail ()

  *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 height in pixels.*

• 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)

  *The label() methods are an obsolete way to set the image attribute of a widget or menu item.*

• virtual void label (Fl_Widget \* w)

  *The label() methods are 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 uncache ()

  *If the image has been cached for display, delete the cache data.*

• int w () const

  *Returns the current image width in pixels.*

• 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_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 Public Attributes inherited from Fl_Image

- static const int ERR_FILE_ACCESS = -2
- static const int ERR_FORMAT = -3
- static const int ERR_NO_IMAGE = -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 array 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.
- void h(int H)
  Sets the current image height in pixels.
- void ld(int LD)
  Sets the current line data size in bytes.
- void w(int W)
  Sets the current image width in pixels.

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)

9.139.1 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.

9.139.2 Constructor & Destructor Documentation

9.139.2.1 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.

```cpp
#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 *)

9.139.3 Member Function Documentation

9.139.3.1 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 FLTK color value `c`. The `i` 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 before changes are applied, to avoid modifying the original image. Reimplemented from Fl_Image.

9.139.3.2 copy()

```cpp
Fl_Image * Fl_Tiled_Image::copy (int W, int H) [virtual]
```

The `copy()` method creates a copy of the specified image. If the width and height are provided, the image is resized to the specified size. The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it. Reimplemented from Fl_Image.

9.139.3.3 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 before changes are applied, to avoid modifying the original image. Reimplemented from Fl_Image.

9.139.3.4 draw()

```cpp
void Fl_Tiled_Image::draw (int X, int Y, int W,
```

```cpp
Generated by Doxygen
```
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

9.140 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)
  
  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.
• \texttt{\textasciitilde\texttt{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.

- 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\_Group * as\_group ()

   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.

- 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, 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)

   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.

• **const Fl_Image ∗deimage () const**
  
  **void deimage (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

• **void deimage (Fl_Image ∗img)**
  
  Sets the image to use as part of the widget label.

• **void do_callback ()**
  
  Calls the widget callback.

• **void do_callback (Fl_Widget ∗o, long arg)**
  
  Calls the widget callback.

• **void do_callback (Fl_Widget ∗o, void ∗arg=0)**
  
  Calls the widget callback.

• **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.
• const F Image *image () const
  
  Sets the image to use as part of the widget label.
• void image (F Image &img)
  
  Sets the image to use as part of the widget label.
• int inside (const F_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 (F Labeltype a, const char *b)
  
  Shortcut to set the label text and type in one call.
• F Color labelcolor () const
  
  Gets the label color.
• void labelcolor (F Color c)
  
  Sets the label color.
• F Font labelfont () const
  
  Gets the font to use.
• void labelfont (F Font f)
  
  Sets the font to use.
• F Fontsize labelsize () const
  
  Gets the font size in pixels.
• void labelsize (F Fontsize pix)
  
  Sets the font size in pixels.
• F Labeltype labeltype () const
  
  Gets the label type.
• void labeltype (F 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.
• unsigned int output () const
  
  Returns if a widget is used for output only.
• F_Group * parent () const
  
  Returns a pointer to the parent widget.
• void parent (F_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.
• F Color selection_color () const
  
  Gets the selection color.
• void selection_color (F 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.

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 (void * v)

Sets the user data for this widget.

unsigned int visible () const

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 () 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.

Protected Member Functions

- `void draw ()`
  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 ()`
  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_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 ∗cb, void ∗d)
  
  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 ,
  USERFLAG3 = 1<29 , USERFLAG2 = 1<30 , USERFLAG1 = 1<31 }

  flags possible values enumeration.

9.140.1 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 Fl::add_timeout() instead.

9.140.2 Constructor & Destructor Documentation

9.140.2.1 Fl_Timer()

Fl_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.
The type parameter can be any of the following symbolic constants:

- FL_NORMAL_TIMER - The timer just does the callback and displays the string "Timer" in the widget.
- FL_VALUE_TIMER - The timer does the callback and displays the current timer value in the widget.
- FL_HIDDEN_TIMER - The timer just does the callback and does not display anything.

9.140.3 Member Function Documentation

9.140.3.1 direction() [1/2]

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().
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().

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:

Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

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.

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.

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

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.
#include <Fl_Toggle_Button.H>

Inheritance diagram for Fl_Toggle_Button:

```
+------------------+
| Fl_Widget        |
|                  |
|                  |
| Fl_Button        |
+------------------+
|                  |
|                  |
| Fl_Toggle_Button |
```

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).*

- **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.*

- virtual int **handle** (int)
  
  *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 Fl_Group ∗ as_group ()
  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.
- 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, 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)
  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.

- `const FL_Image ∗ deimage () const`  
  void deimage (FL_Image &img)`  
  Sets the image to use as part of the widget label.

- `void deimage (const Fl_Image ∗ img)`  
  Sets the image to use as part of the widget label.

- `void do_callback ()`  
  Calls the widget callback.

- `void do_callback (FL_Widget ∗ o, long arg)`  
  Calls the widget callback.

- `void do_callback (FL_Widget ∗ o, void ∗ arg=0)`  
  Calls the widget callback.

- `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.

- `const FL_Image ∗ image () const`  
  void image (FL_Image &img)`  
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.

- **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 (void ∗v)
  
  Sets the user data for this widget.

- **unsigned int** visible () const
  
  Checks whether this widget has a visible focus.

- **unsigned int** visible_focu() 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 *cb, void *d)
  
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

• virtual void draw ()
  
  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 ()
  
  draws a focus rectangle around the widget

• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h) const

Generated by Doxygen
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`

### 9.141.1 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()`.

### 9.141.2 Constructor & Destructor Documentation

#### 9.141.2.1 Fl_Toggle_Button()

```cpp
Fl_Toggle_Button::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.

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**

<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_Toggle_Button.H
- Fl_Button.cxx

9.142 Fl_Tooltip Class Reference

The Fl_Tooltip class provides tooltip support for all FLTK widgets.

```
#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 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 *, int X, int Y, int W, int H, const char *)
  
  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 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.
9.142 Fl_Tooltip Class Reference

- 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

- void Fl_Widget::copy_tooltip(const char ∗)
- void Fl_Widget::tooltip(const char ∗)

9.142.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets. It contains only static methods.

9.142.2 Member Function Documentation

9.142.2.1 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.

9.142.2.2 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.
9.142.3 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.

9.142.4 delay() [1/2]

static float Fl_Tooltip::delay ( ) [inline], [static]

Gets the tooltip delay.
The default delay is 1.0 seconds.

9.142.5 delay() [2/2]

static void Fl_Tooltip::delay ( 
    float f ) [inline], [static]

Sets the tooltip delay.
The default delay is 1.0 seconds.

9.142.6 disable()

static void Fl_Tooltip::disable ( ) [inline], [static]

Same as enable(0), disables tooltips on all widgets.

9.142.7 enable()

static void Fl_Tooltip::enable ( 
    int b = 1 ) [inline], [static]

Enables tooltips on all widgets (or disables if b is false).

9.142.8 enabled()

static int Fl_Tooltip::enabled ( ) [inline], [static]

Returns non-zero if tooltips are enabled.

9.142.9 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).

9.142.2.10 font() [1/2]

static Fl_Font Fl_Tooltip::font ( ) [inline], [static]
Gets the typeface for the tooltip text.

9.142.2.11 font() [2/2]

static void Fl_Tooltip::font ( Fl_Font i ) [inline], [static]
Sets the typeface for the tooltip text.

9.142.2.12 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.

9.142.2.13 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.

9.142.2.14 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.

9.142.2.15 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.

9.142.2.16 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.

9.142.2.17 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.
9.142.18 size() [1/2]
static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]
  Gets the size of the tooltip text.

9.142.19 size() [2/2]
static void Fl_Tooltip::size ( Fl_Fontsize s ) [inline], [static]
  Sets the size of the tooltip text.

9.142.20 textcolor() [1/2]
static Fl_Color Fl_Tooltip::textcolor ( ) [inline], [static]
  Gets the color of the text in the tooltip.
  The default is black.

9.142.21 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.

9.142.22 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.

9.142.23 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

9.143 Fl_Tree Class Reference

Tree widget.
#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.
See if 'item' is currently displayed on-screen (visible within the widget).

- void draw()
  Standard FLTK draw() method, handles drawing the tree 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 &ret_items)
  Returns the currently selected items as an array of 'ret_items'.
- int handle (int e)
  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'.
- 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_labelbgcolor (Fl_Color val)
  
  Set the default label foreground color used for creating new items.

- Fl_Color item_labelfcolor (void) const
  
  Get the default label foreground color used for creating new items.

- Fl_Font item_labelfcolor () const
  
  Get the default font face used for creating new items.

- void item_labelfcolor (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**(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
  
  Return the specified '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)
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, 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.

- **virtual FL_Group * as_group ()**
  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 *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 &, 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**
  
  *See void Fl_Group::resizable(Fl_Widget *box)*

- **void resizable (Fl_Widget &o)**
  
  *See void Fl_Group::resizable(Fl_Widget *box)*

- **void resizable (Fl_Widget *o)**
  
  The resizable widget defines the resizing box for the group.

- **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 Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

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, 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)

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.

  - **const FL_Image** **deimage**( ) const
  
  Sets the image to use as part of the widget label.

  - **void deimage**(FL_Image &img)
    
    Sets the image to use as part of the widget label.

  - **void deimage**(FL_Image *img)
    
    Sets the image to use as part of the widget label.

- **void** **do_callback**()
  
  Calls the widget callback.

  - **void do_callback**(FL_Widget *o, long arg)
  
    Calls the widget callback.

  - **void do_callback**(FL_Widget *o, void *arg=0)
  
    Calls the widget callback.

- **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.

  - **const FL_Image** **image**( ) const
  
  Sets the image to use as part of the widget label.

  - **void image**(FL_Image &img)
  
    Sets the image to use as part of the widget label.

  - **void image**(FL_Image *img)
  
    Sets the image to use as part of the widget label.

- **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.
- 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.
- 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 **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 (void *v)**
  Sets the user data for this widget.

- **unsigned int visible () 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 item_clicked (Fl_Tree_Item *val)**
  Set the item that was last clicked.

### Protected Member Functions inherited from Fl_Group

- **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 ()
  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_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_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 *cb, void *d)
  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, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
flags possible values enumeration.

9.143.1 Detailed Description

Tree widget.

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);
```

Generated by Doxygen
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(), 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().

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:
```
for ( Fl_Tree_Item *i=first_selected_item(); i; i=next_selected_item(i) )
    printf("Item %s is selected\n", i->label());
```

To get an item's full menu pathname, use Fl_Tree::item_pathname(), e.g.
```
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:
```
// 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\n", item->label());
```

To recursively walk all the children of a particular item, define a function that uses recursion:
```
void my_print_all_children(Fl_Tree_Item *item, int indent=0) {
    printf("%s\n", 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 9.45 Fl_Tree elements](image)

The following shows the protected dimension variables 'tree inner' (tix..) and 'tree outer' (tox..):

![Figure 9.46 Fl_Tree inner/outer dimensions](image)

KEYBOARD BINDINGS

The following table lists keyboard bindings for navigating the tree:
Table 9.466 Keyboard bindings.

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>FL_TREE_SELECT_LEFT</th>
<th>FL_TREE_SELECT_RIGHT</th>
<th>FL_TREE_SELECT_UP</th>
<th>FL_TREE_SELECT_DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-A (Linux/Windows) Command-A (Mac)</td>
<td>Selects item.</td>
<td>Selects item.</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>
<td>N/A</td>
</tr>
<tr>
<td>Ctrl-Space</td>
<td>Toggle item.</td>
<td>Toggle item.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Space</td>
<td>Extends selection from last item.</td>
<td>Selects item.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Enter, Ctrl-Enter, Shift-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>N/A</td>
</tr>
<tr>
<td>Right / Left</td>
<td>Open/Close item.</td>
<td>Open/Close item.</td>
<td>Open/Close item.</td>
<td>N/A</td>
</tr>
<tr>
<td>Up / Down</td>
<td>Move focus up/down.</td>
<td>Move focus up/down.</td>
<td>Move focus 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>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>
<td>N/A</td>
</tr>
<tr>
<td>PageUp / PageDown</td>
<td>Page up/down.</td>
<td>Page up/down.</td>
<td>Page up/down.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

9.143.2 Member Function Documentation

9.143.2.1 add() [1/2]

```cpp
Fl_Tree_Item * Fl_Tree::add (const char * path,
               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 path</th>
<th>The path to the item, e.g. &quot;Flintstone/Fred&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in 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.

**Version**

1.3.3

9.143.2.2 add() [2/2]

```cpp
Fl_Tree_Item * Fl_Tree::add (Fl_Tree_Item * parent_item,
               const char * name )
```

Generated by Doxygen
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

1.3.0 release

### 9.143.2.3 calc_dimensions()

```cpp
def 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

### 9.143.2.4 calc_tree()

```cpp
def 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
```

_trees_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 recalc) occurs. (This use by an app should only rarely be needed)

### 9.143.2.5 callback_item() [1/2]

```cpp
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.
```
9.143.2.6 callback_item() [2/2]

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.

9.143.2.7 callback_reason() [1/2]

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:
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
    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

9.143.2.8 callback_reason() [2/2]

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.

9.143.2.9 clear()

void Fl_Tree::clear ( )
Clear the entire tree's children, including the root.
The tree will be left completely empty.

9.143.2.10 clear_children()

void Fl_Tree::clear_children (  
    Fl_Tree_Item * item  )
Clear all the children for 'item'.
Item may not be NULL.

9.143.2.11 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

| path | – the tree item's pathname (e.g. "Flintstones/Fred") |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
<th>– A flag that determines if the callback() is invoked or not:</th>
</tr>
</thead>
<tbody>
<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_CLOSED</td>
</tr>
</tbody>
</table>

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()

9.143.2.12 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

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>– the item to be closed. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td></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_CLOSED</td>
</tr>
</tbody>
</table>

Returns

• 1 – item was closed
• 0 – item was already closed, no change

See also

    open(), close(), is_open(), is_close(), callback_item(), callback_reason()

9.143.2.13 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.
9.143.2.14 closeicon() [2/2]

```cpp
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. |

9.143.2.15 connectorstyle()

```cpp
void Fl_Tree::connectorstyle ( 
    Fl_Tree_Connector val
)
```

Sets the line drawing style for inter-connecting items.
See Fl_Tree_Connector for possible values.

9.143.2.16 deselect() [1/2]

```cpp
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

9.143.2.17 deselect() [2/2]

```cpp
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></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_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

- 0 - item was already deselected, no change was made
- 1 - item's state was changed

9.143.2.18 deselect_all()

```cpp
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></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), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

Count of how many items were actually changed to the deselected state.

9.143.2.19 display()

```cpp
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. |

9.143.2.20 displayed()

```cpp
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

|   | 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.

### 9.143.2.21 draw()

```cpp
void Fl_Tree::draw ( void ) [virtual]
```

Standard FLTK draw() method, handles drawing the tree widget.
Reimplemented from Fl_Group.

### 9.143.2.22 extend_selection()

```cpp
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

|   | from | Starting item |
|   | to   | Ending item |
|   | val  | Select or deselect items (0=deselect, 1=select, 2=toggle) |
|   | visible | true=affect only open(), visible items, false=affect open or closed items (default) |

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3 ABI feature

### 9.143.2.23 extend_selection_dir()

```cpp
int Fl_Tree::extend_selection_dir ( Fl_Tree_Item * from,
                                   Fl_Tree_Item * co,
```
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

| in  | from   | Starting item |
| in  | to     | Ending item   |
| in  | dir    | Direction to extend selection (FL_Up or FL_Down) |
| in  | val    | 0=deselect, 1=select, 2=toggle |
| in  | visible | true=affect only open(), visible items, false=affect open or closed items (default) |

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3

9.143.2.24 find_clicked()

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

9.143.2.25 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.

```cpp
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>
</tr>
</thead>
<tbody>
<tr>
<td>– the tree item's pathname to be found (e.g. &quot;Flintstones/Fred&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

The item, or NULL if not found.

See also

- `item_pathname()`

### 9.143.2.26 first()

```cpp
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.

```cpp
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) )
  printf("Item: %s
", item->label());
```

**Returns**

First item in tree, or 0 if none (tree empty).

See also

- `first()`, `next()`, `last()`, `prev()`

### 9.143.2.27 first_selected_item()

```cpp
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.

```cpp
// 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
", i->label());
```

**Returns**

The first selected item, or 0 if none.

See also

- `first_selected_item()`, `last_selected_item()`, `next_selected_item()`

### 9.143.2.28 first_visible()

```cpp
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.
9.143.29  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

9.143.30  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:

```cpp
// 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

| out | ret_items | The returned array of selected items. |

Returns

The number of items in the returned array.

See also

`first_selected_item(), next_selected_item()`

Version

1.3.3 ABI feature

9.143.31  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.

9.143.32  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.

9.143.2.33 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

| in  | pos | The vertical position (in pixels) to scroll the tree to. |

See also

hposition(), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

9.143.2.34 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:

```cpp
// 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

| in  | item | The existing item to insert new child into. Must not be NULL. |
| in  | name | The label for the new item |
| in  | pos  | The position of the new item in the child list |

Returns

The new item added.
9.143 Fl_Tree Class Reference

See also

\[ \text{insert_above()} \]

9.143.2.35 insert_above()

\[
\text{Fl_Tree_Item} \ast \text{Fl_Tree::insert_above (}
    \text{Fl_Tree_Item} \ast \text{name,}
    \text{const char} \ast \text{above}
)\]

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 \ast item = tree->find_item("Aaa/222"); // get item Aaa/222
if (item) tree->insert_above(item, "New item"); // insert new item above it

Parameters

- in above – the item above which to insert the new item. Must not be NULL.
- in name – the name of the new item

Returns

- The new item added, or 0 if 'above' could not be found.

See also

\[ \text{insert()} \]

9.143.2.36 is_close() [1/2]

\[
\text{int Fl_Tree::is_close (}
    \text{const char} \ast \text{path}) \text{ const}
\]

See if item specified by 'path' is closed.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. \( \text{is_close("←Holidays/12\slash25\slash2010")} \).

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

9.143.2.37 is_close() [2/2]

\[
\text{int Fl_Tree::is_close (}
    \text{Fl_Tree_Item} \ast \text{item}) \text{ const}
\]

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

9.143.2.38 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.

9.143.2.39 is_open() [1/2]

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()

9.143.2.40 is_open() [2/2]

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
9.143.2.41 is_scrollback()

int Fl_Tree::is_scrollback ( 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:
for ( int i=0; i<tree->children(); i++ ) { // walk children
    Fl_Widget *w = tree->child(i);
    if ( tree->is_scrollback(w) ) continue; // skip scrollbars
    ..do work here..
}

Parameters

| in  | w | Widget to test |

Returns

1 if w is a scrollbar, 0 if not.

Todo should be const

9.143.2.42 is_selected() [1/2]

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

| in  | path | the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

• 1 : item selected
• 0 : item deselected
• -1 : item was not found

9.143.2.43 is_selected() [2/2]

int Fl_Tree::is_selected ( Fl_Tree_Item * item ) const
See if the specified 'item' is selected.

Parameters

| in  | item | the item to be tested. Must not be NULL. |

Returns

• 1 : item selected
• 0 : item deselected

9.143.2.44 is_vscroll_visible()

int Fl_Tree::is_vscroll_visible ( ) const
See if the vertical scrollbar is currently visible.

Returns
1 if scrollbar visible, 0 if not.

9.143.2.45 item_clicked() [1/2]

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.

9.143.2.46 item_clicked() [2/2]

void Fl_Tree::item_clicked ( ) [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.

9.143.2.47 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

9.143.2.48 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

9.143.2.49 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
9.143.2.50 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)

9.143.2.51 item_labelbgcolor() [2/2]

Fl_Color Fl_Tree::item_labelbgcolor ( 
    void ) const

Get the default label background color used for creating new items.
If the color is 0xffffffff, it is 'transparent'.

9.143.2.52 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)

9.143.2.53 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)

9.143.2.54 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::labelsizel(Fl_Fontsize)

9.143.2.55 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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td><strong>pathname</strong> The string to use to return the pathname</td>
</tr>
<tr>
<td>In</td>
<td><strong>pathnamelen</strong> The maximum length of the string (including NULL). Must not be zero.</td>
</tr>
<tr>
<td>In</td>
<td><strong>item</strong> The item whose pathname is to be returned.</td>
</tr>
</tbody>
</table>

Returns

- 0 : OK (pathname returns the item's pathname)
- -1 : item not found (pathname="")
- -2 : pathname not large enough (pathname="")
See also

find_item()

9.143.2.56 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

9.143.2.57 item_reselect_mode() [2/2]

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

9.143.2.58 last()

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.
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()

9.143.2.59 last_selected_item()

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.
// Walk tree in reverse, from bottom to top
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
9.143.2.60 last_visible()

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.

9.143.2.61 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

9.143.2.62 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

9.143.2.63 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:
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()
9.143.2.64  next_item()

**Fl_Tree_Item** * Fl_Tree::next_item ( 
    Fl_Tree_Item * item,
    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:

```c
// Walk down the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_item(i, FL_Down, true) )
    printf("Item: %s\n", i->label());

// Walk up the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_item(i, FL_Up, true) )
    printf("Item: %s\n", i->label());

// Walk down the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->first(); i; i=tree->next_item(i, FL_Down, false) )
    printf("Item: %s\n", i->label());

// Walk up the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->last(); i; i=tree->next_item(i, FL_Up, false) )
    printf("Item: %s\n", i->label());
```

**Parameters**

- **in item** The item to use to find the next item. If NULL, returns 0.
- **in dir** Can be FL_Up or FL_Down (default=FL_Down or 'next')
- **in visible** true=return only open(), visible items, false=return open or closed items (default)

**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

9.143.2.65  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

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>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)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default)</td>
</tr>
</tbody>
</table>

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

9.143.2.66 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.

// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_visible_item(i, FL_Down) )
  printf("Item: %s\n", i->label());

// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_visible_item(i, FL_Up) )
  printf("Item: %s\n", i->label());

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item above/below which we'll find the next visible item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>The direction to search. Can be FL_Up or FL_Down.</td>
</tr>
</tbody>
</table>

Returns

The item found, or 0 if there's no visible items above/below the specified item.

Version

1.3.3

9.143.2.67 open() [1/2]

int Fl_Tree::open (const char * path, int docalback = 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()

9.143.2.68 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
9.143 Fl_Tree Class Reference

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

9.143.2.69 open_toggle()

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

| in  | item       | -- the item whose open state is to be toggled. Must not be NULL. |
| in  | docallback | -- A flag that determines if the callback() is invoked or not: |
|     |            | • 0 - callback() is not invoked |
|     |            | • 1 - callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED |

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

9.143.2.70 openicon() [1/2]

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.

9.143.2.71 openicon() [2/2]

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

| in  | val       | -- The new image, or zero to use the default [+ ] icon. |

9.143.2.72 prev()

Fl_Tree_Item * Fl_Tree::prev ( 
    Fl_Tree_Item * item = 0 )

Return the previous item before 'item', or 0 if no more items.
This can be used to walk the tree in reverse, e.g.
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. |

Generated by Doxygen
Returns

Previous item in tree, or 0 if at first item.

See also

first(), next(), last(), prev()

9.143.2.73 recalc_tree()

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.

9.143.2.74 remove()

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.

9.143.2.75 resize()

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 docu-
mented 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.

9.143.2.76 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
9.143.2.77  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'.

9.143.2.78  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)

9.143.2.79  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

<table>
<thead>
<tr>
<th>in</th>
<th>size</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()

9.143.2.80  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

| in  | path  | the tree item's pathname (e.g. "Flintstones/Fred") |

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
<th>— A flag that determines if the callback() is invoked or not:</th>
</tr>
</thead>
<tbody>
<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

9.143.2.81 select() [2/2]

```c
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 

9.143.2.82 select_all()

```c
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

| in | item | The item that will be selected (along with all its children). If NULL, first() is used. |
| in | docallback | A flag that determines if the callback() is invoked or not: |
| | | • 0 - the callback() is not invoked |
| | | • 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED |

Returns

Count of how many items were actually changed to the selected state.

9.143.2.83 select_only()

```c
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

| in | selitem | The item to be selected. If NULL, first() is used. |
| in | docallback | A flag that determines if the callback() is invoked or not: |
| | | • 0 - the callback() is not invoked |
| | | • 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED |

Returns

The number of items whose selection states were changed, if any.

9.143.2.84 select_toggle()

```c
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. |
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
<th>– A flag that determines if the callback() is invoked or not:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 - the callback() is not invoked</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 - the callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED</td>
<td></td>
</tr>
</tbody>
</table>

9.143.2.85 selectbox() [1/2]

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()

9.143.2.86 selectbox() [2/2]

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()

9.143.2.87 selectmode() [1/2]

Fl_Tree_Select Fl_Tree::selectmode ( ) const

Gets the tree’s current selection mode.

See Fl_Tree_Select for possible values.

9.143.2.88 selectmode() [2/2]

void Fl_Tree::selectmode ( Fl_Tree_Select val )

Sets the tree’s selection mode.

See Fl_Tree_Select for possible values.

9.143.2.89 set_item_focus()

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. |

9.143.2.90 show_item() [1/2]

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

| in  | item | The item to be shown. If NULL, first() is used. |
See also
    show_item_top(), show_item_middle(), show_item_bottom()

9.143.2.91  show_item() [2/2]

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be shown. If NULL, first() is used.</td>
</tr>
<tr>
<td>in</td>
<td>yoff</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()

9.143.2.92  show_item_bottom()  

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be shown. If NULL, first() is used.</td>
</tr>
</tbody>
</table>

9.143.2.93  show_item_middle()  

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be shown. If NULL, first() is used.</td>
</tr>
</tbody>
</table>

9.143.2.94  show_item_top()  

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be shown. If NULL, first() is used.</td>
</tr>
</tbody>
</table>
9.143.2.95  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

9.143.2.96  showcollapse() [1/2]

```cpp
int Fl_Tree::showcollapse ( ) const
```
Returns 1 if the collapse icon is enabled, 0 if not.

See also

`showcollapse(int)`

9.143.2.97  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**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
</table>
|     | 1: shows collapse icons (default),
|     | 0: hides collapse icons. |

9.143.2.98  showroot()

```cpp
void Fl_Tree::showroot ( int val )
```
Set if the root item should be shown or not.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 – show the root item (default)</td>
</tr>
<tr>
<td></td>
<td>0 – hide the root item.</td>
</tr>
</tbody>
</table>

9.143.2.99  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.

9.143.2.100  usericon() [1/2]

```cpp
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.
9.143.2.101 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  

| in  | val  | – The new image to be used, or zero to disable user icons.

9.143.2.102 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)

9.143.2.103 vposition()  [2/2]

void Fl_Tree::vposition( )  

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  

| in | 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

9.144 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'.

Generated by Doxygen
• **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_label (const Fl_Tree_Prefs &prefs) const

  Was event on the label() of this item?

• 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 the index number for the specified 'item' in the current item's list of children.

• **Fl_Tree_Item** * find_child_item (const 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(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 yonly) const.

• const Fl_Tree_Item * find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0) const

  Find the item that the last event was over.

• **Fl_Tree_Item** * find_item (const Fl_Tree_Item *o)

  Non-const version of Fl_Tree_Item::find_item(char **names) const.

• const Fl_Tree_Item * find_item (char **arr) const

  Find 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** labelsize () const

  Get item's label font size.

• **void** labelsize (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 the item 'from' to sibling position of 'to'.

  - **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)`
  - Returns the item's 'visible' height.
- `int calc_item_height (const Fl_Tree_Prefs &prefs) const`
  - Internal: Horizontal connector line based on preference settings.
- `void draw_horizontal_connector (int x1, int x2, int y, 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.

9.144.1 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:

![Figure 9.47 Fl_Tree_Item's internal dimensions.](image)

9.144.2 Constructor & Destructor Documentation

9.144.2.1 `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`. 

Generated by Doxygen
**Deprecated** in 1.3.3 ABI – you must use `Fl_Tree_Item(Fl_Tree*)` for proper horizontal scrollbar behavior.

### 9.144.2.2 Fl_Tree_Item() [2/2]

```cpp
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

### 9.144.3 Member Function Documentation

#### 9.144.3.1 activate()

```cpp
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.

#### 9.144.3.2 add() [1/4]

```cpp
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

#### 9.144.3.3 add() [2/4]

```cpp
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

Generated by Doxygen


9.144.3.4  **add() [3/4]**

```cpp
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

---

9.144.3.5  **add() [4/4]**

```cpp
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

---

9.144.3.6  **calc_item_height()**

```cpp
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 !is_visible(), returns 0)
- 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

---

9.144.3.7  **child()**

```cpp
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'.
9.144.3.8 deactivate()

void Fl_Tree_Item::deactivate ( ) [inline]
Deactivate the item; the callback() won't be invoked when clicked.
Same as activate(0)

9.144.3.9 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)

9.144.3.10 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.

9.144.3.11 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".

9.144.3.12 draw()

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, out</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>

Generated by Doxygen
**9.144.3.13 draw_horizontal_connector()**

```cpp
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>x1</td>
<td>The left hand X position of the horizontal connector</td>
</tr>
<tr>
<td>x2</td>
<td>The right hand X position of the horizontal connector</td>
</tr>
<tr>
<td>y</td>
<td>The vertical position of the horizontal connector</td>
</tr>
<tr>
<td>prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>

**9.144.3.14 draw_item_content()**

```cpp
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:

```cpp
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:

```cpp
// 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:

```cpp
// 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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

9.144.3.15 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>in</th>
<th>x</th>
<th>The x position of the vertical connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>y1</td>
<td>The top of the vertical connector</td>
</tr>
<tr>
<td>in</td>
<td>y2</td>
<td>The bottom of the vertical connector</td>
</tr>
<tr>
<td>in</td>
<td>prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>

9.144.3.16 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

9.144.3.17 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

9.144.3.18 find_child() [1/2]

int Fl_Tree_Item::find_child (  
    const char * name )

Return the index of the immediate child of this item that has the label 'name'.

Generated by Doxygen
Returns

index of found item, or -1 if not found.

Version

1.3.0 release

9.144.3.19 find_child() [2/2]

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.

9.144.3.20 find_child_item() [1/2]

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

9.144.3.21 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

9.144.3.22 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>
9.144 Fl_Tree_Item Class Reference

Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

9.144.3.23 find_item()

const Fl_Tree_Item * Fl_Tree_Item::find_item ( const 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

9.144.3.24 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.

9.144.3.25 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 > item->children().

Returns

the new item inserted

See also

Fl_Tree::insert()

9.144.3.26 insert_above()

Fl_Tree_Item * Fl_Tree_Item::insert_above ( const Fl_Tree_Prefs & prefs, const char * new_label )

Insert a new item named 'new_label' above this item.

Returns

the new item inserted, or 0 if an error occurred.

9.144.3.27 label()

void Fl_Tree_Item::label ( const char * name )

Set the label to 'name'.
Makes and manages an internal copy of 'name'.

Generated by Doxygen
9.144.3.28  label_h()

int Fl_Tree_Item::label_h ( ) const [inline]
The item's label height.
Version
1.3.3

9.144.3.29  label_w()

int Fl_Tree_Item::label_w ( ) const [inline]
The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.
Version
1.3.3

9.144.3.30  label_x()

int Fl_Tree_Item::label_x ( ) const [inline]
The item's label x position relative to the window.
Version
1.3.3

9.144.3.31  label_y()

int Fl_Tree_Item::label_y ( ) const [inline]
The item's label y position relative to the window.
Version
1.3.3

9.144.3.32  labelbgcolor() [1/2]

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.)

9.144.3.33  labelbgcolor() [2/2]

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.

9.144.3.34  move() [1/2]

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')
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.)

9.144.3.35 move() [2/2]

int Fl_Tree_Item::move (  
    int to,  
    int from )
Move the item ‘from’ to sibling position of ‘to’.
Returns

• 0: Success
• -1: range error (e.g. if 'to' or 'from' out of range).
• (Other return values reserved for future use)

9.144.3.36 move_above()

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.

9.144.3.37 move_below()

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.

9.144.3.38 move_into()

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.
9.144.3.39  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.

9.144.3.40  next_displayed()

Fl_Tree_Item * Fl_Tree_Item::next_displayed ( Fl_Tree_Prefs & prefs )
Same as next_visible().
Deprecation in 1.3.3 for confusing name, use next_visible() instead.

9.144.3.41  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.

9.144.3.42  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

9.144.3.43  parent()

void Fl_Tree_Item::parent ( Fl_Tree_Item * val ) [inline]
Set the parent for this item.
Should only be used by Fl_Tree's internals.

9.144.3.44  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
9.144.3.45 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).

9.144.3.46 prev_displayed()

Fl_Tree_Item * Fl_Tree_Item::prev_displayed ( Fl_Tree_Prefs & prefs )

Same as prev_visible().
Deprecation in 1.3.3 for confusing name, use prev_visible()

9.144.3.47 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.

9.144.3.48 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.

9.144.3.49 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

9.144.3.50 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

|    | name | The label name of the immediate child to remove |

Returns

0 if removed, -1 if not found.

Version

1.3.3

9.144.3.51 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.

9.144.3.52 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.

9.144.3.53 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

|    | 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*)
9.144 Fl_Tree_Item Class Reference

Version

1.3.3 ABI feature

9.144.3.54 replace_child()

```cpp
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

9.144.3.55 select()

```cpp
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.

9.144.3.56 select_all()

```cpp
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".

9.144.3.57 show_self()

```cpp
void Fl_Tree_Item::show_self ( const char * indent = "" ) const
```

Print the tree as 'ascii art' to stdout.

Used mainly for debugging.

9.144.3.58 show_widgets()

```cpp
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()
9.144.3.59  swap_children() [1/2]

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

\begin{verbatim}
in  a,b  The item ptrs of the two items to swap. Both must be immediate children of the current item.
\end{verbatim}

Returns

- 0 : OK
- -1 : failed: item 'a' or 'b' is not our child.

9.144.3.60  swap_children() [2/2]

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.  
This method is FAST, and does not involve lookups.  
No range checking is done on either index value.

Parameters

\begin{verbatim}
in  ax,bx  the index of the items to swap
\end{verbatim}

9.144.3.61  tree() [1/2]

Fl_Tree * Fl_Tree_Item::tree ( ) [inline]
Return the tree for this item.
Version

1.3.4 (ABI feature)

9.144.3.62  tree() [2/2]

const Fl_Tree * Fl_Tree_Item::tree ( ) const [inline]
Return the tree for this item.
Version

1.3.3 (ABI feature)

9.144.3.63  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

| index | Our index# in the parent. Special case if index=-1: become an orphan; null out all parent/sibling associations. |

9.144.3.64  userdeicon() [1/2]

```c
Fl_Image * Fl_Tree_Item::userdeicon ( ) const [inline]
```

Return the deactivated version of the user icon, if any.
Returns 0 if none.

9.144.3.65  userdeicon() [2/2]

```c
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:
```c
// 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

9.144.3.66  usericon()

```c
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*)

9.144.3.67  visible_r()

```c
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

Generated by Doxygen
9.145  Fl_Tree_Item_Array Class Reference

Manages an array of Fl_Tree_Item pointers.
#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.

9.145.1  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).
### 9.145.2 Constructor & Destructor Documentation

#### 9.145.2.1 Fl_Tree_Item_Array()

```cpp
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.

### 9.145.3 Member Function Documentation

#### 9.145.3.1 add()

```cpp
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.

#### 9.145.3.2 clear()

```cpp
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.

#### 9.145.3.3 deparent()

```cpp
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)

#### 9.145.3.4 insert()

```cpp
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).

#### 9.145.3.5 manage_item_destroy()

```cpp
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.
9.145.3.6 move()

```cpp
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)

9.145.3.7 remove() [1/2]

```cpp
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.

9.145.3.8 remove() [2/2]

```cpp
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>

9.145.3.9 reparent()

```cpp
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)

9.145.3.10 replace()

```cpp
void Fl_Tree_Item_Array::replace ( int index, Fl_Tree_Item * newitem )
```

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
9.146 Fl_TreeNode Prefs Class Reference

Tree widget’s preferences.

#include <Fl_TreeNodePrefs.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_TreeNodePrefs ()**
  
  **Fl_TreeNodePrefs 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_labelfgcolor () const**
  
  Get the default label foreground color.

- **void item_labelfgcolor (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)**

Generated by Doxygen
Set the label's size in pixels to \texttt{val}.

- \texttt{FI_Tree_Item_Reselect_Mode item_reselect_mode() const}
  
  Returns the current item re/selection mode.

- \texttt{void item_reselect_mode(FI_Tree_Item_Reselect_Mode mode)}
  
  Sets the item re/selection mode.

- \texttt{FI_Color labelbgcolor() const}
  
  \texttt{Obsoleted: Get the default label background color. Please use item_labelbgcolor() instead.}

- \texttt{void labelbgcolor(FI_Color val)}
  
  \texttt{Obsoleted: Set the default label background color. Please use item_labelbgcolor(FI_Color) instead.}

- \texttt{FI_Color labelfgcolor() const}
  
  \texttt{Obsoleted: Get the default label foreground color. Please use item_labelfgcolor() instead.}

- \texttt{void labelfgcolor(FI_Color val)}
  
  \texttt{Obsoleted: Set the default label foreground color. Please use item_labelfgcolor(FI_Color) instead.}

- \texttt{FI_Font labelfont() const}
  
  \texttt{Obsoleted: Return the label's font. Please use item_labelfont() instead.}

- \texttt{void labelfont(FI_Font val)}
  
  \texttt{Obsoleted: Set the label's font to \texttt{val}. Please use item_labelfont(FI_Font) instead.}

- \texttt{int labelmarginleft() const}
  
  Get the label's left margin value in pixels.

- \texttt{void labelmarginleft(int val)}
  
  Set the label's left margin value in pixels.

- \texttt{FI_Fontsize labelsize() const}
  
  \texttt{Obsoleted: Return the label's size in pixels. Please use item_labelsize() instead.}

- \texttt{void labelsize(FI_Fontsize val)}
  
  \texttt{Obsoleted: Set the label's size in pixels to \texttt{val}. Please use item_labelsize(FI_Fontsize) instead.}

- \texttt{int linespacing() const}
  
  Get the line spacing value in pixels.

- \texttt{void linespacing(int val)}
  
  Set the line spacing value in pixels.

- \texttt{int marginbottom() const}
  
  Get the bottom margin's value in pixels.

- \texttt{void marginbottom(int val)}
  
  Set the bottom margin's value in pixels. \texttt{This is the extra distance the vertical scroller lets you travel.}

- \texttt{int margintop() const}
  
  Get the top margin's value in pixels.

- \texttt{void margintop(int val)}
  
  Set the top margin's value in pixels.

- \texttt{int openchild_marginbottom() const}
  
  Get the margin below an open child in pixels.

- \texttt{void openchild_marginbottom(int val)}
  
  Set the margin below an open child in pixels.

- \texttt{FI_Image * opendeicon() const}
  
  Return the deactivated version of the open icon, if any.

- \texttt{FI_Image * openicon() const}
  
  Get the current default 'open' icon.

- \texttt{void openicon(FI_Image *val)}
  
  Sets the default icon to be used as the 'open' icon when items are add\texttt{ed} to the tree.
• **Fl_Boxtype** selectbox () const
  
  Get the default selection box's box drawing style as an **Fl_Boxtype**.

• void selectbox (Fl_Boxtype val)
  
  Set the default selection box's box drawing style to **val**.

• **Fl_Tree_Select** selectmode () const
  
  Get the selection mode used for the tree.

• void selectmode (Fl_Tree_Select 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_Tree_Sort** sortorder () const
  
  Get the default sort order value.

• void sortorder (Fl_Tree_Sort val)
  
  Set the default sort order value.

• **Fl_Image** * userdeicon () const
  
  Return the deactivated version of the user icon, if any.

• **Fl_Image** * usericon () const
  
  Gets the default 'user icon' (default is 0)

• void usericon (Fl_Image *val)
  
  Sets the default 'user icon' Returns the **Fl_Image** 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_Tree_Prefs ()
  
  **Fl_Tree_Prefs** destructor.

### 9.146.1 Detailed Description

Tree widget's preferences.

**Fl_Tree**'s Preferences class.

This class manages the **Fl_Tree**'s defaults. You should probably be using the methods in **Fl_Tree** instead of trying to accessing tree's preferences settings directly.

### 9.146.2 Member Function Documentation

#### 9.146.2.1 closedeicon()

**Fl_Image** * Fl_Tree_Prefs::closedeicon ( ) const [inline]

Return the deactivated version of the close icon, if any.

Returns 0 if none.
9.146.2.2 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. |

9.146.2.3 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.

9.146.2.4 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.

9.146.2.5 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().

9.146.2.6 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.

9.146.2.7 opendeicon()

Fl_Image * Fl_Tree_Prefs::opendeicon () const [inline]

Return the deactivated version of the open icon, if any. Returns 0 if none.

9.146.2.8 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.

9.146.2.9 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. |

### 9.146.2.10 selectmode()

```cpp
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.

### 9.146.2.11 showcollapse()

```cpp
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. |

### 9.146.2.12 showroot()

```cpp
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. |

### 9.146.2.13 sortorder()

```cpp
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.

### 9.146.2.14 userdeicon()

```cpp
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:

- `Fl_Tree_Prefs.H`
- `Fl_Tree_Prefs.cxx`
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.

```c
#include <Fl_Valuator.H>
```

Inheritance diagram for Fl_Valuator:

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 $10^{-digits}$.

- **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**()
  
• **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 Fl_Group * as_group**()
  
  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.
  
• **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, 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)**
  
  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.

- `const Fl_Image ∗deimage () const`
  Gets the image that is used as part of the widget label.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label.

- `void deimage (Fl_Image ∗img)`
  Sets the image to use as part of the widget label.

- `void do_callback ()`
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, long arg)`
  Calls the widget callback.

- `void do_callback (Fl_Widget ∗o, void ∗arg=0)`
  Calls the widget callback.

- `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.

- **const Fl_Image * image () const**
  
  Sets the image to use as part of the widget label.

- **void image (Fl_Image &img)**
  
  Sets the image to use as part of the widget label.

- **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.

- **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.

- **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 (void ∗ v)
  Sets the user data for this widget.

- **unsigned int** visible () const
  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 () 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 ()
  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_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 ∗cb, void ∗d)
  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 ,
  USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }
  flags possible values enumeration.
9.147 Fl_Valuator Class Reference

9.147.1 Detailed Description

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.

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.

9.147.2 Constructor & Destructor Documentation

9.147.2.1 Fl_Valuator()

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.

9.147.3 Member Function Documentation

9.147.3.1 format()

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.

9.147.3.2 increment()

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.

9.147.3.3 maximum() [1/2]
double Fl_Valuator::maximum ( ) const [inline]

Gets the maximum value for the valuator.

9.147.3.4 maximum() [2/2]

void Fl_Valuator::maximum ( double a ) [inline]

Sets the maximum value for the valuator.

9.147.3.5 minimum() [1/2]
double Fl_Valuator::minimum ( ) const [inline]

Gets the minimum value for the valuator.

9.147.3.6 minimum() [2/2]

void Fl_Valuator::minimum ( double a ) [inline]

Sets the minimum value for the valuator.

9.147.3.7 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.

9.147.3.8 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.

### 9.147.3.9 round()

double Fl_Valuator::round ( double v )

Round the passed value to the nearest step increment.

Does nothing if step is zero.

### 9.147.3.10 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.

### 9.147.3.11 value()[1/2]

double Fl_Valuator::value ( ) const [inline]

Gets the floating point(double) value.

See int `value(double)`

### 9.147.3.12 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).

### 9.147.3.13 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

---

### 9.148 Fl_Value_Input Class Reference

The `Fl_Value_Input` widget displays a numeric value.

```cpp
#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)**
  Handles the specified event.
- **void resize (int, int, int, int)**
  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内部 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 ()

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_Group ∗ as_group ()

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.
• 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, 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)
  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)
• void deactivate ()
  Deactivates the widget.

• Fl_Image ∗ deimage ()
  Gets the image that is used as part of the widget label.

• const Fl_Image ∗ deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.

• void deimage (Fl_Image ∗img)
  Sets the image to use as part of the widget label.

• void do_callback ()
  Calls the widget callback.

• void do_callback (Fl_Widget ∗o, long arg)
  Calls the widget callback.

• void do_callback (Fl_Widget ∗o, void ∗arg=0)
  Calls the widget callback.

• 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.

• const Fl_Image ∗ image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.

• void image (Fl_Image ∗img)
  Sets the image to use as part of the widget label.

• 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.

• 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.

• 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 (void *v)
  Sets the user data for this widget.

- unsigned int visible () const
  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 () 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 ()
  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()`
  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_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 *cb, void *d)`
  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_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
    USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

9.148.1 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.

![Figure 9.49 Fl_Value_Input](image)

9.148.2 Constructor & Destructor Documentation

9.148.2.1 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.

9.148.3 Member Function Documentation

9.148.3.1 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.

9.148.3.2 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.

### 9.148.3.3 draw()

```cpp
doctor{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:
```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```Implements Fl_Widget.

### 9.148.3.4 handle()

```cpp
doctor{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.
```

**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.

### 9.148.3.5 resize()

```cpp
doctor{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.

9.148.3.6 shortcut() [1/2]

int Fl_Value_Input::shortcut ( ) const [inline]

Returns the current shortcut key for the Input.

See also

Fl_Value_Input::shortcut(int)

9.148.3.7 shortcut() [2/2]

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).

9.148.3.8 soft()

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.

9.148.3.9 textcolor()

Fl_Color Fl_Value_Input::textcolor ( ) const [inline]

Gets the color of the text in the value box.

9.148.3.10 textfont() [1/2]

Fl_Font Fl_Value_Input::textfont ( ) const [inline]

Gets the typeface of the text in the value box.

9.148.3.11 textfont() [2/2]

void Fl_Value_Input::textfont ( Fl_Font s ) [inline]

Sets the typeface of the text in the value box.
9.148.3.12  `textsize()` [1/2]

*Fl_Fontsize* `Fl_Value_Input::textsize ( ) const [inline]`

Gets the size of the text in the value box.

9.148.3.13  `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*

### 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_Value_Output
  |   
  V   
Fl_Valuator
  |   
  V   
Fl_Widget
```

#### 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)`**
  
  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)`**
  
  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 \(1.0 / 10^{\text{digits}}\).

- 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 Fl_Group * as_group ()**
  
  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.

- **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, 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)**
  
  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.
• const FL_IMAGE * **deimage** () const
• void **deimage** (FL_IMAGE &img)
  Sets the image to use as part of the widget label.
• void **deimage** (FL_IMAGE *img)
  Sets the image to use as part of the widget label.
• void **do_callback** ()
  Calls the widget callback.
• void **do_callback** (FL_Widget *o, long arg)
  Calls the widget callback.
• void **do_callback** (FL_Widget *o, void *arg=0)
  Calls the widget callback.
• void **draw_label** (int, int, int, int, FLAlign) 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.
• const FL_IMAGE * **image** () const
• void **image** (FL_IMAGE &img)
  Sets the image to use as part of the widget label.
• void **image** (FL_IMAGE *img)
  Sets the image to use as part of the widget label.
• 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)
  
  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.

• 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.

• 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 * topWindow () const
  Returns a pointer to the top-level window for the widget.
• Fl_Window * topWindow_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 * userData () const
  Gets the user data for this widget.
• void userData (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ()
  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** ()
  
  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_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)
Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *cb, void *d)
  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, TOOLBOX_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP:relative = 1<<16, COPYED_TOOLBOX = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19, USERFLAG1 = 1<<20, USERFLAG2 = 1<<21, USERFLAG3 = 1<<22
}

flags possible values enumeration.

9.149.1 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.

9.149.2 Constructor & Destructor Documentation

9.149.2.1 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.
9.149.3 Member Function Documentation

9.149.3.1 draw()

```cpp
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:

```cpp
Fl_Widget *s = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

9.149.3.2 handle()

```cpp
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.

**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`.

9.149.3.3 soft() [1/2]

```cpp
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.

9.149.3.4 soft() [2/2]

```cpp
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.

9.149.3.5 textcolor() [1/2]

```cpp
Fl_Color Fl_Value_Output::textcolor ( ) const [inline]
```

Sets the color of the text in the value box.
9.149.3.6  `textcolor()` [2/2]

```c
void Fl_Value_Output::textcolor (Fl_Color s) [inline]
```

Gets the color of the text in the value box.

9.149.3.7  `textfont()` [1/2]

```c
Fl_Font Fl_Value_Output::textfont () const [inline]
```

Gets the typeface of the text in the value box.

9.149.3.8  `textfont()` [2/2]

```c
void Fl_Value_Output::textfont (Fl_Font s) [inline]
```

Sets the typeface of the text in the value box.

9.149.3.9  `textsize()`

```c
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

9.150  **Fl_Value_Slider Class Reference**

The `Fl_Value_Slider` widget is a `Fl_Slider` widget with a box displaying the current value.

```c
#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)`
  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.

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)
  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 Fl_Group * as_group ()
  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.
• 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, 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)`
  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.

- **const FL_Image * deimage () const**

- **void deimage (FL_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void deimage (FL_Image *img)**
  
  Sets the image to use as part of the widget label.

- **void do_callback ()**
  
  Calls the widget callback.

- **void do_callback (FL_Widget *o, long arg)**
  
  Calls the widget callback.

- **void do_callback (FL_Widget *o, void *arg=0)**
  
  Calls the widget callback.

- **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.

- **const FL_Image * image () const**

- **void image (FL_Image &img)**
  
  Sets the image to use as part of the widget label.

- **void image (FL_Image *img)**
  
  Sets the image to use as part of the widget label.

- **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 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.

- 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.

- 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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ()
  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 ()**
  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_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 ∗ cb, void ∗ d)**
  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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
} flags possible values enumeration.

9.150.1 Detailed Description

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

Figure 9.51 Fl_Value_Slider

9.150.2 Constructor & Destructor Documentation

9.150.2.1 Fl_Value_Slider()

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. 
The default boxtype is FL_DOWN_BOX.

9.150.3 Member Function Documentation

9.150.3.1 draw()

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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Slider.

9.150.3.2 handle()

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.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

**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_Slider.

9.150.3.3 textcolor() [1/2]

`Fl_Color Fl_Value_Slider::textcolor ( ) const [inline]`

Gets the color of the text in the value box.

9.150.3.4 textcolor() [2/2]

`void Fl_Value_Slider::textcolor ( Fl_Color s ) [inline]`

Sets the color of the text in the value box.

9.150.3.5 textfont() [1/2]

`Fl_Font Fl_Value_Slider::textfont ( ) const [inline]`

Gets the typeface of the text in the value box.

9.150.3.6 textfont() [2/2]

`void Fl_Value_Slider::textfont ( Fl_Font s ) [inline]`

Sets the typeface of the text in the value box.

9.150.3.7 textsize() [1/2]

`Fl_Fontsize Fl_Value_Slider::textsize ( ) const [inline]`

Gets the size of the text in the value box.
### Fl_Widget Class Reference

**Fl_Widget** is the base class for all widgets in FLTK.

```c
#include <Fl_Widget.H>
```

Inheritance diagram for Fl_Widget:

```
Fl_Widget
   |   |
   V   V
Fl_Box
|     |
|     |
Fl_Button
|     |
|     |
Fl_Chart
|     |
|     |
Fl_Clock_Output
|     |
|     |
Fl_FormsBitmap
|     |
|     |
Fl_FormsPixmap
|     |
|     |
Fl_FormsText
|     |
|     |
Fl_Free
|     |
|     |
Fl_Group
|     |
|     |
Fl_Input_
|     |
|     |
Fl_Menu_
|     |
|     |
Fl_Positioner
|     |
|     |
Fl_Progress
|     |
|     |
Fl_Timer
|     |
|     |
Fl_Valuator
```

#### 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

The documentation for this class was generated from the following files:

- `Fl_Value_Slider.H`
- `Fl_Value_Slider.cxx`
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_Group * as_group()**
  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.

- **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, 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)**
  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.
- const Fl_Image *deimage () const
- void deimage (Fl_Image &img)
  
  Sets the image to use as part of the widget label.
- void deimage (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
- void do_callback ()
  
  Calls the widget callback.
- void do_callback (Fl_Widget *o, long arg)
  
  Calls the widget callback.
- void do_callback (Fl_Widget *o, void *arg=0)
  
  Calls the widget callback.
- 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.
- const Fl_Image *image () const
- void image (Fl_Image &img)
  
  Sets the image to use as part of the widget label.
- void image (Fl_Image *img)
  
  Sets the image to use as part of the widget label.
- 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.
- `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.
• 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 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 useAccents_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 (void *v)
  Sets the user data for this widget.
• unsigned int visible () const
  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 () 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 ∗cb, void ∗d)
  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

• 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_ACCENTS_MENU
  = 1<<19 ,
  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 ()
  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_label () const
  Draws the widget's label at the given position and size.

• 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.
9.151 Fl_Widget Class Reference

• void x (int v)
  Internal use only.
• void y (int v)
  Internal use only.

Friends

• class Fl_Group

9.151.1 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.

9.151.2 Member Enumeration Documentation

9.151.2.1 anonymous enum

anonymous enum [protected]
flags possible values enumeration.
See activate(), output(), visible(), changed(), set_visible_focus()

Enumerator

<table>
<thead>
<tr>
<th>Flag</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 (Fl_Window)</td>
</tr>
<tr>
<td>FORCE_POSITION</td>
<td>don't let the window manager position the window (Fl_Window)</td>
</tr>
<tr>
<td>NON_MODAL</td>
<td>this is a hovering toolbar window (Fl_Window)</td>
</tr>
<tr>
<td>SHORTCUT_LABEL</td>
<td>the label contains a shortcut we need to draw</td>
</tr>
<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 winows (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 Mac OS platform, pressing and holding a key on the keyboard opens an accented-character menu window (Fl_Input_, Fl_Text_Editor)</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>
</tbody>
</table>

Generated by Doxygen
### 9.151.3 Constructor & Destructor Documentation

#### 9.151.3.1 Fl_Widget()

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</th>
<th>x,y</th>
<th>the position of the widget relative to the enclosing window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w,h</td>
<td>size of the widget in pixels</td>
</tr>
<tr>
<td>in</td>
<td>label</td>
<td>optional text for the widget label</td>
</tr>
</tbody>
</table>

#### 9.151.3.2 ∼Fl_Widget()

Fl_Widget::~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.

Since FLTK 1.3, the widget's destructor removes the widget from its parent group, if it is member of a group.

Destruction removes the widget from any parent group! And groups when destroyed destroy all their children. This is convenient and fast.

### 9.151.4 Member Function Documentation

#### 9.151.4.1 activate()

void Fl_Widget::activate ( )

Activates the widget. Changing this value will send FL_ACTIVATE to the widget if active_r() is true.

See also

activate(), active_r(), deactivate()

#### 9.151.4.2 active()

unsigned int Fl_Widget::active ( ) const [inline]

Returns whether the widget is active.
Return values

0 if the widget is inactive

See also

active_r(), activate(), deactivate()

9.151.4.3 active_r()

int Fl_Widget::active_r ( ) const

Returns whether the widget and all of its parents are active.

Return values

0 if this or any of the parent widgets are inactive

See also

active(), activate(), deactivate()

9.151.4.4 align()[1/2]

Fl_Align Fl_Widget::align ( ) const [inline]

Gets the label alignment.

Returns

label alignment

See also

label(), align(Fl_Align), Fl_Align

9.151.4.5 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

9.151.4.6 argument()[1/2]

long Fl_Widget::argument ( ) const [inline]

Gets the current user data (long) argument that is passed to the callback function.

Todo  The user data value must be implemented using intptr_t or similar to avoid 64-bit machine incompatibilities
9.151.4.7 argument() [2/2]

void Fl_Widget::argument ( 
    long v ) [inline]
Sets the current user data (long) argument that is passed to the callback function.

Todo  The user data value must be implemented using intptr_t or similar to avoid 64-bit machine incompatibilities.

9.151.4.8 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.

9.151.4.9 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.

9.151.4.10 as_window()

virtual Fl_Window * Fl_Widget::as_window ( ) [inline], [virtual]
Returns an Fl_Widget 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.

9.151.4.11 box() [1/2]

Fl_Boxtype Fl_Widget::box ( ) const [inline]

Gets the box type of the widget.

Returns

the current box type

See also

box(Fl_Boxtype), Fl_Boxtype

9.151.4.12 box() [2/2]

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

| in  | new_box | the new box type |

See also

box(), Fl_Boxtype

9.151.4.13 callback() [1/5]

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
**9.151.4.14 callback() [2/5]**

```cpp
void Fl_Widget::callback (
    Fl_Callback ∗ cb ) [inline]
```

Sets the current callback function for the widget. Each widget has a single callback.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>new callback</th>
</tr>
</thead>
</table>

**9.151.4.15 callback() [3/5]**

```cpp
void Fl_Widget::callback (
    Fl_Callback ∗ cb,
    void ∗ p ) [inline]
```

Sets the current callback function for the widget. Each widget has a single callback.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>new callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>user data</td>
</tr>
</tbody>
</table>

**9.151.4.16 callback() [4/5]**

```cpp
void Fl_Widget::callback (
    Fl_Callback0 ∗ cb ) [inline]
```

Sets the current callback function for the widget. Each widget has a single callback.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>new callback</th>
</tr>
</thead>
</table>

**9.151.4.17 callback() [5/5]**

```cpp
void Fl_Widget::callback (
    Fl_Callback1 ∗ cb,
    long p = 0 ) [inline]
```

Sets the current callback function for the widget. Each widget has a single callback.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>new callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>user data</td>
</tr>
</tbody>
</table>

**9.151.4.18 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.

Return values

| 0 | if the value did not change |

See also

- `set_changed()`, `clear_changed()`

### 9.151.4.19 clear_active()

```c
void Fl_Widget::clear_active();
```

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()`

### 9.151.4.20 clear_changed()

```c
void Fl_Widget::clear_changed();
```

Marks the value of the widget as unchanged.

See also

- `changed()`, `set_changed()`

### 9.151.4.21 clear_damage()

```c
void Fl_Widget::clear_damage(uchar c = 0);
```

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!

Note

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()`

### 9.151.4.22 clear_output()

```c
void Fl_Widget::clear_output();
```

Sets a widget to accept input.
9.151.4.23 clear_visible()

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.

9.151.4.24 clear_visible_focus()

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)

9.151.4.25 color() [1/3]

Fl_Color Fl_Widget::color ( ) const [inline]
Gets the background color of the widget.

Returns
current background color

See also
color(Fl_Color), color(Fl_Color, Fl_Color)

9.151.4.26 color() [2/3]

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)

9.151.4.27 color() [3/3]

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)

---

### 9.151.4.28 color2() [1/2]

```cpp
Fl_Color Fl_Widget::color2 () const [inline]
```

For back compatibility only.

**Deprecated** Use selection_color() instead.

### 9.151.4.29 color2() [2/2]

```cpp
void Fl_Widget::color2 (unsigned a ) [inline]
```

For back compatibility only.

**Deprecated** Use selection_color(unsigned) instead.

---

### 9.151.4.30 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.

---

### 9.151.4.31 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**

| in       | **new_label** | the new label text |

See also

- `label()`

---

Generated by Doxygen
9.151.4.32 copy_tooltip()

```c
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

| in  | text | New tooltip text (an internal copy is made and managed) |

See also

- tooltip(const char*), tooltip()

9.151.4.33 damage() [1/3]

```c
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)

9.151.4.34 damage() [2/3]

```c
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

| in  | c | bitmask of flags to set |

See also

- damage(), clear_damage(uchar)

9.151.4.35 damage() [3/3]

```c
void Fl_Widget::damage (uchar c, int x, int y, int w, int h)
```

Sets the damage bits for an area inside the widget.
Setting damage bits will schedule the widget for the next redraw.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>c</th>
<th>bitmask of flags to set</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>size of damaged area</td>
</tr>
</tbody>
</table>

See also

damage(), clear_damage(uchar)

9.151.4.36 deactivate()

```cpp
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 active() is true on it and all of its parents.

Changing this value will send FL_DEACTIVATE to the widget if active_r() is true.

Currently you cannot deactivate Fl_Window widgets.

See also

activate(), active(), active_r()

9.151.4.37 default_callback()

```cpp
void Fl_Widget::default_callback ( Fl_Widget * cb, void * d ) [static]
```

The 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 Fl::readqueue().

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 * cb, void * data) or one of its variants.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>the widget given to the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>d</td>
<td>user data associated with that callback</td>
</tr>
</tbody>
</table>

See also

callback(), do_callback(), Fl::readqueue()

9.151.4.38 deimage() [1/3]

```cpp
Fl_Image * Fl_Widget::deimage () [inline]
```

Gets the image that is used as part of the widget label.

This image is used when drawing the widget in the inactive state.

Returns

the current image for the deactivated widget

9.151.4.39 deimage() [2/3]

```cpp
void Fl_Widget::deimage ( Fl_Image & img ) [inline]
```
Sets the image to use as part of the widget label. This image is used when drawing the widget in the inactive state.

Parameters

| in  | img | the new image for the deactivated widget |

9.151.4.40 deimage() [3/3]

```c
void Fl_Widget::deimage ( Fl_Image * img ) [inline]
```

Sets the image to use as part of the widget label.

This image is used when drawing the widget in the inactive state.

Parameters

| in  | img | the new image for the deactivated widget |

9.151.4.41 do_callback() [1/3]

```c
void Fl_Widget::do_callback ( ) [inline]
```

Calls the widget callback.
Causes a widget to invoke its callback function with default arguments.

See also

callback()

9.151.4.42 do_callback() [2/3]

```c
void Fl_Widget::do_callback ( Fl_Widget * o, long arg ) [inline]
```

Calls the widget callback.
Causes a widget to invoke its callback function with arbitrary arguments.

Parameters

| in  | o   | call the callback with o as the widget argument |
| in  | arg | call the callback with arg as the user data argument |

See also

callback()

9.151.4.43 do_callback() [3/3]

```c
void Fl_Widget::do_callback ( Fl_Widget * o, void * arg = 0 )
```

Calls the widget callback.
Causes a widget to invoke its callback function with arbitrary arguments.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>o</th>
<th>call the callback with o as the widget argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>use arg as the user data argument</td>
</tr>
</tbody>
</table>

See also

callback()

9.151.4.44 draw()

```cpp
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 = &scroll; // scroll is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```


9.151.4.45 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.

9.151.4.46 draw_label() [2/3]

```cpp
void Fl_Widget::draw_label(int X, 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.

9.151.4.47 draw_label() [3/3]

```cpp
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.

9.151.4.48 h() [1/2]

```cpp
int Fl_Widget::h() const [inline]
```

Gets the widget height.
Returns

the height of the widget in pixels.

9.151.4.49 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.

9.151.4.50 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.

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


9.151.4.51 hide()

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_MENU_WINDOW, Fl_OVERLAY_WINDOW, and Fl_WINDOW.

9.151.4.52 image() [1/3]

Fl_Image * Fl_Widget::image ( ) [inline]

Gets the image that is used as part of the widget label.
This image is used when drawing the widget in the active state.
9.151.4.53  image() [2/3]

void Fl_Widget::image (  
    Fl_Image & img  
) [inline]

Sets the image to use as part of the widget label.  
This image is used when drawing the widget in the active state.

Parameters

  in  img  the new image for the label

9.151.4.54  image() [3/3]

void Fl_Widget::image (  
    Fl_Image * img  
) [inline]

Sets the image to use as part of the widget label.  
This image is used when drawing the widget in the active state.

Parameters

  in  img  the new image for the label

9.151.4.55  inside()

int Fl_Widget::inside (  
    const Fl_Widget * wgt  
) const [inline]

Checks if this widget is a child of wgt.  
Returns 1 if this widget is a child of wgt, or is equal to wgt. Returns 0 if wgt is NULL.

Parameters

  in  wgt  the possible parent widget.

See also

  contains()

9.151.4.56  is_label_copied()

int Fl_Widget::is_label_copied ( ) const [inline]

Returns whether the current label was assigned with copy_label().  
This can be useful for temporarily overwriting the widget's label and restoring it later.

Return values

  0  current label was assigned with label().  
  1  current label was assigned with copy_label().
9.151.4.57  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 *)`

9.151.4.58  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. 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

| in | text | pointer to new label text |

See also

`copy_label()`

9.151.4.59  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)`

9.151.4.60  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 `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

| t | text or label to search for '&x' shortcut. |

Returns

Unicode (UCS-4) value of shortcut in `t` or 0.
Note
Internal use only.

9.151.4.61 labelcolor() [1/2]

\texttt{Fl\_Color Fl\_Widget::labelcolor ( ) const [inline]}

Gets the label color.
The default color is FL\_FOREGROUND\_COLOR.

Returns
the current label color

9.151.4.62 labelcolor() [2/2]

\texttt{void Fl\_Widget::labelcolor ( Fl\_Color c ) [inline]}

Sets the label color.
The default color is FL\_FOREGROUND\_COLOR.

Parameters
\begin{itemize}
  \item \texttt{in c} the new label color\end{itemize}

9.151.4.63 labelfont() [1/2]

\texttt{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 \texttt{Fl::set\_font()} can define new typefaces.

Returns
the current font used by the label

See also
\texttt{Fl\_Font}

9.151.4.64 labelfont() [2/2]

\texttt{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 \texttt{Fl::set\_font()} can define new typefaces.

Parameters
\begin{itemize}
  \item \texttt{in f} the new font for the label\end{itemize}
See also

\texttt{Fl_Font}

\subsection{labelsize() [1/2]}

\begin{verbatim}
Fl_Fontsize Fl_Widget::labelsize ( ) const [inline]
\end{verbatim}

Gets the font size in pixels.
The default size is 14 pixels.

Returns

the current font size

\subsection{labelsize() [2/2]}

\begin{verbatim}
void Fl_Widget::labelsize ( Fl_Fontsize pix ) [inline]
\end{verbatim}

Sets the font size in pixels.

Parameters

\begin{verbatim}
in pix the new font size
\end{verbatim}

See also

\texttt{Fl_Fontsize labelsize()}

\subsection{labeltype() [1/2]}

\begin{verbatim}
Fl_Labeltype Fl_Widget::labeltype ( ) const [inline]
\end{verbatim}

Gets the label type.

Returns

the current label type.

See also

\texttt{Fl_Labeltype}

\subsection{labeltype() [2/2]}

\begin{verbatim}
void Fl_Widget::labeltype ( Fl_Labeltype a ) [inline]
\end{verbatim}

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 \texttt{FL_NORMAL\_LABEL} prints the label as plain text.

Parameters

\begin{verbatim}
in a new label type
\end{verbatim}

See also

\texttt{Fl_Labeltype}
9.151.4.69 measure_label()

    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)

9.151.4.70 output()

    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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the widget is used for input and output</td>
</tr>
</tbody>
</table>

See also

    set_output(), clear_output()

9.151.4.71 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if the widget has no parent</td>
</tr>
</tbody>
</table>

See also

    Fl_Group::add(Fl_Widget*)

9.151.4.72 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.

9.151.4.73 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X, Y</td>
</tr>
</tbody>
</table>

See also

resize(int, int), size(int, int)

9.151.4.74 redraw()

void Fl_Widget::redraw ()
Schedules the drawing of the widget. Marks the widget as needing its draw() routine called.

9.151.4.75 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.

9.151.4.76 resize()

void Fl_Widget::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

<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)


9.151.4.77 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)

9.151.4.78 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 se-
lected, 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)

9.151.4.79 set_active()

void Fl_Widget::set_active ( ) [inline]
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()

9.151.4.80 set_changed()

void Fl_Widget::set_changed ( ) [inline]
Marks the value of the widget as changed.

See also

changed(), clear_changed()

9.151.4.81 set_output()

void Fl_Widget::set_output ( ) [inline]
Sets a widget to output only.

See also

output(), clear_output()

9.151.4.82 set_visible()

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.
9.151.4.83  set_visible_focus()

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)

9.151.4.84  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_Menu_Window, Fl_Overlay_Window,
   Fl_Single_Window, and Fl_Window.

9.151.4.85  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)

9.151.4.86  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.

9.151.4.87  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 |

9.151.4.88  test_shortcut() [1/2]

```cpp
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.

Returns

true, if the entered text matches the widget's '&x' shortcut, false (0) otherwise.

Note

Internal use only.

9.151.4.89  test_shortcut() [2/2]

```cpp
int Fl_Widget::test_shortcut ( const char * t, 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

Internal use only.

9.151.4.90  tooltip() [1/2]

```cpp
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*)
9.151.4.91 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()

9.151.4.92 top_window()

Fl_Window * 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()

9.151.4.93 top_window_offset()

Fl_Window * 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)

9.151.4.94 type() [1/2]

uchar Fl_Widget::type() const [inline]

Gets the widget type.
Returns the widget type value, which is used for Forms compatibility and to simulate RTTI.

Todo Explain "simulate RTTI" (currently only used to decide if a widget is a window, i.e. type() >= FL_WINDOW ?). Is type() really used in a way that ensures "Forms compatibility"?
9.151.4.95  type() [2/2]

void Fl_Widget::type (uchar t) [inline]

Sets the widget type.
This is used for Forms compatibility.

9.151.4.96  user_data() [1/2]

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

9.151.4.97  user_data() [2/2]

void Fl_Widget::user_data (void *v) [inline]

Sets the user data for this widget.
Sets the new user data (void *) argument that is passed to the callback function.

Parameters

|   | v   | new user data |

9.151.4.98  visible()

unsigned int Fl_Widget::visible ( ) const [inline]

Returns whether a widget is visible.

Return values

|   | if the widget is not drawn and hence invisible. |

See also
show(), hide(), visible_r()

9.151.4.99  visible_focus() [1/2]

unsigned int Fl_Widget::visible_focus ( ) [inline]

Checks whether this widget has a visible focus.

Return values

|   | if this widget has no visible focus. |
See also

visible_focus(int), set_visible_focus(), clear_visible_focus()

9.151.4.100 visible_focus() [2/2]

void Fl_Widget::visible_focus ( int v ) [inline]

Modifies keyboard focus navigation.

Parameters

| in v | set or clear visible focus |

See also

set_visible_focus(), clear_visible_focus(), visible_focus()

9.151.4.101 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()

9.151.4.102 w() [1/2]

int Fl_Widget::w ( ) const [inline]

Gets the widget width.

Returns

the width of the widget in pixels.

9.151.4.103 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.

9.151.4.104 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)
**when()** [2/2]

```cpp
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_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 |

**window()**

```cpp
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()`

**x()** [1/2]

```cpp
int Fl_Widget::x () const [inline]
```

Gets the widget position in its window.

**Returns**

the x position relative to the window
9.151.4.108 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.

9.151.4.109 y() [1/2]
int Fl_Widget::y ( ) const [inline]
Gets the widget position in its window.
Returns
the y position relative to the window

9.151.4.110 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,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

9.152 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.
#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.
9.152.1 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, if 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.

It is intended to be used as an automatic (local/stack) variable, such that the automatic 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 not needed any more.

Example:

```c
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.
    // It is safe to access the widget
    clear_changed(); // access the widget
  }
}
```

9.152.2 Member Function Documentation

9.152.2.1 deleted()

```c
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

```c
if (wp.widget() == 0) // ...
```

where wp is an Fl_Widget_Tracker object.

9.152.2.2 exists()

```c
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

```c
if (wp.widget() != 0) // ...
```

where wp is an Fl_Widget_Tracker object.

9.152.2.3 widget()

```c
Fl_Widget * Fl_Widget_Tracker::widget ( ) [inline]
```

Returns a pointer to the watched widget. This pointer is NULL, if the widget has been deleted.

The documentation for this class was generated from the following files:

- Fl.H
- Fl.cxx

9.153 Fl_Window Class Reference

This widget produces an actual window.

```c
#include <Fl_Window.H>
```

Inheritance diagram for Fl_Window:
Classes

- struct shape_data_type
  
  Data supporting a non-rectangular window shape.

Public Member Functions

- virtual Fl_Window ∗ as_window ()
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int border () const
  
  See void Fl_Window::border(int)
- 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.
- 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 ()
  
  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w ()
  
  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.
- Fl_Window (int w, int h, const char ∗title=0)
  
  Creates a window from the given size and title.
- Fl_Window (int x, int y, int w, int h, const char ∗title=0)
  
  Creates a window from the given position, size 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.

- **virtual** **int** **handle** (int)
  Handles the specified event.

- **virtual void** **hide** ()
  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)
  Sets the current icon window target dependent data.

- **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.

- **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.

- **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.
• unsigned int **override** () const  
  Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.

• virtual void **resize** (int X, int Y, int W, int H)  
  Changes the size and position of 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|FL_OVERRIDE.

• void **set_tooltip_window** ()  
  Marks the window as a tooltip window.

• 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.

• virtual void **show** ()  
  Puts the window on the screen.

• void **show** (int argc, char **argv)  
  Puts the window on the screen 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 **wait_forExpose** ()  
  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.

• virtual Fl_Group * as_group ()
  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
  See void Fl_Group::resizable(Fl_Widget +box)

• void **resizable** (Fl_Widget &o)
  See void Fl_Group::resizable(Fl_Widget +box)

• void **resizable** (Fl_Widget +o)
  The resizable widget defines the resizing box for the group.

• virtual ~Fl_Group ()
  The destructor also defines the resizing box for the group.
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.
- 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, void * p)
  Sets the current callback function 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.

- **const Fl_Image** `* deimage () const`

- **void** `deimage (Fl_Image &img)`
  
  Sets the image to use as part of the widget label.

- **void** `deimage (Fl_Image *img)`
  
  Sets the image to use as part of the widget label.

- **void** `do_callback ()`
  
  Calls the widget callback.

- **void** `do_callback (Fl_Widget *o, long arg)`
  
  Calls the widget callback.

- **void** `do_callback (Fl_Widget *o, void *arg=0)`
  
  Calls the widget callback.

- **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.

- **const Fl_Image** `* image () const`

- **void** `image (Fl_Image &img)`
  
  Sets the image to use as part of the widget label.

- **void** `image (Fl_Image *img)`
  
  Sets the image to use as part of the widget label.

- **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 labelszize () const
  Gets the font size in pixels.
• void labelszize (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.
• 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.
• 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(void *v)
  Sets the user data for this widget.

- unsigned int visible() const
  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() 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 *)
  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 const char * default_xclass ()
  Returns the default xclass.
- static void default_xclass (const char *)
  Sets the default window xclass.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()
  Returns the currently active group.
- static void current (Fl_Group *)
  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 Member Functions

- virtual void draw ()
  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.

Protected Member Functions inherited from Fl_Group

- 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 ()
  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_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

- shape_data_type ∗ shape_data
  non-null means the window has a non-rectangular shape

Static Protected Attributes

- static Fl_Window ∗ current_
  Stores the last window that was made current.

Friends

- 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,
  USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

9.153.1 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().

9.153.2 Constructor & Destructor Documentation

9.153.2.1 Fl_Window() [1/2]

Fl_Window::Fl_Window (int w, int h, const char * title = 0)

Creates a window from the given size 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)

9.153.2.2 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 position, size and title.

See also

\texttt{Fl\_Window(int \ w, int \ h, const char \* \ title)}

9.153.2.3 \texttt{\sim Fl\_Window()}

\texttt{Fl\_Window::\sim Fl\_Window();} \texttt{[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 \texttt{Fl\_Window} and all of its children can be automatic (local) variables, but you must declare the \texttt{Fl\_Window} first so that it is destroyed last.

9.153.3 Member Function Documentation

9.153.3.1 \texttt{as\_window()}

\texttt{virtual Fl\_Window* Fl\_Window::as\_window();} \texttt{[virtual]}

Returns an \texttt{Fl\_Window} pointer if this widget is an \texttt{Fl\_Window}.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from \texttt{Fl\_Window}. If it returns non-NULL, then the widget in question is derived from \texttt{Fl\_Window}, and you can use the returned pointer to access its children or other \texttt{Fl\_Window}-specific methods.

Return values

\texttt{NULL} if this widget is not derived from \texttt{Fl\_Window}.

Note

This method is provided to avoid \texttt{dynamic\_cast}.

See also

\texttt{Fl\_Widget::as\_group(), Fl\_Widget::as\_gl\_window()}

Reimplemented from \texttt{Fl\_Widget}.

9.153.3.2 \texttt{border()}

\texttt{void Fl\_Window::border(int \ b);} \texttt{[inline]}

Sets whether or not the window manager border is around the window.

The default value is true. \texttt{void border(int)} can be used to turn the border on and off. \textit{Under most X window managers this does not work after show() has been called, although SGI's 4DWM does work.}

9.153.3.3 \texttt{clear\_border()}

\texttt{void Fl\_Window::clear\_border();} \texttt{[inline]}

Fast inline function to turn the window manager border off.

It only works before \texttt{show()} is called.

9.153.3.4 \texttt{clear\_modal\_states()}

\texttt{void Fl\_Window::clear\_modal\_states();} \texttt{[inline]}

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

Note that there are \texttt{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 \texttt{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()`

### 9.153.3.5 current()

**FL_Window** ∗ **FL_Window:** current ( ) [static]

Returns the last window that was made current.

See also

- `FL_Window::make_current()`

### 9.153.3.6 cursor() [1/3]

```cpp
void FL_Window::cursor ( 
    const FL_RGB_Image ∗ image,
    int hotx,
    int hoty )
```

Changes the cursor for this window.

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.

See also

- `cursor(Fl_Cursor), default_cursor()`

### 9.153.3.7 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)`

### 9.153.3.8 cursor() [3/3]

```cpp
void FL_Window::cursor ( 
    Fl_Cursor c )
```

Changes the cursor for this window.

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()

9.153.3.9  decorated_h()

int Fl_Window::decorated_h ( )
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.

9.153.3.10  decorated_w()

int Fl_Window::decorated_w ( )
Returns the window width including any frame added by the window manager.
Same as w() if applied to a subwindow.

9.153.3.11  default_cursor() [1/2]

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)

9.153.3.12  default_cursor() [2/2]

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

cursor(const Fl_RGB_Image*, int, int), default_cursor()

9.153.3.13  default_icon()

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

Fl_Window::default_icons(const Fl_RGB_Image*[], int)
Fl_Window::icon(const Fl_RGB_Image*)
Fl_Window::icons(const Fl_RGB_Image*[], int)

9.153.3.14  default_icons()

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>icons</th>
<th>default icons for all windows subsequently created</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</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)`

### 9.153.3.15 default_xclass() [1/2]

```
const char * Fl_Window::default_xclass ( ) [static]
```

Returns the default xclass.

See also

```
Fl_Window::default_xclass(const char *)
```

### 9.153.3.16 default_xclass() [2/2]

```
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 *)
```

### 9.153.3.17 draw()

```
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:
FL_Widget *s = &scroll;  // scroll is an embedded Fl_Scrollbar
s->draw();       // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Group.
Reimplemented in Fl_Cairo_Window, Fl_GL_Window, and Fl_Glut_Window.

### 9.153.3.18 flush()

```c
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, Fl_Menu_Window, Fl_Overlay_Window, and Fl_Single_Window.

### 9.153.3.19 force_position() [1/2]

```c
int Fl_Window::force_position ( ) const [inline], [protected]
```
Returns the internal state of the window's FORCE_POSITION flag.

**Return values**

- 1 if flag is set
- 0 otherwise

See also

- force_position(int)

### 9.153.3.20 force_position() [2/2]

```c
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 |

### 9.153.3.21 free_icons()

```c
void Fl_Window::free_icons ( ) [protected]
```
Deletes all icons previously attached to the window.

See also

- Fl_Window::icons(const Fl_RGB_Image *icons[], int count)

### 9.153.3.22 free_position()

```c
void Fl_Window::free_position ( ) [inline]
```
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.
This is for Forms compatibility only.

**Deprecated** please use force_position(0) instead
9.153.23  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()

9.153.24  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()

9.153.25  handle()

int Fl_Window::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.

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.
Reimplemented in Fl_Gl_Window, and Fl_Glut_Window.
9.153.3.26  hide()

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.
Reimplemented in Fl_Double_Window, Fl_Gl_Window, Fl_Menu_Window, and Fl_Overlay_Window.

9.153.3.27  hotspot()

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()

9.153.3.28  icon() [1/3]

const void * Fl_Window::icon ( ) const
Gets the current icon window target dependent data.

Deprecated in 1.3.3

9.153.3.29  icon() [2/3]

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.

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)

9.153.3.30  icon() [3/3]

void Fl_Window::icon ( const void * ic )
Sets the current icon window target dependent data.

Deprecated in 1.3.3
9.153.31 iconize()

```
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.

9.153.32 icons()

```
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

<table>
<thead>
<tr>
<th>in</th>
<th>icons</th>
<th>icons for this window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>number of images in icons. Set to 0 to remove the current icons</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 ∗)

9.153.33 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.

9.153.34 modal()

```
unsigned int Fl_Window::modal ( ) const [inline]
```

Returns true if this window is modal.

9.153.35 resize()

```
virtual 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`.

Reimplemented in `Fl_Double_Window`, `Fl_Gl_Window`, and `Fl_Overlay_Window`.

### 9.153.3.36 set_menu_window()

```cpp
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.

### 9.153.3.37 set_modal()

```cpp
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()`.

### 9.153.3.38 set_non_modal()

```cpp
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.

### 9.153.3.39 set_tooltip_window()

```cpp
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.

### 9.153.3.40 shape() [1/2]

```cpp
void Fl_Window::shape ( const Fl_Image & b ) [inline]
```

Set the window's shape with an `Fl_Image`.

See also

```cpp
void shape(const Fl_Image* img)
```
9.153.3.41 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 dimension 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_Shared_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 MSWindows 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 (and requires compilation with FLTK_ABI_VERSION >= 10303)

9.153.3.42 show() [1/2]

virtual void Fl_Window::show() [virtual]

Puts the window on the screen.
Usually (on X) this has the side effect of opening the display.
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

\texttt{Fl\_Window::show(int argc, char \ast\ast argv)}

Reimplemented from \texttt{Fl\_Widget}.
Reimplemented in \texttt{Fl\_Double\_Window}, \texttt{Fl\_Gl\_Window}, \texttt{Fl\_Menu\_Window}, \texttt{Fl\_Overlay\_Window}, and \texttt{Fl\_Single\_Window}.

### 9.153.3.43 \texttt{show()} [2/2]

void \texttt{Fl\_Window::show (}
\begin{verbatim}
   int argc,
   char \ast\ast argv)
\end{verbatim}

Puts the window on the screen and parses command-line arguments.
Usually (on X) this has the side effect of opening the display.
This form should be used for top-level windows, at least for the first (main) window. It allows standard arguments to
be parsed from the command-line. You can use \texttt{argc} and \texttt{argv} from \texttt{main(int argc, char \ast\ast argv)} for this call.
The first call also sets up some system-specific internal variables like the system colors.

\textbf{Todo} explain which system parameters are set up.

#### Parameters

<table>
<thead>
<tr>
<th>\texttt{argc}</th>
<th>command-line argument count, usually from main()</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{argv}</td>
<td>command-line argument vector, usually from main()</td>
</tr>
</tbody>
</table>

See also

\texttt{virtual void Fl\_Window::show()}

### 9.153.3.44 \texttt{shown()}

int \texttt{Fl\_Window::shown ( )} [inline]

Returns non-zero if \texttt{show()} has been called (but not \texttt{hide()}).
You can tell if a window is iconified with \texttt{(w->shown() && !w->visible())}.

### 9.153.3.45 \texttt{size\_range()}

void \texttt{Fl\_Window::size\_range (}
\begin{verbatim}
   int minw,
   int minh,
   int maxw = 0,
   int maxh = 0,
   int dw = 0,
   int dh = 0,
   int aspect = 0) [inline]
\end{verbatim}

Sets the allowable range the user can resize this window to.
This only works for top-level windows.

- \texttt{minw} and \texttt{minh} are the smallest the window can be. Either value must be greater than 0.
- \texttt{maxw} and \texttt{maxh} are the largest the window can be. If either is \texttt{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.
- \texttt{dw} and \texttt{dh} are size increments. The window will be constrained to widths of \texttt{minw + N \times dw}, where \texttt{N} is any
  non-negative integer. If these are less or equal to 1 they are ignored (this is ignored on WIN32).
- \texttt{aspect} is 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 WIN32 and by many X window managers).
If this function is not called, FLTK tries to figure out the range from the setting of \texttt{resizable()}:

- If \texttt{resizable()} is \texttt{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.

- If either dimension of \texttt{resizable()} is less than 100, then that is considered the minimum size. Otherwise the \texttt{resizable}() has a minimum size of 100.

- If either dimension of \texttt{resizable()} is zero, then that is also the maximum size (so the window cannot resize in that direction).

It is undefined what happens if the current size does not fit in the constraints passed to \texttt{size_range()}.

### 9.153.3.46 \texttt{wait_for_expose()}

```cpp
void Fl_Window::wait_for_expose ( )
```

Waits for the window to be displayed after calling \texttt{show()}. \texttt{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, \texttt{show()} is always synchronous. The effect of \texttt{show()} varies with versions of Mac OS X: early versions have the window appear on the screen when \texttt{show()} returns, later versions don't. If you want to write portable code and need this synchronous \texttt{show()} feature, add \texttt{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 \texttt{Fl::run()} or for having exact control over which window has the focus after calling \texttt{show()}. If the window is not \texttt{shown()}, this method does nothing.

**Note**

Depending on the platform and window manager \texttt{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 \texttt{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 \texttt{Fl::flush()} after \texttt{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 \texttt{wait_for_expose()} and \texttt{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 \texttt{Fl::run()}.

**See also**

- \texttt{virtual void Fl_Window::show()}
- Example code for displaying a window before calling \texttt{Fl::run()}
  ```cpp
  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 \texttt{Fl::run()}.

### 9.153.3.47 \texttt{xclass()} [1/2]

```cpp
const char * Fl_Window::xclass ( ) const
```

Returns the xclass for this window, or a default.

**See also**

- \texttt{Fl_Window::default_xclass(const char *)}
- \texttt{Fl_Window::xclass(const char *)}
9.153.48  xclass() [2/2]

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.

See also

Fl_Window::default_xclass(const char *)

9.153.4  Member Data Documentation

9.153.4.1  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.cxx
- Fl_arg.cxx
- fl_cursor.cxx
- Fl_Window.cxx
- Fl_Window_fullscreen.cxx
- Fl_Window_hotspot.cxx
- Fl_Window_iconize.cxx
- Fl_Window_shape.cxx

9.154  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.

- **virtual Fl_Group ∗ as_group ()**
  
  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.

- **int** handle (int)
  
  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 &w, Fl_Widget *before)
  
  This does insert(w, find(before)).

• void remove (Fl_Widget &w)
  
  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
  
  See void Fl_Group::resizable(Fl_Widget *box)

• void resizable (Fl_Widget &o)
  
  See void Fl_Group::resizable(Fl_Widget *box)

• void resizable (Fl_Widget *o)
  
  The resizable widget defines the resizing box for the group.

  void resize (int, int, int, int)

  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 Fl_Window *as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.

• 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, 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)
  
  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.
• const Fl_Image *deimage () const
• void deimage (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void deimage (Fl_Image *img)
  Sets the image to use as part of the widget label.
• void do_callback()
  Calls the widget callback.
• void do_callback (Fl_Widget *o, long arg)
  Calls the widget callback.
• void do_callback (Fl_Widget *o, void *arg=0)
  Calls the widget callback.
• 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.
• const Fl_Image *image () const
• void image (Fl_Image &img)
  Sets the image to use as part of the widget label.
• void image (Fl_Image *img)
  Sets the image to use as part of the widget label.
• 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.
• 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.
• 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 (void ∗v)
  
  Sets the user data for this widget.

- unsigned int visible () const
  
  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 () 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 ∗cb, void ∗d)
  
  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,
USERFLAG3 = 1 << 29, USERFLAG2 = 1 << 30, USERFLAG1 = 1 << 31
}
flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

• 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 ()
  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_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.
9.154.1 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.

9.154.2 Constructor & Destructor Documentation

9.154.2.1 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.

9.154.3 Member Function Documentation

9.154.3.1 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

9.155 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
   ↓
Fl_XBM_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

- Fl_Image ∗ copy ()
- virtual Fl_Image ∗ copy (int W, int H)

  The copy() method creates a copy of the specified image.
• void **draw** (int X, int Y)
  
  Draws the image with a bounding box.

• **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 W, int H)
  
  The constructors create a new bitmap from the specified bitmap data.

• virtual void **label** (Fl_Menu_Item *m)
  
  The **label()** methods are an obsolete way to set the image attribute of a widget or menu item.

• virtual void **label** (Fl_Widget *w)
  
  The **label()** methods are an obsolete way to set the image attribute of a widget or menu item.

• virtual void **uncache** ()
  
  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 void **color_average** (Fl_Color c, float i)
  
  The **color_average()** method averages the colors in the image with the FLTK color value c.

• **Fl_Image** *copy ()
  
  The **copy()** method creates a copy of the specified image.

• int **count** () const
  
  The **count()** method returns the number of data values associated with the image.

• int **d** () const
  
  Returns the current image depth.

• const char *const *data () const
  
  Returns a pointer to the current image data array.

• virtual void **desaturate** ()
  
  The **desaturate()** method converts an image to grayscale.

• void **draw** (int X, int Y)
  
  Draws the image.

• int **fail** ()
  
  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 height in pixels.

• 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.

• int **w** () const
  
  Returns the current image width in pixels.

• 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_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).

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_NO_IMAGE = -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 array 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.

- void h (int H)
  
  Sets the current image height in pixels.

- void ld (int LD)
  
  Sets the current line data size in bytes.

- void w (int W)
  
  Sets the current image width in pixels.

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)

9.155.1 Detailed Description

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

9.155.2 Constructor & Destructor Documentation

9.155.2.1 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
9.156 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

9.157 Fl_Xlib_Graphics_Driver Class Reference

The Xlib-specific graphics class.

#include <Fl_Device.H>

Inheritance diagram for Fl_Xlib_Graphics_Driver:

```
  Fl_Device
   └── Fl_Graphics_Driver
       └── Fl_Xlib_Graphics_Driver
```

Public Member Functions

- const char * class_name ()
  Returns the name of the class of this object.
- void color (Fl_Color c)
  see fl_color(Fl_Color c).
- void color (uchar r, uchar g, uchar b)
  see fl_color(uchar r, uchar g, uchar b).
- void copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  see fl_copy_offscreen()
- int descent ()
  see fl_descent().
- void draw (const char *str, int n, int x, int y)
  see fl_draw(const char *str, int n, int x, int y).
- void draw (Fl_Bitmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Bitmap object to the device.
- void draw (Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_Pixmap object to the device.
- void draw (Fl_RGB_Image *img, int XP, int YP, int WP, int HP, int cx, int cy)
  Draws an Fl_RGB_Image object to the device.
- void draw (int angle, const char *str, int n, int x, int y)
  see fl_draw(int angle, const char *str, int n, int x, int y).
- void draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  see fl_draw_image(const uchar* buf, int X, int Y, int W, int H, int D, int L).
- void draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  see fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D).

Generated by Doxygen
• void draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)
  see fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D, int L).
• void draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)
  see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D).
• void font (Fl_Font face, Fl_Fontsize size)
  see fl_font(Fl_Font face, Fl_Fontsize size).
• int height ()
  see fl_height().
• void rtl_draw (const char *str, int n, int x, int y)
  see fl_rtl_draw(const char *str, int n, int x, int y).
• void text_extents (const char *, int n, int &dx, int &dy, int &w, int &h)
  see fl_text_extents(const char *, int n, int &dx, int &dy, int &w, int &h).
• double width (const char *str, int n)
  see fl_width(const char *str, int n).
• double width (unsigned int c)
  see fl_width(unsigned int n).

Public Member Functions inherited from Fl_Graphics_Driver

• Fl_Color color ()
  see fl_color(void).
• virtual int draw_scaled (Fl_Image *img, int X, int Y, int W, int H)
  Draws an Fl_Image scaled to width W & height H with top-left corner at X,Y.
• Fl_Font font ()
  see fl_font(void).
• Fl_Font_Descriptor * font_descriptor ()
  Returns a pointer to the current Fl_Font_Descriptor for the graphics driver.
• void font_descriptor (Fl_Font_Descriptor *d)
  Sets the current Fl_Font_Descriptor for the graphics driver.
• Fl_Fontsize size ()
  see fl_size().
• virtual ~Fl_Graphics_Driver ()
  The destructor.

Public Member Functions inherited from Fl_Device

• virtual ~Fl_Device ()
  Virtual destructor.

Static Public Attributes

• static const char * class_id = "Fl_Xlib_Graphics_Driver"

Static Public Attributes inherited from Fl_Graphics_Driver

• static const char * class_id = "Fl_Graphics_Driver"

Static Public Attributes inherited from Fl_Device

• static const char * class_id = "Fl_Device"
  A string that identifies each subclass of Fl_Device.
Additional Inherited Members

Protected Member Functions inherited from Fl_Graphics_Driver

- virtual void arc (double x, double y, double r, double start, double end)
  
  see fl_arc(double x, double y, double r, double start, double end).

- virtual void arc (int x, int y, int w, int h, double a1, double a2)
  
  see fl_arc(int x, int y, int w, int h, double a1, double a2).

- virtual void begin_complex_polygon ()

  see fl_begin_complex_polygon().

- virtual void begin_line ()

  see fl_begin_line().

- virtual void begin_loop ()

  see fl_begin_loop().

- virtual void begin_points ()

  see fl_begin_points().

- virtual void begin_polygon ()

  see fl_begin_polygon().

- virtual void circle (double x, double y, double r)

  see fl_circle(double x, double y, double r).

- virtual void clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

  see fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H).

- Fl_Region clip_region ()

  see fl_clip_region().

- void clip_region (Fl_Region r)

  see fl_clip_region(Fl_Region r).

- virtual void curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

  see fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3).

- virtual void end_complex_polygon ()

  see fl_end_complex_polygon().

- virtual void end_line ()

  see fl_end_line().

- virtual void end_loop ()

  see fl_end_loop().

- virtual void end_points ()

  see fl_end_points().

- virtual void end_polygon ()

  see fl_end_polygon().

- Fl_Graphics_Driver ()

  The constructor.

- virtual void gap ()

  see fl_gap().

- virtual void line (int x, int y, int x1, int y1)

  see fl_line(int x, int y, int x1, int y1).

- virtual void line (int x, int y, int x1, int y1, int x2, int y2)

  see fl_line(int x, int y, int x1, int y1, int x2, int y2).

- virtual void line_style (int style, int width=0, char *dashes=0)

  see fl_line_style(int style, int width, char *dashes).

- virtual void loop (int x0, int y0, int x1, int y1, int x2, int y2)

  see fl_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)

  see fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).
• void mult_matrix (double a, double b, double c, double d, double x, double y)
  see fl_mult_matrix(double a, double b, double c, double d, double x, double y).
• virtual int not_clipped (int x, int y, int w, int h)
  see fl_not_clipped(int x, int y, int w, int h).
• virtual void pie (int x, int y, int w, int h, double a1, double a2)
  see fl_pie(int x, int y, int w, int h, double a1, double a2).
• virtual void point (int x, int y)
  see fl_point(int x, int y).
• virtual void polygon (int x0, int y0, int x1, int y1, int x2, int y2)
  see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2).
• virtual void polygon (int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3)
  see fl_polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3).
• virtual void pop_clip ()
  see fl_pop_clip().
• void pop_matrix ()
  see fl_pop_matrix().
• virtual void push_clip (int x, int y, int w, int h)
  see fl_push_clip(int x, int y, int w, int h).
• void push_matrix ()
  see fl_push_matrix().
• virtual void push_no_clip ()
  see fl_push_no_clip().
• virtual void rect (int x, int y, int w, int h)
  see fl_rect(int x, int y, int w, int h).
• virtual void rectf (int x, int y, int w, int h)
  see fl_rectf(int x, int y, int w, int h).
• void restore_clip ()
  see fl_restore_clip().
• void rotate (double d)
  see fl_rotate(double d).
• void scale (double x)
  see fl_scale(double x).
• void scale (double x, double y)
  see fl_scale(double x, double y).
• double transform_dx (double x, double y)
  see fl_transform_dx(double x, double y).
• double transform_dy (double x, double y)
  see fl_transform_dy(double x, double y).
• double transform_x (double x, double y)
  see fl_transform_x(double x, double y).
• double transform_y (double x, double y)
  see fl_transform_y(double x, double y).
• virtual void transformed_vertex (double xf, double yf)
  see fl_transformed_vertex(double xf, double yf).
• void translate (double x, double y)
  see fl_translate(double x, double y).
• virtual void vertex (double x, double y)
  see fl_vertex(double x, double y).
• virtual void xyline (int x, int y, int x1)
  see fl_xyline(int x, int y, int x1).
• virtual void xyline (int x, int y, int x1, int y2)
see \texttt{fl}_\texttt{xyline}(\texttt{x, y, x1, y2}).
• virtual void \texttt{xyline} (\texttt{x, y, x1, y2, x3})
  see \texttt{fl}_\texttt{xyline}(\texttt{x, y, x1, y2, x3}).
• virtual void \texttt{yxline} (\texttt{x, y, y1})
  see \texttt{fl}_\texttt{yxline}(\texttt{x, y, y1}).
• virtual void \texttt{yxline} (\texttt{x, y, y1, x2})
  see \texttt{fl}_\texttt{yxline}(\texttt{x, y, y1, x2}).
• virtual void \texttt{yxline} (\texttt{x, y, y1, x2, y3})
  see \texttt{fl}_\texttt{yxline}(\texttt{x, y, y1, x2, y3}).

Protected Attributes inherited from \texttt{FL_Graphics_Driver}
  • \texttt{matrix} * \texttt{fl_matrix}
    Points to the current coordinate transformation matrix.

9.157.1 Detailed Description
The Xlib-specific graphics class.
This class is implemented only on the Xlib platform.

9.157.2 Member Function Documentation

9.157.2.1 \texttt{class_name()}
\begin{verbatim}
const char * Fl_Xlib_Graphics_Driver::class_name ( ) [inline], [virtual]
\end{verbatim}
Returns the name of the class of this object.
Use of the \texttt{class_name()} function is discouraged because it will be removed from future FLTK versions.
The class of an instance of an \texttt{FL_Device} subclass can be checked with code such as:
\begin{verbatim}
if ( instance->class_name() == Fl_Printer::class_id ) {...}
\end{verbatim}
Reimplemented from \texttt{FL_Graphics_Driver}.

9.157.2.2 \texttt{color() [1/2]}
\begin{verbatim}
void Fl_Xlib_Graphics_Driver::color ( 
    Fl_Color c ) [virtual]
\end{verbatim}
see \texttt{fl}_\texttt{color}(\texttt{Fl_Color c}).
Reimplemented from \texttt{FL_Graphics_Driver}.

9.157.2.3 \texttt{color() [2/2]}
\begin{verbatim}
void Fl_Xlib_Graphics_Driver::color ( 
    uchar r, 
    uchar g, 
    uchar b ) [virtual]
\end{verbatim}
see \texttt{fl}_\texttt{color}(uchar r, uchar g, uchar b).
Reimplemented from \texttt{FL_Graphics_Driver}.

9.157.2.4 \texttt{copy_offscreen()}
\begin{verbatim}
void Fl_Xlib_Graphics_Driver::copy_offscreen ( 
    int x, 
    int y, 
    int w, 
    int h, 
    Fl_Offscreen pixmap, 
    int srcx, 
    int srcy ) [virtual]
\end{verbatim}
see \texttt{fl}_\texttt{copy_offscreen()}
Reimplemented from \texttt{FL_Graphics_Driver}.

Generated by Doxygen
9.157.2.5 descent()

int Fl_Xlib_Graphics_Driver::descent() [virtual]
see fl_descent()
Reimplemented from Fl_Graphics_Driver.

9.157.2.6 draw() [1/5]

void Fl_Xlib_Graphics_Driver::draw(const char *str, int n, int x, int y) [virtual]
see fl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.157.2.7 draw() [2/5]

void Fl_Xlib_Graphics_Driver::draw(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
Draws an Fl_Bitmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.157.2.8 draw() [3/5]

void Fl_Xlib_Graphics_Driver::draw(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
Draws an Fl_Pixmap object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.

9.157.2.9 draw() [4/5]

void Fl_Xlib_Graphics_Driver::draw(Fl_RGB_Image *rgb, int XP, int YP, int WP, int HP, int cx, int cy) [virtual]
Draws an Fl_RGB_Image object to the device.
Specifies a bounding box for the image, with the origin (upper left-hand corner) of the image offset by the cx and cy arguments.
Reimplemented from Fl_Graphics_Driver.
9.157.2.10  draw() [5/5]

```cpp
void Fl_Xlib_Graphics_Driver::draw (  
    int angle,  
    const char * str,  
    int n,  
    int x,  
    int y ) [virtual]
```

See `fl_draw(int angle, const char * str, int n, int x, int y)`.
Reimplemented from `Fl_Graphics_Driver`.

9.157.2.11  draw_image() [1/2]

```cpp
void Fl_Xlib_Graphics_Driver::draw_image (  
    const uchar * buf,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3,  
    int L = 0 ) [virtual]
```

See `fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L)`.
Reimplemented from `Fl_Graphics_Driver`.

9.157.2.12  draw_image() [2/2]

```cpp
void Fl_Xlib_Graphics_Driver::draw_image (  
    Fl_Draw_Image_Cb cb,  
    void * data,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 3 ) [virtual]
```

See `fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D)`.
Reimplemented from `Fl_Graphics_Driver`.

9.157.2.13  draw_image_mono() [1/2]

```cpp
void Fl_Xlib_Graphics_Driver::draw_image_mono (  
    const uchar * buf,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 1,  
    int L = 0 ) [virtual]
```

See `fl_draw_image_mono(const uchar * buf, int X, int Y, int W, int H, int D, int L)`.
Reimplemented from `Fl_Graphics_Driver`.

9.157.2.14  draw_image_mono() [2/2]

```cpp
void Fl_Xlib_Graphics_Driver::draw_image_mono (  
    Fl_Draw_Image_Cb cb,  
    void * data,  
    int X,  
    int Y,  
    int W,  
    int H,  
    int D = 1,  
    int L = 0 ) [virtual]
```

See `fl_draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D, int L)`.
Reimplemented from `Fl_Graphics_Driver`.

Generated by Doxygen
int H,
int D = 1 ) [virtual]
see fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D). Reimplemented from Fl_Graphics_Driver.

9.157.2.15 font()

void Fl_Xlib_Graphics_Driver::font {
  Fl_Font face,
  Fl_Fontsize fsize } [virtual]
see fl_font(Fl_Font face, Fl_Fontsize size).
Reimplemented from Fl_Graphics_Driver.

9.157.2.16 height()

int Fl_Xlib_Graphics_Driver::height ( ) [virtual]
see fl_height().
Reimplemented from Fl_Graphics_Driver.

9.157.2.17 rtl_draw()

void Fl_Xlib_Graphics_Driver::rtl_draw {
  const char * str, 
  int n, 
  int x, 
  int y ) [virtual]
see fl_rtl_draw(const char *str, int n, int x, int y).
Reimplemented from Fl_Graphics_Driver.

9.157.2.18 text_extents()

void Fl_Xlib_Graphics_Driver::text_extents {
  const char * t, 
  int n, 
  int & dx, 
  int & dy, 
  int & w, 
  int & h } [virtual]
see fl_text_extents(const char*, int n, int& dx, int& dy, int& w, int& h).
Reimplemented from Fl_Graphics_Driver.

9.157.2.19 width() [1/2]

double Fl_Xlib_Graphics_Driver::width {
  const char * str, 
  int n ) [virtual]
see fl_width(const char *str, int n).
Reimplemented from Fl_Graphics_Driver.

9.157.2.20 width() [2/2]

double Fl_Xlib_Graphics_Driver::width {
  unsigned int c ) [virtual]
see fl_width(unsigned int n).
Reimplemented from Fl_Graphics_Driver.
The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Bitmap.cxx
- fl_color.cxx
9.158 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_Image
Fl_Pixmap
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

- `virtual void color_average (Fl_Color c, float i)`
  
  The color_average() method averages the colors in the image with the FLTK color value c.

- `Fl_Image * copy ()`
  
  The copy() method creates a copy of the specified image.

- `virtual Fl_Image * copy (int W, int H)`
  
  The copy() method creates a copy of the specified image.

- `virtual void desaturate ()`
  
  The desaturate() method converts an image to grayscale.

- `void draw (int X, int Y)`
  
  Draws the image 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.

- `virtual void label (Fl_Menu_Item *m)`
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- `virtual void label (Fl_Widget *w)`
  
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- `virtual void uncache ()`
  
  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

- Fl/Image * copy ()
  The copy() method creates a copy of the specified image.
- int count () const
  The count() method returns the number of data values associated with the image.
- int d () const
  Returns the current image depth.
- const char * const * data () const
  Returns a pointer to the current image data array.
- void draw (int X, int Y)
  Draws the image.
- int fail ()
  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 height in pixels.
- 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.
- int w () const
  Returns the current image width in pixels.
- 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_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).

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_NO_IMAGE = -1

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 array 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.
- **void h (int H)**
  - Sets the current image height in pixels.
- **void ld (int LD)**
  - Sets the current line data size in bytes.
- **void w (int W)**
  - Sets the current image width in pixels.

### 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)**

### 9.158.1 Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

### 9.158.2 Constructor & Destructor Documentation

#### 9.158.2.1 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.hxx

### 9.159 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
9.159.1 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

9.160 Fl_Graphics_Driver::matrix Struct Reference

A 2D coordinate transformation matrix.
#include <Fl_Device.H>

Public Attributes

- double a
- double b
- double c
- double d
- double x
- double y

9.160.1 Detailed Description

A 2D coordinate transformation matrix.
The documentation for this struct was generated from the following file:

- Fl_Device.H

9.161 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.

9.161.1 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.
9.161.2 Constructor & Destructor Documentation

9.161.2.1 Name() [1/2]

```cpp
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.

```cpp
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], "" );
```

9.161.2.2 Name() [2/2]

```cpp
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'.

```cpp
int n, i;
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
prefs.get( "nFiles", 0 );
for ( i=0; i<n; i++ )
  prev.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

9.162 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 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)

Generated by Doxygen
Static Public Attributes

- static int lastEntrySet = -1

The documentation for this class was generated from the following files:

- FL_Preferences.H
- FL_Preferences.cxx

9.163 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

9.163.1 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

9.164 Fl_Preferences::RootNode Class Reference

Public Member Functions

- char getPath (char *path, int pathlen)
- int read ()
- RootNode (Fl_Preferences *)
- RootNode (Fl_Preferences *, const char *path, const char *vendor, const char *application)
- RootNode (Fl_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

9.165 Fl_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
- `Fl.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
- `Fl.Scrollbar_Data vscroll`
  vertical scrollbar region + values

9.165.1 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.

Note

Availability in FLTK_ABI_VERSION 10303 or higher.

The documentation for this struct was generated from the following file:

- `Fl_Scroll.H`

9.166 Fl_Window::shape_data_type Struct Reference

Data supporting a non-rectangular window shape.

#include <Fl_Window.H>

Public Attributes

- `int lh_`
  height of shape image
- `int lw_`
  width of shape image
- `Fl_Image * shape_`
  shape image
- `Fl_Bitmap * todelete_`
  auxiliary bitmap image

9.166.1 Detailed Description

Data supporting a non-rectangular window shape. The documentation for this struct was generated from the following file:

- `Fl_Window.H`
9.167  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.

```c
#include <Fl_Text_Display.H>
```

Public Attributes

- `unsigned attr`  
  currently unused (this may be change in the future)
- `Fl_Color color`  
  text color
- `Fl_Font font`  
  text font
- `Fl_Fontsize size`  
  text font size

9.167.1  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
Chapter 10

File Documentation

10.1 abi-version.h

```c
/* FL/abi-version.h. Generated from abi-version.in by configure. */

/*
ABI Configuration file for the Fast Light Tool Kit (FLTK).

DO NOT EDIT - This file is generated by configure!

define FL_ABI_VERSION: 1xxyy for 1.x.y (xx,yy with leading zero)
*/

#undef FL_ABI_VERSION
```

10.2 dirent.h

```c
// $Id$
// Directory header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2011 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
// this file is for back-compatibility only
#include "filename.H"
```

10.3 Enumerations.H File Reference

This file contains type definitions and general enumerations.

```c
#include <FL/abi-version.h>
#include "Fl_Export.H"
#include "fl_types.h"
```

Macros

Mouse and Keyboard Events
This and the following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events.

\todo FL_Button and FL_key... constants could be structured better (use an enum or some doxygen grouping ?)

\sa 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_Isu_Key 0xff0c
  The additional key of ISO 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 /* Previous track */`
- `#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.

```
\sa 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`
  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_COMMAND FL_CTRL`
  An alias for FL_CTRL on WIN32 and X11, or FL_META on MacOS X.
- `#define FL_CONTROL FL_META`
  An alias for FL_META on WIN32 and X11, or FL_CTRL on MacOS X.
- `#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.

Enumerations

When Conditions

```c
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 }
```

These constants determine when a callback is performed.

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.

```c
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 = 8 , 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 }
```
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

- `#define FL_MAJOR_VERSION 1`
  The major release version of this FLTK library.
- `#define FL_MINOR_VERSION 3`
  The minor release version for this library.
- `#define FL_PATCH_VERSION 9`
  The patch version for this library.
- `#define FL_VERSION`
  The FLTK version number as a double.
- `#define FLTK_ABI_VERSION FL_ABI_VERSION`

**Box Types**

FLTK standard box types

This enum defines the standard box types included with FLTK. 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 Figure 1 is the area that is not drawn by the frame types.

![Figure 10.1 FLTK standard box types](image)

**Todo** Description of box types is incomplete. See below for the defined enum Fl_Boxtype.

See also

src/Fl_get_system_colors.cxx

- `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_ENGRAVED_FRAME,
    FL_ENGRAVED_BOX, FL_EMBOSSED_BOX, FL_EMBOSSED_FRAME,
    FL_BORDER_BOX, FL_SHADOW_BOX,
    FL_BORDER_FRAME, FL_SHADOW_FRAME,
    FL_ROUNDED_BOX, FL_OSHADOW_BOX,
    FL_ROUNDED_FRAME, FL_OFLAT_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,
    ...
}Generated by Doxygen
• FL_CIRCLE_BOX
  • #define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
• FL_EXPORT Fl_Boxtype fl_define_FL_DIAMOND_BOX()
  • #define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX
  • #define FL_DIAMOND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)
• FL_EXPORT Fl_Boxtype fl_define_FL_ENGRAVED_LABEL()
• FL_LABELTYPE FL_EXPORT fl_define_FL_ENGRAVED_LABEL()
  • #define FL_ENGRAVED_LABEL FL_ENGRAVED_FRAME
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  • #define FL_GLEAM_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
  • #define FL_GLEAM_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+6)
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • #define FL_GTK_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+1)
  • #define FL_GTK_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+3)
  • #define FL_GTK_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)
  • #define FL_GTK_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+8)
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_FRAME()
  • #define FL_GTK_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+5)
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_FRAME()
  • #define FL_GTK_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+7)
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • #define FL_GTK_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+4)
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • #define FL_GTK_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+6)
• FLEXPORT FL_FREE_BOXTYPE

• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
  • FL_EXPORT Fl_Boxtype fl界定 (Fl_Boxtype b)
    • Get the "pressed" or "down" version of a box.
  • #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()
  • #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()
• FL_LABELTYPE FL_EXPORT fl_define_FL_ENGRAVED_LABEL()
  • 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
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+9)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+8)
• #define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+9)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+8)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+4)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
"Get the "pressed" or "down" version of a box.
• #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()
  • #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()
• 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
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+9)
  • #define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+8)
• #define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+9)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+8)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)
  • #define FL_GLEAM_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+4)
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()
"Get the "pressed" or "down" version of a box.
• #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()
  • #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()
• FL_BOXTYPE fl_frame (Fl_Boxtype b)
  "Get the unfilled, frame only version of a box."
#10.3 Enumerations.H File Reference

- `#define FL_GTK_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+2)
- 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.
- `#define FL_OFLAT_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+3)
- `#define FL_OSHADOW_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+1)
- `#define FL_OVAL_BOX fl_define_FL_OVAL_BOX()
- `#define FL_OVAL_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)
- `#define FL_PLASTIC_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+1)
- `#define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+3)
- `#define FL_PLASTIC_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)
- `#define FL_PLASTIC_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
- `#define FL_PLASTIC_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+5)
- `#define FL_PLASTIC_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+4)
- `#define FL_PLASTIC_UP_BOX fl_define_FL_PLASTIC_UP_BOX()
- `#define FL_PLASTIC_UP_FRAME FL_PLASTIC_UP_BOX
- `#define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()
- `#define FL_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_ROUND_UP_BOX()+1)
- `#define FL_ROUND_UP_BOX fl_define_FL_ROUND_UP_BOX()
- `#define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()
- `#define FL_ROUNDED_FRAME (Fl_Boxtype)(fl_define_FL_ROUNDED_BOX()+2)
- `#define FL_RSHADOW_BOX fl_define_FL_RSHADOW_BOX()
- `#define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()
- `#define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)
- `#define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()
- `#define FL_SYMBOL_LABEL FL_NORMAL_LABEL

Sets the current label type and return 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.

\[
Fl\_Color \rightarrow 0xrrggbbii
\]

| | | |
| | | +---- 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 FLBACKGROUND2_COLOR = 7`
  the default background color for text, list, and valuator widgets
- `const Fl_Color FLBACKGROUND_COLOR = 49`
- `const Fl_Color FLBLACK = 56`
- `const Fl_Color FLBLUE = 216`
- `typedef unsigned int Fl\_Color`

An FLTK color value; see also Colors

- `FL_EXPORT 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_EXPORT Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg)`
  Returns a color that contrasts with the background color.
- `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`
- `#define FL_GRAY FL_BACKGROUND_COLOR`
- `const Fl_Color FL_GRAY0 = 32`
- `#define FL_GRAY_RAMP (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_GREEN = 63`
- `FL_EXPORT Fl_Color fl_inactive (Fl_Color c)`
  Returns the inactive, dimmed version of the given 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.
- `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`
Cursors

- enum { FL_READ = 1, FL_WRITE = 4, FL_EXCEPT = 8 }
  
  "when" conditions.

- 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.

- 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.

- #define FL_IMAGE_WITH_ALPHA 0x40000000

- 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
}

visual types and FI_Gl_Window::mode() (values match Glut)

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.

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):

```
+---------------------------------+
<table>
<thead>
<tr>
<th>TOP_LEFT</th>
<th>TOP</th>
<th>TOP_RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_TOP</td>
<td>RIGHT_TOP</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>LEFT</td>
<td>CENTER</td>
<td>RIGHT</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>LEFT_BOTTOM</td>
<td>RIGHT_BOTTOM</td>
<td></td>
</tr>
</tbody>
</table>
+---------------------------------+
```

Inside alignments (FL_ALIGN\_INSIDE is set):

```
+---------------------------------+
<table>
<thead>
<tr>
<th>TOP_LEFT</th>
<th>TOP</th>
<th>TOP_RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>CENTER</td>
<td>RIGHT</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>BOTTOM_LEFT</td>
<td>BOTTOM</td>
<td>BOTTOM_RIGHT</td>
</tr>
</tbody>
</table>
+---------------------------------+
```

See also

FL_ALIGN\_CENTER, etc.

- typedef unsigned Fl_Align
  
  FLTK type for alignment control.

- const Fl_Align FL_ALIGN\_BOTTOM = (Fl_Align)2
  
  Align the label at the bottom of the widget.
• const Fl_Align FL_ALIGN_BOTTOM_LEFT = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT
  Align the label horizontally in the middle.
• const Fl_Align FL_ALIGN_BOTTOM_RIGHT = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_CENTER = (Fl_Align)0
  Align the label horizontally in the middle.
• const Fl_Align FL_ALIGN_CLIP = (Fl_Align)64
  All parts of the label that are larger than the widget will not be drawn.
• const Fl_Align FL_ALIGN_IMAGE_BACKDROP = (Fl_Align)0x0200
  If the label contains an image, draw the image or deimage in the background.
• const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320
• const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = (Fl_Align)0x0100
  If the label contains an image, draw the text to the right of the image.
• const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = (Fl_Align)0x0000
  If the label contains an image, draw the text below the image.
• const Fl_Align FL_ALIGN_INSIDE = (Fl_Align)16
  Draw the label inside of the widget.
• const Fl_Align FL_ALIGN_LEFT = (Fl_Align)4
  Align the label at the left of the widget.
• const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d
• const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007
• const Fl_Align FL_ALIGN_NOWRAP = (Fl_Align)0
• const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f
• const Fl_Align FL_ALIGN_RIGHT = (Fl_Align)8
  Align the label to the right of the widget.
• const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e
• const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b
• const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = (Fl_Align)0x0120
  If the label contains an image, draw the text to the left of the image.
• const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = (Fl_Align)0x0020
  If the label contains an image, draw the text on top of the image.
• const Fl_Align FL_ALIGN_TOP = (Fl_Align)1
  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 = (Fl_Align)128
  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
A font number is an index into the internal font table.

- **typedef int Fl_Fontsize**
  Size of a font in pixels.
- **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 FL_ITALIC = 2**
  add this to helvetica, courier, or times
- **FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE**
  normal font size
- **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.

### 10.3.1 Detailed Description

This file contains type definitions and general enumerations.

### 10.3.2 Macro Definition Documentation

#### 10.3.2.1 FL_ABI_VERSION

```c
#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>FLTK 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>

Generated by Doxygen
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

10.3.2.2 FL_API_VERSION

#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()

10.3.2.3 FL_MAJOR_VERSION

#define FL_MAJOR_VERSION 1
The major release version of this FLTK library.
See also

FL_VERSION

10.3.2.4 FL_MINOR_VERSION

#define FL_MINOR_VERSION 3
The minor release version for this library.
FLTK remains mostly source-code compatible between minor version changes.

10.3.2.5 FL_PATCH_VERSION

#define FL_PATCH_VERSION 9
The patch version for this library.
FLTK remains binary compatible between patches.

10.3.2.6 FL_VERSION

#define FL_VERSION
Value:

 ( (double)FL_MAJOR_VERSION + \n  (double)FL_MINOR_VERSION * 0.01 + \n  (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.
10.3 Enumerations.H File Reference

10.3.3 Typedef Documentation

10.3.3.1 Fl_Fontsize
typedef int Fl_Fontsize
Size of a font in pixels.
This is the approximate height of a font in pixels.

10.3.4 Enumeration Type Documentation

10.3.4.1 anonymous enum
anonymous enum
FD "when" conditions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_READ</td>
<td>Call the callback when there is data to be read.</td>
</tr>
<tr>
<td>FL_WRITE</td>
<td>Call the callback when data can be written without blocking.</td>
</tr>
<tr>
<td>FL_EXCEPT</td>
<td>Call the callback if an exception occurs on the file.</td>
</tr>
</tbody>
</table>

10.3.4.2 Fl_Boxtype
enum Fl_Boxtype

<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 1</td>
</tr>
<tr>
<td>FL_DOWN_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_UP_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_DOWN_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_THIN_UP_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_THIN_DOWN_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_THIN_UP_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_THIN_DOWN_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_ENGRAVED_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_EMBOSSED_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_EMBOSSED_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_BORDER_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_BORDER_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_SHADOW_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_SHADOW_FRAME</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_ROUNDED_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_ROUNDED_FRAME</td>
<td>see figure 1</td>
</tr>
</tbody>
</table>
10.3.4.3 Fl_Cursor

**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.

**Todo** enum Fl_Cursor needs maybe an image.
10.3Enumerations.H File Reference

10.3.4.4 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.</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>

10.3.4.5 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>
</tbody>
</table>
| 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 `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. |
| 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. |
| 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 handle() method. It then becomes the `Fl::belowmouse()` widget and will receive FL_MOVE and FL_LEAVE events. |
| FL_LEAVE             | 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.                                                                 |
| 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. |
| 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 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. |
<p>| FL_UNFOCUS           | This event is sent to the previous <code>Fl::focus()</code> widget when another widget gets the focus or the window loses focus.                                                                                         |</p>
<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 classed. The default callback closes the window calling Fl_Window::hide().</td>
</tr>
<tr>
<td>FL_MOVE</td>
<td>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.</td>
</tr>
<tr>
<td>FL_SHORTCUT</td>
<td>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 &quot;global&quot; shortcuts by using Fl::add_handler(). 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 Fl_Widget::deactivate() being called on it or one of its parents. Fl_Widget::active() may still be true after this, the widget is only active if Fl_Widget::active() is true on it and all its parents (use Fl_Widget::active_r() to check this).</td>
</tr>
<tr>
<td>FL_ACTIVATE</td>
<td>This widget is now active, due to Fl_Widget::activate() being called on it or one of its parents.</td>
</tr>
</tbody>
</table>
### FL_HIDE
This widget is no longer visible, due to `Fl_Widget::hide()` being called on it or one of its parents, or due to a parent window being minimized. `Fl_Widget::visible()` may still be true after this, but the widget is visible only if `visible()` is true for it and all its parents (use `Fl_Widget::visible_r()` to check this).

### FL_SHOW
This widget is visible again, due to `Fl_Widget::show()` being called on it or one of its parents, or due to a parent window being restored. Child `Fl_Widget` objects 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 `Fl_Widget::handle()` method!

### 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()`.

### 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.

### FL_MOUSEWHEEL
The user has moved the mouse wheel. The `Fl::event_dx()` and `Fl::event_dy()` methods can be used to find the amount to scroll horizontally and vertically.

### 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.

### 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.

### FL_DND_LEAVE
The mouse has moved out of the widget.

### 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.

### FL_SCREEN_CONFIGURATION_CHANGED
The screen configuration (number, positions) was changed. Use `Fl::add_handler()` to be notified of this event.

### FL_FULLSCREEN
The fullscreen state of the window has changed.

### FL_ZOOM_GESTURE
The user has made a zoom/pinch/magnification gesture. The `Fl::event_dy()` method can be used to find magnification amount, `Fl::event_x()` and `Fl::event_y()` are set as well.

### 10.3.4.6 Fl_Labeltype

The `labeltype()` method sets the type of the label.

The following standard label types are included:

**Todo** The doxygen comments are incomplete, and some labeltypes start with an underscore. Also, there are three external functions undocumented (yet):

- `fl_define_FL_SHADOW_LABEL()
- `fl_define_FL_ENGRAVED_LABEL()
- `fl_define_FL_EMBOSSED_LABEL()`
10.3 Enumerations.H File Reference

### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</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 though the text is engraved</td>
</tr>
<tr>
<td>_FL_EMBOSSED_LABEL</td>
<td>draws edges as though 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 labelfype to use for creating own labeltypes</td>
</tr>
</tbody>
</table>

### 10.3.4.7 Fl_When

**enum Fl_When**

These constants determine when a callback is performed.

See also

Fl_Widget::when();

**Todo** doxygen comments for values are incomplete and maybe wrong or unclear

### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</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>?</td>
</tr>
</tbody>
</table>

### 10.3.5 Function Documentation

#### 10.3.5.1 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.
10.3.5.2 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,  
and 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);      

10.3.5.3 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.

10.3.5.4 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.

10.3.5.5 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)

10.3.6 Variable Documentation

10.3.6.1 FL_ALIGN_LEFT

const Fl_Align FL_ALIGN_LEFT = (Fl_Align)4

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.

10.3.6.2 FL_ALIGN_TOP

const Fl_Align FL_ALIGN_TOP = (Fl_Align)1

Align the label at the top of the widget.

Inside labels appear below the top, outside labels are drawn on top of the widget.

10.3.6.3 FL_NORMAL_SIZE

FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE [extern]

normal font size

normal font size
# 10.4 Enumerations.H

Go to the documentation of this file.

---

10.4 Enumerations.H

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

---

```c
#ifndef Fl_Enumerations_H
#define Fl_Enumerations_H

#include <FL/abi-version.h>

#include "Fl_Export.H"
#include "fl_types.h"

#define FL_MAJOR_VERSION 1
#define FL_MINOR_VERSION 3
#define FL_PATCH_VERSION 9

#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

/*
 * Notes on FL_ABI_VERSION and deprecated (obsolete) FLTK_ABI_VERSION:
 * (1) FLTK_ABI_VERSION is deprecated, but still defined below.
 * Do NOT define FLTK_ABI_VERSION here - it would be overwritten later.
 * (2) FL_ABI_VERSION is now (as of FLTK 1.3.4) defined by configure
 * or CMake. Do NOT define it here. Its definition will be included
 * below by "#include <FL/abi-version.h>".
 * (3) If you use the provided IDE files (Windows VC++ or Xcode) you should
 * edit the definition in the provided file abi-version.ide. The correct
 * file is '/path/to/fltk/abi-version.ide'.
 *
 * For more informations on FL_ABI_VERSION see README.abi-version.txt.
 *
 * For more informations on FL_ABI_VERSION see README.abi-version.txt.
 */
```

---

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.

---

Generated by Doxygen
# undef FL_ABI_VERSION
#define FL_ABI_VERSION FL_API_VERSION
#endif

/*
FLTK_ABI_VERSION is deprecated (replaced by FL_ABI_VERSION).
This deprecated constant will be removed in FLTK 1.4.0 and later.
Please use FL_ABI_VERSION when FLTK 1.4.0 has been released.
*/

#ifdef FLTK_ABI_VERSION
#undef FLTK_ABI_VERSION
#endif
#define FLTK_ABI_VERSION FL_ABI_VERSION

#ifdef FL_ABI_VERSION
#endif

// group: Version Numbers
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_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
};

enum Fl_When { // Fl_Widget::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
};

/*
// DEV NOTE: Keep this list in sync with FL/names.H
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_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
};

enum Fl_When { // Fl_Widget::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
};

// group: When Conditions
*/
00449
00462 // FIXME: These codes collide with valid Unicode keys
00444
00465 #define FL_Button 0xffe8
00466 #define FL_BackSpace 0xff08
00467 #define FL_Tab 0xff09
00468 #define FL_ISO_Key 0xff0c
00469 #define FL_Escape 0xff1b
00470 #define FL_Kana 0xff20
00471 #define FL_Eisu 0xff21
00472 #define FL_Yen 0xff30
00473 #define FL_JIS_Underscore 0xff31
00474 #define FL_Home 0xff50
00475 #define FL_Left 0xff51
00476 #define FL_Up 0xff52
00477 #define FL_Right 0xff53
00478 #define FL_Down 0xff54
00479 #define FL_Page_Up 0xff55
00480 #define FL_Page_Down 0xff56
00481 #define FL_End 0xff57
00482 #define FL_Print 0xff61
00483 #define FL_Insert 0xff63
00484 #define FL_Menu 0xff67
00485 // These use the Private Use Area (PUA) of the Basic Multilingual Plane
00486 // of Unicode. Guaranteed not to conflict with a proper Unicode character.
00488
00489 #define FL_Volume_Down 0xEF11 / * Volume control down */
00490 #define FL_Volume_Mute 0xEF12 / * Mute sound from the system */
00491 #define FL_Volume_Up 0xEF13 / * Volume control up */
00492 // group: Mouse and Keyboard Events
00493 #define FL_LEFT_MOUSE 1
00494 #define FL_MIDDLE_MOUSE 2
00495 #define FL_RIGHT_MOUSE 3
00496 // group: Mouse Buttons
00497 // FIXME: it would be nice to have the modifiers in the upper 8 bit so that
00498 // a unicode ke (24bit) can be sent as an unsigned with the modifiers.
00499 #define FL_SHIFT 0x00010000
00500 #define FL_CAPS_LOCK 0x00020000
00501 #define FL_CTRL 0x00040000
00502 #define FL_ALT 0x00080000
00503 #define FL_NUM_LOCK 0x00100000
00504 #define FL_META 0x00400000
00505 // correct for XFree86
00506 #define FL_SCROLL_LOCK 0x00800000
00507
00508 // group: Event States
00509 // These primarily map to the XFree86 keysym range
00510 #define FL_Volume_Down 0xFF11 / * Volume control down */
00511 #define FL_Volume_Mute 0xFF12 / * Mute sound from the system */
00512 #define FL_Volume_Up 0xFF13 / * Volume control up */
00513 #define FL_Media_Play 0xFF14 / * Start playing of audio */
00514 #define FL_Media_Stop 0xFF15 / * Stop playing audio */
00515 #define FL_Media_FastForward 0xFF16 / * Previous track */
00516 #define FL_Media_Next 0xFF17 / * Next track */
00517 #define FL_Home_Page 0xFF18 / * Display user's homepage */
00518 #define FL_Email 0xFF19 / * Invoke user's email program */
00519 #define FL_Search 0xFF1B / * Search */
00520 #define FL_Back 0xFF26 / * Like back on a browser */
00521 #define FL_Favorites 0xFF30 / * Show favorite locations */
00522 #define FL_Sleep 0xFF3F / * Put system to sleep */
00523 // most X servers do this?
00524 // group: Mouse Buttons
00525 //FIXME: it would be nice to have the modifiers in the upper 8 bit so that
00526 // a unicode ke (24bit) can be sent as an unsigned with the modifiers.
00527 // group: Event States
00528 //FIXME: it would be nice to have the modifiers in the upper 8 bit so that
00529 // a unicode ke (24bit) can be sent as an unsigned with the modifiers.
// correct for XFree86
#define FL_BUTTON1 0x01000000
#define FL_BUTTON2 0x02000000
#define FL_BUTTON3 0x04000000
#define FL_BUTTONS 0x7f000000
#define FL_BUTTON(n) (0x00800000«(n))

#define FL_KEY_MASK 0x0000ffff

// FIXME: Unicode needs 24 bits!

#ifdef __APPLE__
#define FL_COMMAND FL_META
#define FL_CONTROL FL_CTRL
#else
#define FL_COMMAND FL_CTRL
#define FL_CONTROL FL_META
#endif // __APPLE__

// group: Event States

enum Fl_Boxtype { // boxtypes (if you change these you must fix 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_SHADOw_BOX,
FL_OSHADOW_BOX,
FL_OVAL_BOX,
FL_OVAL_FRAME,
FL_OFLAT_BOX,
FL_OFLAT_FRAME,
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_ROUND_UP_BOX,
FL_GTK_ROUND_DOWN_BOX,
FL_GTK_THIN_UP_BOX,
FL_GTK_THIN_DOWN_BOX,
FL_GTK_THIN_UP_FRAME,
FL_GTK_THIN_DOWN_FRAME,
FL_GTK_ROUND_UP_FRAME,
FL_GTK_ROUND_DOWN_FRAME,
FL_GTK_ROUND_UP_BOX,
FL_GTK_ROUND_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_GTK_UP_BOX,
FL_GTK_DOWN_BOX,
FL_GTK_THIN_UP_BOX,
FL_GTK_THIN_DOWN_BOX,
FL_GTK_THIN_DOWN_FRAME,
FL_GTK_THIN_UP_FRAME,
FL_GTK_ROUND_UP_BOX,
FL_GTK_ROUND_DOWN_BOX,
FL_GTK_ROUND_UP_FRAME,
FL_GTK_THIN_UP_BOX,
FL_GTK_THIN_DOWN_BOX,
FL_GTK_THIN_DOWN_FRAME,
FL_GTK_THIN_UP_FRAME,
FL_GTK_ROUND_UP_BOX,
FL_GTK_ROUND_DOWN_BOX,
};

extern FL_EXPORT Fl_Boxtype fl_define_FL_ROUND_UP_BOX();
#define FL_ROUND_UP_BOX fl_define_FL_ROUND_UP_BOX()
#define FL_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_ROUND_UP_BOX()+1)
extern FL_EXPORT Fl_Boxtype fl_define_FL_SHADOW_BOX();
#define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()
#define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)
extern FL_EXPORT Fl_Boxtype fl_define_FL_ROUNDED_BOX();
#define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()
#define FL_ROUNDED_FRAME (Fl_Boxtype)(fl_define_FL_ROUNDED_BOX()+2)
extern FL_EXPORT Fl_Boxtype fl_define_FL_RFLAT_BOX();
#define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()
extern FL_EXPORT Fl_Boxtype fl_define_FL_RSHADOW_BOX();
#define FL_RSHADOW_BOX (Fl_Boxtype)(fl_define_FL_RSHADOW_BOX()+1)
#define FL_DIAMOND_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+2)
#define FL_DIAMOND_UP_BOX fl_define_FL_DIAMOND_BOX()
#define FL_DIAMOND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)
#define FL_OVAL_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+1)
#define FL_OVAL_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)
#define FL_OFLAT_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+3)
#define FL_OVAL_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)
extern FL_EXPORT Fl_Boxtype fl_define_FL_PLASTIC_UP_BOX();
#define FL_PLASTIC_UP_BOX fl_define_FL_PLASTIC_UP_BOX()
#define FL_PLASTIC_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+1)
#define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+2)
#define FL_PLASTIC_UP_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_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
#define FL_PLASTIC_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)
#define FL_PLASTIC_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+8)
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_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+2)
#define FL_GTK_UP_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_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+6)
#define FL_GTK_THIN_UP_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_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
#define FL_GLEAM_UP_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_THIN_DOWN_FRAME (Fl_Boxtype)(fl_DEFINE_FL_GLEAM_UP_BOX()+6)
#define FL_GLEAM_THIN_UP_FRAME (Fl_Boxtype)(fl_DEFINE_FL_GLEAM_UP_BOX()+7)
#define FL_GLEAM_ROUND_UP_BOX (Fl_Boxtype)(fl_DEFINE_FL_GLEAM_UP_BOX()+8)
#define FL_GLEAM_ROUND_DOWN_BOX (Fl_Boxtype)(fl_DEFINE_FL_GLEAM_UP_BOX()+9)
// 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
};
#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
#define FL_SHADOW_LABEL FL_SHADOW_LABEL
#define FL_EMBOSSED_LABEL FL_EMBOSSED_LABEL
#define FL_ICON_LABEL FL_ICON_LABEL
#define FL_IMAGE_LABEL FL_IMAGE_LABEL
#define FL_FREE_LABELTYPE
FL_ALIGN_CENTER = (Fl_Align)0;
FL_ALIGN_TOP = (Fl_Align)1;
const Fl_Align FL_ALIGN_BOTTOM = (Fl_Align)2;
const Fl_Align FL_ALIGN_LEFT = (Fl_Align)4;
const Fl_Align FL_ALIGN_RIGHT = (Fl_Align)8;
const Fl_Align FL_ALIGN_INSIDE = (Fl_Align)16;
const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = (Fl_Align)0x0020;
const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = (Fl_Align)0x0000;
const Fl_Align FL_ALIGN_CLIP = (Fl_Align)64;
const Fl_Align FL_ALIGN_WRAP = (Fl_Align)128;
const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = (Fl_Align)0x0100;
const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = (Fl_Align)0x0120;
const Fl_Align FL_ALIGN_IMAGE_BACKDROP = (Fl_Align)0x0200;
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_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_LEFT_TOP = 0x0007; // magic value
const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b; // magic value
const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d; // magic value
const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e; // magic value
const Fl_Align FL_ALIGN_NOWRAP = (Fl_Align)0; // for back compatibility
const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f; // left, right, top, bottom
const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320; // l/r, t/b, backdrop

/*
 * Font selection
 *
 */

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 = 15;
const Fl_Font FL_BOLD = 1;
const Fl_Font FL_ITALIC = 2;
const Fl_Font FL_BOLD_ITALIC = 3;

/*
 * Font sizes
 *
 */

typedef 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_BACKGROUND_COLOR = 7;
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;  // 'I'
const Fl_Color FL_DARK1 = 47;  // 'K'
const Fl_Color FL_BACKGROUND_COLOR = 49;  // 'K' 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);
FL_EXPORT Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg);
FL_EXPORT Fl_Color fl_color_average(Fl_Color c1, Fl_Color c2, float weight);

inline Fl_Color fl_lighter(Fl_Color c) { return fl_color_average(c, FL_WHITE, .67f); }  
inline Fl_Color fl_darker(Fl_Color c) { return fl_color_average(c, FL_BLACK, .67f); }  
inline Fl_Color fl_rgb_color(uchar r, uchar g, uchar b) {
    if (!r && !g && !b) return FL_BLACK;
    else return (Fl_Color)(((((r << 8) | g) << 8) | b) << 8);
}
inline Fl_Color fl_rgb_color(uchar g) {
    if (!g) return FL_BLACK;
    else return (Fl_Color)(((((g << 8) | g) << 8) | g) << 8);
}
inline Fl_Color fl_gray_ramp(int i) {return (Fl_Color)(i+FL_GRAY_RAMP);}  
inline Fl_Color fl_color_cube(int r, int g, int b) {
    return (Fl_Color)((b*FL_NUM_RED + r) * FL_NUM_GREEN + g + FL_COLOR_CUBE);} 

// group: Colors

/* FIXME: We should renumber these, but that will break the ABI */
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_NW = 68,
    FL_CURSOR_WE = 79,
    FL_CURSOR_NSEW = 80,
    FL_CURSOR_NWSE = 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
}; // group: Cursors

enum {
    FL_READ = 1,
    FL_WRITE = 4,
    FL_EXCEPT = 8
};

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
};

Generated by Doxygen
// image alpha blending
#define FL_IMAGE_WITH_ALPHA 0x40000000

enum Fl_Damage {
  FL_DAMAGE_CHILD = 0x01,
  FL_DAMAGE_EXPOSE = 0x04,
  FL_DAMAGE_SCROLL = 0x08,
  FL_DAMAGE_OVERLAY = 0x10,
  FL_DAMAGE_USER1 = 0x20,
  FL_DAMAGE_USER2 = 0x40,
  FL_DAMAGE_ALL = 0x80
};

// FLTK 1.0.x compatibility definitions...
#ifdef FLTK_1_0_COMPAT
  #define contrast fl_contrast
  #define down fl_down
  #define frame fl_frame
  #define inactive fl_inactive
#endif // FLTK_1_0_COMPAT

10.5 filename.H File Reference

File names and URI utility functions.
#include "Fl_Export.H"
#include <sys/types.h>
#include <dirent.h>

Macros
• #define fl_dirent_h_cyclic_include
• #define FL_FILENAME_H
• #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
• FL_EXPORT void fl_decode_uri (char *uri)
  Decodes a URL-encoded string.
• FL_EXPORT int fl_filename_absolute (char *to, int tolen, const char *from)
  Makes a filename absolute from a relative filename.
• FL_EXPORT int fl_filename_expand (char *to, int tolen, const char *from)
  Expands a filename containing shell variables and tilde (~).
• FL_EXPORT const char * fl_filename_ext (const char *buf)
  Gets the extensions of a filename.
• FL_EXPORT void fl_filename_free_list (struct dirent ***, int n)
  Free the list of filenames that is generated by fl_filename_list().
• FL_EXPORT int fl_filename_isdir (const char *name)
  Determines if a file exists and is a directory from its filename.
• FL_EXPORT 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.

- **FL_EXPORT int fl_filename_match (const char *name, const char *pattern)**
  Checks if a string s matches a pattern p.

- **FL_EXPORT const char * fl_filename_name (const char *filename)**
  Gets the file name from a path.

- **FL_EXPORT int fl_filename_relative (char *to, int tolen, const char *from)**
  Makes a filename relative to the current working directory.

- **FL_EXPORT char * fl_filename_setext (char *to, int tolen, const char *ext)**
  Replaces the extension in buf of max.

- **FL_EXPORT int fl_open_uri (const char *uri, char *msg, int msglen)**
  Opens the specified Uniform Resource Identifier (URI).

### 10.5.1 Detailed Description

File names and URI utility functions.

### 10.6 filename.H

Go to the documentation of this file.

```c
#ifndef FL_FILENAME_H
#define FL_FILENAME_H

#include "Fl_Export.H"

#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);

// Under WIN32, we include filename.h from numericsort.c; this should probably change...

```

Generated by Doxygen
```c
# if defined(WIN32) && !defined(__MINGW32__) && !defined(__CYGWIN__) && !defined(__WATCOMC__)

struct dirent {char d_name[1];};

# elif defined(__WATCOMC__)
# include <sys/types.h>
# include <direct.h>
#
# else
  /* WARNING: on some systems (very few nowadays?) <dirent.h> may not exist.
   * The correct information is in one of these files:
   * 
   * #include <sys/ndir.h>
   * #include <sys/dir.h>
   * #include <ndir.h>
   * 
   * plus you must do the following #define:
   * 
   * #define dirent direct
   * 
   * It would be best to create a <dirent.h> file that does this...
   */
# include <sys/types.h>
#define fl_dirent_h_cyclic_include
#include <dirent.h>
#undef fl_dirent_h_cyclic_include
# endif

# if defined (__cplusplus)
extern "C" {
#endif /* __cplusplus */

# if !defined(FL_DOXYGEN)
FL_EXPORT int fl_alphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casealphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casenumeric sort (struct dirent **, struct dirent **);
FL_EXPORT int fl_numeric sort ( struct dirent **, struct dirent **);
#endif

#if defined(__cplusplus)
}
#endif /* __cplusplus */

typedef int (Fl_File_Sort_F)(struct dirent **, struct dirent **);

/* Portable "scandir" function. Ugly but necessary... */

/* Generic function to open a Uniform Resource Identifier (URI) using a
system-defined program (added in FLTK 1.1.8) */

/* _fl_filename_isdir_quick() is a private function that checks for a
trailing slash and assumes that the passed name is a directory if
it finds one. This function is used by Fl_File_Browser and
Fl_File_Chooser to avoid extra stat() calls, but is not supported
outside of FLTK... */

int _fl_filename_isdir_quick(const char *name);

/* FLTK 1.0.x compatibility definitions... */
define filename_absolute fl_filename_absolute
#define filename_expand fl_filename_expand
#define filename_ext fl_filename_ext
#define filename_isdir fl_filename_isdir
#define filename_list fl_filename_list
#define filename_match fl_filename_match
#define filename_name fl_filename_name
```

10.7 FL.H File Reference

**FL static class.**

```c
#include <FL/FL_Export.H>
#include <FL/FL_Cairo.H>
#include "fl_utf8.h"
#include "Enumerations.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!

- `#define FL_SOCKET int`

**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 label drawing functions passed as parameters.
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 some timeout callback functions passed as parameters.

Variables
• FL_EXPORT const char * fl_local_alt
  string pointer used in shortcuts, you can change it to another language
• FL_EXPORT const char * fl_local_ctrl
  string pointer used in shortcuts, you can change it to another language
• FL_EXPORT const char * fl_local_meta
  string pointer used in shortcuts, you can change it to another language
• FL_EXPORT const char * fl_local_shift
  string pointer used in shortcuts, you can change it to another language

10.7.1 Detailed Description

Fl static class.

10.8 Fl.H

Go to the documentation of this file.
# Fl.H 1627

```cpp
// Keep avoiding having the socket deps at that level but make sure it will work in both 32 & 64 bit builds
#if defined(WIN32) && !defined(_CYGWIN_)
#else
#endif

// Pointers you can use to change FLTK to a foreign language.
// Note: Similar pointers are defined in FL/fl_ask.H and src/fl_ask.cxx
extern FL_EXPORT const char* fl_local_ctrl;
extern FL_EXPORT const char* fl_local_meta;
extern FL_EXPORT const char* fl_local_alt;
extern FL_EXPORT const char* fl_local_shift;

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 {
    Fl() {;} // no constructor!
private:
    static int use_high_res_GL;
public: // should be private!
#ifdef FL_DOXYGEN
    static int e_number;
    static int e_x;
    static int e_y;
    static int e_root_x;
    static int e_root_y;
    static int e_dX;
    static int e_dY;
    static int e_state;
    static int e_clicks;
    static int e_is_click;
    static int e_keysym;
    static char* e_text;
    static int e_length;
    static void* e_clipboard_data;
    static const char* e_clipboard_type;
    static Fl_Event_Dispatch* e_dispatch;
    static Fl_Widget* belowmouse_;
    static Fl_Widget* pushed_;
static Fl_Widget* focus_;  
static int damage_;  
static Fl_Widget* selection_owner_;  
static Fl_Window* modal_;  
static Fl_Widget* grab_;  
static int compose_state; // used for dead keys (WIN32) or marked text (MacOS)  
static void call_screen_init(); // recompute screen number and dimensions  
#ifdef __APPLE__  
  static void reset_marked_text(); // resets marked text  
  static void insertion_point_location(int x, int y, int height); // sets window coordinates & height of insertion point  
#endif  
#endif // FL_DOXYGEN  

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.  
    // typeofme  
    // 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.  
    // typeofme  
    OPTION_FILECHOOSER_PREVIEW,  
    OPTION_VISIBLE_FOCUS,  
    OPTION_DND_TEXT,  
    OPTION_SHOW_TOOLTIP,  
    OPTION_FNFC_USES_GTK,  
    // don't change this, leave it always as the last element  
    OPTION_LAST  
  } Fl_Option;

private:
  static unsigned char options_[OPTION_LAST];  
  static unsigned char options_read_;  
  public:
  /**<  
  Return a global setting for all FLTK applications, possibly overridden  
  by a setting specifically for this application.  
  */  
  static bool option(Fl_Option opt);
  /**<  
  Override an option while the application is running.  
  */  
  static void option(Fl_Option opt, bool val);

  static void ( *idle)();
  #ifndef FL_DOXYGEN  
  static Fl_Awake_Handler* awake_ring_;  
  static void** awake_data_;  
  static int awake_ring_size_;  
  static int awake_ring_head_;  
  static int awake_ring_tail_;  
  static const char* scheme_;  
  static Fl_Image* scheme_bg_;  
  #endif  

  static int e_original_keysym; // late addition  
  static int scrollbar_size_;  
  #ifndef FL_DOXYGEN
  static Fl_Awake_Handler* awake_ring_;  
  static void** awake_data_;  
  #endif  

  static int add_awake_handler_(Fl_Awake_Handler, void*);  
  static int get_awake_handler_(Fl_Awake_Handler&, void*&);
  public:
  /**<  
  API version number  
  */  
  static double version();  
  static int api_version();  
  /**<  
  ABI version number  
  */  
  static int abi_version();  
  static inline int abi_check(const int val = FL_ABI_VERSION) {  
    return val == abi_version();  
  }
  /**<  
  // argument parsers:
  */
  #ifdef FL_DOXYGEN
static int arg(int argc, char **argv, int &i);
static int argc(char **argv, int i, Fl_Args_Handler cb = 0);
static void args(int argc, char **argv);

static int fl_visual(int, int *alist=0); // platform dependent
static int own_colormap();
static void foreground(uchar, uchar, uchar);
static void background(uchar, uchar, uchar);
static void background2(uchar, uchar, uchar);

static const char * const help;

// things called by initialization:
static void display(const char *);
static int visual(int);
static int gl_visual(int, int *alist=0); // platform dependent
static void own_colormap();
static void get_system_colors();
static void foreground(uchar, uchar, uchar);
static void background(uchar, uchar, uchar);

// schemes:
static int is_scheme(const char *name) {
  return (scheme_ && name && !strcmp(name, scheme_));
}
static int reload_scheme(); // platform dependent
static int scrollbar_size();
static void scrollbar_size(int W);
static void foreground(uchar, uchar, uchar);
static void background(uchar, uchar, uchar);

// execution:
static int wait();
static double wait(double time);
static int check();
static int ready();
static int run();

static Fl_Widget * readqueue();
static void add_timeout(double t, Fl_Timeout_Handler,void * = 0); // platform dependent
static void repeat_timeout(double t, Fl_Timeout_Handler, void * = 0); // platform dependent
static int has_timeout(Fl_Timeout_Handler, void * = 0);
static void remove_timeout(Fl_Timeout_Handler, void * = 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 void add_fd(int fd, int when, Fl_FD_Handler cb, void * = 0); // platform dependent
static void add_fd(int fd, Fl_FD_Handler cb, void * = 0); // platform dependent
static void remove_fd(int, int when); // platform dependent
static void remove_fd(int); // platform dependent

static int damage() {return damage_;
static void redraw();
static void flush();

static Fl_Window * first_window();
static void first_window(Fl_Window *);
static Fl_Window * next_window(const Fl_Window*);

static Fl_Window * modal() {return modal_;
static Fl_Window * grab();
static Fl_Window * modal(Fl_Window *);
static Fl_Window * grab(Fl_Window *);

static Fl_Window * modal() { return modal_; }
static Fl_Window* grab() { return grab_; }
static void grab(Fl_Window*); // platform dependent

// event information:
static int event() {return e_number;}
static int event_x() {return e_x;}
static int event_y() {return e_y;}

static Fl_Window * modal() { return modal_; }
static Fl_Window* grab() { return grab_; }
static void grab(Fl_Window*); // platform dependent

// 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_dx() { return e_dx;}
static int event_dy() { return e_dy;}

static int event_x_root() { return e_x_root;}
static int event_y_root() { return e_y_root;}

static int event_is_click() { return e_is_click;}
static int event_button() { return e_keysym-FL_Button;}
static int event_state() { return e_state;}

static void get_mouse(int &,int &); // platform dependent
static void get_clicks(int i) { e_clicks = i; }
static void get_is_click() { return e_is_click;}
static void get_event_button() { return e_keysym-FL_Button;}

Generated by Doxygen
static int compose(int &del);
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();

// cut/paste:
#if FLTK_ABI_VERSION >= 10303 || defined(FL_DOXYGEN)
static void copy(const char * stuff, int len, int destination = 0, const char *type = Fl::clipboard_plain_text); // platform dependent
#else
static void copy(const char * stuff, int len, int destination, const char *type);
static void copy(const char * stuff, int len, int destination = 0);
#endif

#if !(defined(__APPLE__) || defined(WIN32) || defined(FL_DOXYGEN))
static void copy_image(const unsigned char * data, int W, int H, int destination = 0); // platform dependent
#endif

#if FLTK_ABI_VERSION >= 10303 || defined(FL_DOXYGEN)
static void paste(Fl_Widget &receiver, int source, const char *type = Fl::clipboard_plain_text); // platform dependent
#else
static void paste(Fl_Widget &receiver, int source, const char *type);
static void paste(Fl_Widget &receiver, int source / *=0*/);
#endif

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

// These are for back-compatibility only:
static Fl_Widget* selection_owner() {return selection_owner_};
static void selection_owner(Fl_Widget*);
static void selection(Fl_Widget &owner, const char *, int len);
static void paste(Fl_Widget &receiver);

// screen size:
static int x(); // platform dependent
static int y(); // platform dependent
static int w(); // platform dependent
static int h(); // platform dependent

// multi-head support:
static int screen_count();
static int screen_x(int X, int Y, int W, int H) {
    int x, y;
    Fl::get_mouse(x, y);
    screen_xwh(X, Y, W, H, x, y);
}
static int screen_y(int X, int Y, int W, int H) {
    int x, y;
    Fl::get_mouse(x, y);
    screen_ywh(X, Y, W, H, x, y);
}
static int screen_w(int X, int Y, int W, int H) {
    int x, y;
    Fl::get_mouse(x, y);
    screen_whwh(X, Y, W, H, x, y);
}
static int screen_size() {
    int x, y;
    Fl::get_mouse(x, y);
    return screen_whwh(x, y, x, y);
}
static int screen_work_area(int X, int Y, int W, int H) {
    int x, y;
    Fl::get_mouse(x, y);
    screen_work_area(X, Y, W, H, x, y, x, y);
}
static int screen_work_area(int X, int Y, int W, int H) {
    int x, y;
    Fl::get_mouse(x, y);
    screen_work_area(X, Y, W, H, x, y, x, y);
}

// color map:
static void set_color(Fl_Color, uchar, uchar, uchar);
static void set_color(Fl_Color i, unsigned c); // platform dependent
static unsigned get_color(Fl_Color i);
static void get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue);
static void free_color(Fl_Color i, int overlay = 0); // platform dependent

// fonts:
static const char * get_font(Fl_Font);
static const char * get_font_name(Fl_Font, int* attributes = 0);
static int get_font_sizes(Fl_Font, int* & sizep);
static void set_font(Fl_Font, const char *);
static void set_font(Fl_Font, Fl_Font);
static Fl_Font set_fonts(const char * = 0); // platform dependent

// labeltypes:
static void set_labeltype(Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *);
static void set_labeltype(Fl_Labeltype, Fl_Labeltype from); // is it defined ?

// boxtypes:
static Fl_Box_Draw_F * get_boxtype(Fl_Boxtype);
static void set_boxtype(Fl_Boxtype, Fl_Boxtype from);
static int box_dx(Fl_Boxtype);
static int box_dy(Fl_Boxtype);
static int box_dw(Fl_Boxtype);
static int box_dh(Fl_Boxtype);
static int draw_box_active();
static Fl_Color box_color(Fl_Color);
static void set_box_color(Fl_Color);

// back compatibility:
static void set_abort(Fl_Abort_Handler f) { fatal = f; } 
static void (*atclose)(Fl_Window*, void*); 
static void default_atclose(Fl_Window *, void*);
static void set_atclose(Fl_Atclose_Handler f) { atclose = f; }
static int event_shift() { return e_state & FL_SHIFT; }
static int event_ctrl() { return e_state & FL_CTRL; }
static int event_command() { return e_state & FL_COMMAND; }
static int event_alt() { return e_state & FL_ALT; }
static int event_buttons() { return e_state & 0x7f000000; }
static int event_button1() { return e_state & FL_BUTTON1; }
static int event_button2() { return e_state & FL_BUTTON2; }
static int event_button3() { return e_state & FL_BUTTON3; }
static int lock();
static void unlock();
static void awake(void * message = 0);
static int awake(Fl_Awake_Handler cb, void * message = 0);
static void * thread_message(); // platform dependent
static void delete_widget(Fl_Widget *w);
static void do_widget_deletion();
static void watch_widget_pointer(Fl_Widget &, void*);
static void release_widget_pointer(Fl_Widget &);
static void clear_widget_pointer(Fl_Widget const *);
static void use_high_res_GL(int val) { use_high_res_GL_ = val; }
static int use_high_res_GL() { return use_high_res_GL_; }

#include "FLTK_HAVE_CAIRO"

public:
#include "Cairo support API"
static Cairo_t * cairo_make_current(Fl_Window *, int);
static bool cairo_antialias_context(bool alink) [cairo_state_.antialias(alink)];
static Cairo_t * cairo_antialias_context() [return cairo_state_.antialias();]
static Cairo_t * cairo_complex(int c, bool own=false) [cairo_state_.complex(c, own)];
static Cairo_t * cairo_complex(void* gc);
static Cairo_t * cairo_make_current(void* gc, int W, int H);
static Fl_CairoState cairo_state_;

private:
#include "FLTK_HAVE_CAIRO"
01336 
01337 class FL_EXPORT Fl_Widget_Tracker {
01338     Fl_Widget* wp_;  
01339 
01340 public:
01341     Fl_Widget_Tracker(Fl_Widget *wi); 
01342     ~Fl_Widget_Tracker();
01343     Fl_Widget* widget() {return wp_;}
01344     int deleted() {return wp_ == 0;}
01345     int exists() {return wp_ != 0;}
01346 };
01347 #endif // !Fl_H

10.9 Fl_Adjuster.H

00001 //
00002 // "$Id$"
00003 //
00004 // Adjuster widget header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2010 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 /*
00020 Fl_Adjuster widget. */
00021 // 3-button "slider", made for Nuke
00022 //
00023 ifndef Fl_Adjuster_H
00024 #define Fl_Adjuster_H
00025 #ifndef Fl_Valuator_H
00026 
00027 #endif // Fl_Adjuster_H
00028 
00029 #endif
00030 #endif // Fl_Adjuster_H
00031
class FL_EXPORT Fl_Adjuster : public Fl_Valuator {
00032     int drag;
00033     int ix;
00034     int soft_;  
00035     protected:
00036     void draw();
00037     int handle(int);
00038     void value_damage();
00039 public:
00040     Fl_Adjuster(int X, int Y, int W, int H, const char *l=0);
00041     void soft(int s) {soft_ = s;}
00042     int soft() const {return soft_;}
00043     ~Fl_Adjuster();
00044 
00048     void #endif // "$Id$"

10.10 Fl_ask.H File Reference

API for common dialogs.
#include "Enumerations.H"
Macros

- #define __fl_attr(x)

Enumerations

- enum Fl_Beep {
  FL_BEEP_DEFAULT = 0, FL_BEEP_MESSAGE, FL_BEEP_ERROR, FL_BEEP_QUESTION,
  FL_BEEP_PASSWORD, FL_BEEP_NOTIFICATION
}

Different system beeps available.

Functions

- FL_EXPORT void FL_EXPORT void fl_alert(const char *,...) __fl_attr((__format__(__printf__.)

- FL_EXPORT void FL_EXPORT void FL_EXPORT int fl_ask(const char *,...) __fl_attr((__format__(__printf__.)

- FL_EXPORT void fl_beep (int type=FL_BEEP_DEFAULT)

  Emits a system beep message.

- FL_EXPORT int fl_choice(const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__.)

- FL_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT int fl_choice_n(const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__.)

- FL_EXPORT int FL_EXPORT const char * fl_input(const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__.)

- FL_EXPORT void fl_message (const char *,...) __fl_attr((__format__(__printf__.)

- void fl_message_font (Fl_Font f, Fl_Fontsize s)

- FL_EXPORT void fl_message_hotspot (int enable)

  Sets whether or not to move the common 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_EXPORT int fl_message_hotspot (void)

  Gets whether or not to move the common 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_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT int FL_EXPORT 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() to follow the mouse pointer.

- FL_EXPORT void fl_message_title (const char *title)

  Sets the title of the dialog window used in many common dialogs.

- FL_EXPORT void fl_message_title_default (const char *title)

  Sets the default title of the dialog window used in many common dialogs.

- FL_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT int FL_EXPORT Fl_Widget * fl_password (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__.)

Variables

- FL_EXPORT void FL_EXPORT void FL_EXPORT int __deprecated__

- FL_EXPORT const char * fl_cancel

  String pointer used in common dialogs, you can change it to another language

- FL_EXPORT const char * fl_close

  String pointer used in common dialogs, you can change it to another language

- FL_EXPORT Fl_Font fl_message_font__

- FL_EXPORT Fl_Fontsize fl_message_size__

- FL_EXPORT const char * fl_no

  String pointer used in common dialogs, you can change it to another language

- FL_EXPORT const char * fl_ok
• FL_EXPORT const char * fl_yes

string pointer used in common dialogs, you can change it to another language

10.10.1 Detailed Description

API for common dialogs.

10.10.2 Enumeration Type Documentation

10.10.2.1 Fl_Beep

```cpp
enum Fl_Beep
{
    FL_BEEP_DEFAULT = 0,
    FL_BEEP_MESSAGE,
    FL_BEEP_ERROR,
    FL_BEEP_QUESTION,
    FL_BEEP_PASSWORD,
    FL_BEEP_NOTIFICATION
};
```

Different system beeps available.

See also

```cpp
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>

10.11 fl_ask.H

Go to the documentation of this file.
FL_EXPORT void fl_beep(int type = FL_BEEP_DEFAULT);
FL_EXPORT void fl_message(const char *,...) __fl_attr((__format__ (__printf__, 1, 2)));
FL_EXPORT void fl_alert(const char *,...) __fl_attr((__format__ (__printf__, 1, 2)));

// 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() instead with the appropriate verbs instead.
FL_EXPORT int fl_choice(const char *q, const char *b0, const char *b1, const char *b2, ...)
  __fl_attr((__format__ (__printf__, 1, 5)));
FL_EXPORT const char *fl_input(const char *label, const char *deflt = 0, ...)
  __fl_attr((__format__ (__printf__, 1, 3)));
FL_EXPORT const char *fl_password(const char *label, const char *deflt = 0, ...)
  __fl_attr((__format__ (__printf__, 1, 3)));

FL_EXPORT Fl_Widget *fl_message_icon();
extern FL_EXPORT Fl_Font fl_message_font_;
extern FL_EXPORT Fl_Fontsize fl_message_size_;
```cpp
int alloc_array;

private:
    int start(int XP, int YP, int WP, int HP, int &cx, int &cy,
              int &X, int &Y, int &W, int &H);

#if defined(__APPLE__) || defined(WIN32)
void *id_;
#else
unsigned id_;
#endif // __APPLE__ || WIN32

public:

Fl_Bitmap(const uchar *bits, int W, int H) :
    Fl_Image(W,H,0), array(bits), alloc_array(0), id_(0) {data((const char **)&array, 1);}
Fl_Bitmap(const char *bits, int W, int H) :
    Fl_Image(W,H,0), array((const uchar *)bits), alloc_array(0), id_(0) {data((const char **)&array, 1);}

virtual ~Fl_Bitmap();

virtual Fl_Bitmap *copy(int W, int H);

virtual void draw(int X, int Y, int W, int H, int cx=0, int cy=0);

void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}

virtual void label(Fl_Widget *w);

virtual void label(Fl_Menu_Item *m);

virtual void uncache();

};

#endif

// End of "$Id$".

// End of "$Id$".

10.13 Fl_BMP_Image.H

10.14 Fl_Box.H
```

```
10.13 Fl_BMP_Image.H

10.14 Fl_Box.H
```
10.15 Fl_Browser.H

00001 //
00002 // Fl_Browser widget . */
00003 //
00004 // Forms-compatible browser. Probably useful for other
00005 // lists of textual data. Notice that the line numbers
00006 // start from 1, and 0 means "no line".
00007 //
00008 #ifndef Fl_Browser_H
00009 #define Fl_Browser_H
00010 //
00011 #include "FL_EXPORT.h"
00012 #include "FL_BLINE.h"
00013 #include "FL_Image.H"
00014 //
00015 class FL_EXPORT Fl_Browser : public Fl_Widget {
00016 protected:
00017  void draw();
00018  FL_BLINE *first; // the array of lines
00019  FL_BLINE *last;
00020  int cacheline; // line number of cache
00021  int lines; // Number of lines
00022  int full_height_; //
00023  const int * column_widths_; //
00024  char format_char_; // alternative to @-sign
00025  char column_char_; // alternative to tab
00026
00027 #endif
00028
00029 struct FL_BLINE;
00030
00031 #endif
protected:

// required routines for Fl_Browser_ subclass:

void * item_first() const ;
void * item_next(void* item) const ;
void * item_prev(void* item) const ;
int item_selected(void* item) const ;
void item_select(void * item, int val);
int item_height(void * item) const ;
int item_width(void * item) const ;
void item_draw(void * item, int X, int Y, int W, int H) const ;
int full_height() const ;
int incr_height() const ;
const char *item_text(void *item) const ;
void item_swap(void *a, void *b) { swap((FL_BLINE*)a, (FL_BLINE*)b); }
void *item_at(int line) const { return (void*)find_line(line); }
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; }
void size(int W, int H) { Fl_Widget::size(W, H); }
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 middleline(int line) { lineposition(line, MIDDLE); }
int select(int line, int val=1);
int selected(int line) const ;
void show(int line);
void show(int line, const char* newtext);
void* data(int line) const ;
void data(int line, void* d);
int value() const ;
int value(int line) { select(line); }
const char * text(int line) const ;

Fl_Browser(int X, int Y, int W, int H, const char *L = 0);
~Fl_Browser() { clear(); }
char format_char() const { return format_char_; }
char column_char() const { return column_char_; }
const int * column_widths() const { return column_widths_; }
int displayed(int line) const { return FL_Browser_::displayed(find_line(line)); }

// icon support
void icon(int line, Fl_Image * icon);
void remove_icon(int line);

void replace(int a, const char * b) { text(a, b); }
void display(int line, int val=1);

void update_top();

protected:

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 = 0;
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 *redraw1,*redraw2; // minimal update pointers
virtual void *max_width_item; // which item has max_width_
virtual int hasScrollbar_; // which scrollbars are enabled
virtual int offset_; // how far down top_ item the real_position is
virtual int hposition_; // where user wants it panned to
virtual int real_hposition_; // the current horizontal scrolling position
virtual int position_; // where user wants it scrolled to
virtual int real_position_; // the current vertical scrolling position
virtual int max_width_; // widest object seen so far
virtual Fl_Font textfont_; // which font is used
virtual Fl_Fontsize textsize_; // which size is used
virtual Fl_Color textcolor_; // which color is used
virtual int scrollbar_size_; // size of scrollbar trough
virtual int sort(); // which sort is used
virtual int sort(ascending?); // which sort is used
虚拟函数必须由子类提供：

Fl_Browser_ widget.

### FL_Browser_.H

```c
#ifndef Fl_Browser__H
#define Fl_Browser__H

#include "Fl_Group.H"
#include "Fl_Scrollbar.H"
#include <FL/Fl.H> // Fl::scrollbar_size()

#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

class FL_EXPORT Fl_Browser_ : public Fl_Group {
private:
  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_; // which font is used
  Fl_Fontsize textsize_; // which size is used
  Fl_Color textcolor_; // which color is used
  int scrollbar_size_; // size of scrollbar trough
  virtual int sort(); // which sort is used
  virtual int sort(ascending?); // which sort is used

  Fl_Browser_();
public:
  int scrollbar_size() { return scrollbar_size_; }

  void update_top();

protected:

  void *item_first() const = 0;
  void *item_next(void *item) const = 0;
  void *item_prev(void *item) const = 0;
  void *item_last() const { return 0L; }
  int item_height(void *item) const = 0;
  int item_width(void *item) const = 0;
  int item_quick_height(void *item) const = 0;
  void *item_draw(void *item, int X, int Y, int W, int H) const = 0;
  const char *item_text(void *item) const { (void)item; return 0L; }
  void *redraw1,*redraw2; // minimal update pointers
  void *max_width_item; // which item has max_width_
  int hasScrollbar_; // which scrollbars are enabled
  int offset_; // how far down top_ item the real_position is
  int hposition_; // where user wants it panned to
  int real_hposition_; // the current horizontal scrolling position
  int position_; // where user wants it scrolled to
  int real_position_; // the current vertical scrolling position

  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 = 0;
  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; }
};
```

## File

Fl_Browser_.h

### 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:

http://www.fltk.org/COPYING.php

Please report all bugs and problems on the following page:

http://www.fltk.org/str.php

Generated by Doxygen
virtual void item_swap(void *a, void *b) { (void)a; (void)b; }
virtual void *item_at(int index) const { (void)index; return 0L; }
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
// These only need to be provided by subclass if you want a multi-browser:
virtual void *item_select(void *item, int val=1);
virtual int item_selected(void *item) const;
// things the subclass may want to call:
void *top() const { return top_; }
void selection() const { return selection_; }
void new_list(); // completely clobber all data, as though list replaced
void deleting(void *item); // get rid of any pointers to item
void replacing(void *a, void *b); // change a pointers to b
void swapping(void *a, void *b); // exchange pointers a and b
int displayed(void *item) const; // true if this item is visible
void redraw_line(void *item); // minimal update, no change in size
void redraw_lines() { damage(FL_DAMAGE_SCROLL); } // redraw all of them
void bbox(int &X, int &Y, int &W, int &H) const;
int leftedge() const; // x position after scrollbar & border
void *find_item(int ypos); // item under mouse

public:
Fl_Scrollbar scrollbar;
Fl_Scrollbar hscrollbar;
int handle(int event);
void resize(int X, int Y, int W, int H);
int select(void *item, int val=1, int docallbacks=0);
int select_only(void *item, int docallbacks=0);
int deselect(void *item, int docallbacks=0);
int position() const { return position_; }
int hposition() const { return hposition_; }
inum { // values for has_scrollbar()
HORIZONTAL = 1,
VERTICAL = 2,
BOTH = 3,
ALWAYS_ON = 4,
HORIZONTAL_ALWAYS = 5,
VERTICAL_ALWAYS = 6,
BOTH_ALWAYS = 7
};
uchar has_scrollbar() const { return has_scrollbar_; }
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_width() const { return Fl::scrollbar_size(); }
void scrollbar_width(int width) { Fl::scrollbar_size(width); }
void scrollbar_right() { scrollbar.align(FL_ALIGN_RIGHT); }
void scrollbar_left() { scrollbar.align(FL_ALIGN_LEFT); }
void sort(int flags=0);
void display(void *item); // scroll so this item is shown

Generated by Doxygen
**Fl_Button.H**

```
10.17 Fl_Button.H

00001 //
00002 // "$Id$"
00003 //
00004 // Button header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2014 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 /*
00020  
00021  
00022 #ifndef Fl_Button_H
00023 #define Fl_Button_H
00024
00025 #ifndef Fl_Widget_H
00026 #include "Fl_Widget.H"
00027 #endif
00028
00029 // values for type()
00030 #define FL_NORMAL_BUTTON 0
00031 #define FL_TOGGLE_BUTTON 1
00032 #define FL_RADIO_BUTTON (FL_RESERVED_TYPE+2)
00033 #define FL_HIDDEN_BUTTON 3
00034
00035 extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);
00036
00037 class Fl_Widget_Tracker;
00038
00039 class FL_EXPORT Fl_Button : public Fl_Widget {
00040
00041 int shortcut_;  
00042 char value_;  
00043 char oldval;  
00044 uchar down_box_;  
00045
00046 protected:
00047 static Fl_Widget_Tracker *key_release_tracker;
00048 static void key_release_timeout(void *);
00049 void simulate_key_action();
00050
00051 virtual void draw();
00052
00053 public:
00054 int handle(int);
00055 int value(int v);
00056 int set() {return value(1);}
00057 int clear() {return value(0);}
00058 void setonly(); // this should only be called on FL_RADIO_BUTTONs
00059 int shortcut() const {return shortcut_;
00060 void shortcut(const char *s) {shortcut(fl_old_shortcut(s));
00061 Fl_Boxtype down_box() const {return selection_color();
```

Generated by Doxygen
00167 00169 void down_color(unsigned c) {selection_color(c);} 00170 }; 00171 00172 #endif
00173 00174 //
00175 // End of "$Id$".
00176 //

10.18 Fl_Cairo.H

00001 //
00002 // "$Id$"
00003 //
00004 // Main header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2010 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 //   http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 //   http://www.fltk.org/str.php
00017 //
00018 //
00019 /*
00020 Handling transparently platform dependent cairo include files
00021 */
00022
00023 #ifndef FL_CAIRO_H
00024 # define FL_CAIRO_H
00025 # ifdef FLTK_HAVE_CAIRO
00026
00027 // Cairo is currently supported for the following platforms:
00028 // Win32, Apple Quartz, X11
00029
00030 # include <FL/Fl_Export.H>
00031
00032 # include <cairo.h>
00033
00034 class FL_EXPORT Fl_Cairo_State {
00035 public:
00036 Fl_Cairo_State() : cc_(0), own_cc_(false), autolink_(false), window_(0), gc_(0) {} 00037 00038 // access attributes
00039 00040 cairo_t* cc() const {return cc_;}
00041 00042 bool autolink() const {return autolink_;
00043 00044 void cc(cairo_t* c, bool own=true) {
00045   if (c && own) cairo_destroy(cc_); 00046   cc_=c;
00047   if (!cc_) window_=0;
00048   own_cc_=own;
00049 00050   }
00051 void autolink(bool b);
00052 void window(void* w) {window_=w;}
00053 void* window() const {return window_;}
00054 void gc(void* c) {gc_=c;}
00055 void* gc() const {return gc_;}
00056 private:
00057 cairo_t* cc_; 00058 // contains the unique autoupdated cairo context
00059 bool own_cc_; 00060 // indicates whether we must delete the cc, useful for internal cleanup
00061 bool autolink_; 00062 // false by default, prevents the automatic cairo mapping on fltk windows
00063 // for custom cairo implementations.
00064 void* window_, *gc_; 00065 // for keeping track internally of last win+gc treated
00066 00067 #endif // FLTK_HAVE_CAIRO
00068 #endif // FL_CAIRO_H
00069
00070 //
00071 // End of "$Id$".
00072 //

10.19 Fl_Cairo_Window.H

00001 //
file
Fl_Cairo_Window Handling transparently a fltk window incorporte a cairo draw callback.
*/

#ifndef FL_CAIRO_WINDOW_H
#define FL_CAIRO_WINDOW_H

#ifdef FLTK_HAVE_CAIRO

// Cairo is currently supported for the following platforms:
// Win32, Apple Quartz, X11

#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) : Fl_Double_Window(w,h),draw_cb_(0) {}

protected:
void draw() {
Fl_Double_Window::draw();
// manual method ? if yes explicitly get a cairo_context here
if (!Fl::cairo_autolink_context())
Fl::cairo_make_current(this);
if (draw_cb_) draw_cb_(this, 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_;}


#endif // FLTK_HAVE_CAIRO
#endif // FL_CAIRO_WINDOW_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;
    unsigned 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_;  // in FL_CRT/FL Core/FL_Symbol.h
    Fl_Fontsize textsize_;  // in FL_CRT/FL Core/FL_Symbol.h
    Fl_Color textcolor_;  // in FL_CRT/FL Core/FL_Symbol.h

    void draw();
    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);
    int size() const {return numb;}
    void size(int W, int H) { Fl_Widget::size(W, H); }
    int maxsize() const {return maxnumb;}
    void maxsize(int m);
    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;}
}

#endif

10.21 Fl_Check_Browser.H

// Fl_Check_Browser header file for the Fast Light Tool Kit (FLTK).
Fl_Check_Browser widget . */
#endif // Fl_Doxygen
struct cb_item {
    cb_item *next;
    cb_item *prev;
    char checked;
    char selected;
    char *text;
};
private:
    cb_item *first;
    cb_item *last;
    cb_item *cache;
    int cached_item;
    int nitems_; 
    int nchecked_; 
    cb_item *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
    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_; }
    cb_item *find_item(int) const;
    int lineno(cb_item *) const;
    protected:
        int handle(int);
10.22 Fl_Check_Button.H

/*
  class: Fl_Check_Button.
  
  A button with a "checkmark" to show its status.
*/

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);
};

10.23 Fl_Choice.H

/*
  file
  Fl_Choice widget . */

class FL_EXPORT Fl_Choice : public Fl_Menu_ {
public:
  Fl_Choice(int X, int Y, int W, int H, const char *L = 0);
};

Generated by Doxygen
int handle(int);

int 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);

Fl_Clock_Output(int X, int Y, int W, int H, const char *L = 0);

void drawhands(Fl_Color, Fl_Color); // part of draw

protected:

void draw();

public:

void 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_;

};

};

};

Fl_Clock(int X, int Y, int W, int H, const char *L = 0);

void value(ulong v); // part of draw

protected:

void draw();

public:

void Fl_Clock(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_;

};

};

};

class Fl_EXPORT Fl_Clock : public Fl_Clock_Output {

};

class Fl_EXPORT Fl_Clock_Output : public Fl_Widget {

};

class Fl_EXPORT Fl_Clock : public Fl_Clock_Output {

};
10.25 Fl_Color_Chooser.H File Reference

Fl_Color_Chooser widget.
#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.

10.25.1 Detailed Description

Fl_Color_Chooser widget.

10.26 Fl_Color_Chooser.H

Go to the documentation of this file.
class FL_EXPORT Flcc_ValueBox : public Fl_Widget {
int py;
protected:
void draw();
int handle_key(int);
public:
int handle(int);
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*); // Format the string
Flcc_Value_Input(int X, int Y, int W, int H) : Fl_Value_Input(X,Y,W,H) {} // Constructor
};

#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;
double hue_, saturation_, value_; // HSV values
double r_, g_, b_; // RGB values
void set_valuators(); // Set the valuators
static void rgb_cb(Fl_Widget *, void*);
static void mode_cb(Fl_Widget *, void*);
public:
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_}
int 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 *L = 0);
};

FL_EXPORT int fl_color_chooser(const char* name, double& r, double &g, double &b, int m=-1);
FL_EXPORT int fl_color_chooser(const char* name, uchar& r, uchar &g, uchar &b, int m=-1);
#endif

// End of "$Id$".
// Copyright 1998-2014 by Bill Spitzak and others.

10.27 Fl_Copy_Surface.H
#ifndef Fl_Copy_Surface_H
#define Fl_Copy_Surface_H

#include <FL/Fl_Paged_Device.H>
#include <FL/Fl_Printer.H>
#include <FL/x.H>

class FL_EXPORT Fl_Copy_Surface : public Fl_Surface_Device {
private:
  int width;
  int height;
  Fl_Paged_Device *helper;
  #ifdef __APPLE__
    CFMutableDataRef pdfdata;
  #endif
  CGContextRef oldgc;
  CGContextRef gc;
  void prepare_copy_pdf_and_tiff(int w, int h);
  void complete_copy_pdf_and_tiff();
  void init_PDF_context(int w, int h);
  static size_t MyPutBytes(void *info, const void* buffer, size_t count);
  #elif defined(WIN32)
    HDC oldgc;
    HDC gc;
  #else // Xlib
    Fl_Offscreen xid;
    Window oldwindow;
    Fl_Surface_Device *ss;
  #endif
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Copy_Surface(int w, int h);
  ~Fl_Copy_Surface();
  void set_current();
  void draw(Fl_Widget *widget, int delta_x = 0, int delta_y = 0);
  void draw_decorated_window(Fl_Window *win, int delta_x = 0, int delta_y = 0);
  int w() { return width; }
  int h() { return height; }
};

#if defined(__APPLE__)

// Mac class to reimplement Fl_Paged_Device::printable_rect()
class FL_EXPORT Fl_Quartz_Surface_ : public Fl_System_Printer {
protected:
  int width;
  int height;
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Quartz_Surface_(int w, int h);
  virtual int printable_rect(int *w, int *h);
  virtual ~Fl_Quartz_Surface_() {};
};
#endif

#elif defined(WIN32)

// Win class to implement translate()/untranslate()
class FL_EXPORT Fl_GDI_Surface_ : public Fl_Paged_Device {
  int width;
  int height;
  unsigned depth;
  POINT origins[10];
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_GDI_Surface_();
  virtual void translate(int x, int y);
  virtual void untranslate();
  virtual ~Fl_GDI_Surface_();
};
#endif

#elif !defined(FL_DOXYGEN)

// Xlib class to implement translate()/untranslate()
class FL_EXPORT Fl_Xlib_Surface_ : public Fl_Paged_Device {
  int width;
  int height;
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Xlib_Surface_();
  virtual void translate(int x, int y);
  virtual void untranslate();
  virtual ~Fl_Xlib_Surface_();
};
#endif

// endif defined(_APPLE_)
#endif

/* Mac class to reimplement Fl_Paged_Device::printable_rect() */
class FL_EXPORT Fl_Quartz_Surface_ : public Fl_System_Printer {
  int width;
  int height;
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Quartz_Surface_(int w, int h);
  virtual int printable_rect(int *w, int *h);
  virtual ~Fl_Quartz_Surface_() {};
};

/* Win class to implement translate()/untranslate() */
class FL_EXPORT Fl_GDI_Surface_ : public Fl_Paged_Device {
  int width;
  int height;
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_GDI_Surface_();
  virtual void translate(int x, int y);
  virtual void untranslate();
  virtual ~Fl_GDI_Surface_();
};

/* Xlib class to implement translate()/untranslate() */
class FL_EXPORT Fl_Xlib_Surface_ : public Fl_Paged_Device {
  int width;
  int height;
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Xlib_Surface_();
  virtual void translate(int x, int y);
  virtual void untranslate();
  virtual ~Fl_Xlib_Surface_();
};

Generated by Doxygen
public:
    static const char *class_id;
    const char *class_name() {return class_id;};
    Fl_Xlib_Surface_();
    virtual void translate(int x, int y);
    virtual void untranslate();
    virtual ~Fl_Xlib_Surface_();

#endif  // Fl_Copy_Surface_H

#endif // Fl_Counter_H
10.29  Fl_Device.H File Reference

#include <FL/x.H>
#include <FL/Fl_Plugin.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Bitmap.H>
#include <FL/Fl_Pixmap.H>
#include <FL/Fl_RGB_Image.H>
#include <stdlib.h>

Classes

• class Fl_Device
  All graphical output devices and all graphics systems.
• class Fl_Device_Plugin
  This plugin socket allows the integration of new device drivers for special window or screen types.
• class Fl_Display_Device
  A display to which the computer can draw.
• class Fl_GDI_Graphics_Driver
  The MSWindows-specific graphics class.
• class Fl_GDI_Printer_Graphics_Driver
  The graphics driver used when printing on MSWindows.
• class Fl_Graphics_Driver
  A virtual class subclassed for each graphics driver FLTK uses.
• class Fl_Quartz_Graphics_Driver
  The Mac OS X-specific graphics class.
• class Fl_Surface_Device
  A drawing surface that's susceptible to receive graphical output.
• class Fl_Xlib_Graphics_Driver
  The Xlib-specific graphics class.
• struct Fl_Graphics_Driver::matrix
  A 2D coordinate transformation matrix.

Macros

• #define FL_MATRIX_STACK_SIZE 32
• #define FL_REGION_STACK_SIZE 10
• #define XPOINT XPoint

Typedefs

• typedef short COORD_T
• typedef void(* Fl_Draw_Image_Callback)(void *data, int x, int y, int w, uchar *buf)
  signature of image generation callback function.
Variables

- FL_EXPORT Fl_Graphics_Driver * fl_graphics_driver

Points to the driver that currently receives all graphics requests.

10.29.1 Detailed Description


10.29.2 Typedef Documentation

10.29.2.1 Fl_Draw_Image_Cb

typedef void(* Fl_Draw_Image_Cb)(void* data, int x, int y, int w, uchar *buf)

Signature of image generation callback function.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data user data passed to function</td>
</tr>
<tr>
<td>in</td>
<td>x,y,w position and width of scan line in image</td>
</tr>
<tr>
<td>out</td>
<td>buf buffer for generated image data. You must copy w pixels from scanline y, starting at pixel x to this buffer.</td>
</tr>
</tbody>
</table>

10.30 Fl_Device.H

Go to the documentation of this file.

Generated by Doxygen
```cpp
typedef short COORD_T;
#define XPOINT XPoint

class FL_EXPORT Fl_Device {
public:
    static const char *class_id;
    virtual const char *class_name() { return class_id; }; 
    virtual ~Fl_Device() {}; 
private:
    static const matrix m0;
    Fl_Font font_; // current font 
    Fl_Fontsize size_; // current font size
    Fl_Color color_; // current color 
    int sptr;
    static const int matrix_stack_size = FL_MATRIX_STACK_SIZE;
    matrix stack[FL_MATRIX_STACK_SIZE];
    matrix m;
    int n, p_size, gap_; 
    XPOINT *p;
    int what;
    int fl_clip_state_number;
    static const int region_stack_max = FL_REGION_STACK_SIZE - 1;
    Fl_Region rstack[FL_REGION_STACK_SIZE];
    #ifdef WIN32
    int numcount;
    int counts[20];
    #endif
    Fl_Font_Descriptor *font_descriptor_; 
    void transformed_vertex0(COORD_T x, COORD_T y); 
    void fixloop(); 
    protected:
    #ifndef FL_DOXYGEN
    enum {LINE, LOOP, POLYGON, POINT_};
    inline int vertex_no() { return n; } 
    inline XPOINT *vertices() {return p;}
    inline int vertex_kind() {return what;}
    #endif
    uchar bg_r_;
    uchar bg_g_; 
    uchar bg_b_; 
    friend class Fl_Pixmap; 
    friend class Fl_Bitmap; 
    friend class Fl_RGB_Image; 
    friend void fl_rect(int x, int y, int w, int h);
    friend void fl_rectf(int x, int y, int w, int h);
    friend void fl_line_style(int style, int width, char * dashes);
    friend void fl_xyline(int x, int y, int x1);
    friend void fl_xyline(int x, int y, int x1, int y1, int x2);
    friend void fl_xylne(int x, int y, int x1, int y1, int x2, int y2);
    friend void fl_xylne(int x, int y, int x1, int y1, int x2, int y3);
    friend void fl_xylne(int x, int y, int x1, int y1, int x2, int y2, int y3);
    friend void fl_yxline(int x, int y, int y1);
    friend void fl_yxline(int x, int y, int y1, int x2);
    friend void fl_yxline(int x, int y, int y1, int x2, int y3);
    friend void fl_yxline(int x, int y, int x1, int y1, int x2, int y2, int y3);
    #ifdef __APPLE__
    friend void fl_draw(const char *str, int n, float x, float y);
    #endif
    friend void fl_draw(const char *str, int n, float x, float y);
    friend void fl_draw(int angle, const char *str, int n, int x, int y);
    friend void fl_fil_draw(const char *str, int n, int x, int y);
    friend void fl_font(Fl_Font face, Fl_Fontsize size);
    friend void fl_color(Fl_Color c);
    friend void fl_color(uchar r, uchar g, uchar b);
    friend void fl_point(int x, int y);
    friend void fl_loop(int x0, int y0, int x1, int y1, int x2, int y2);
    friend void fl_loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
    friend void fl_polyline(int x0, int y0, int x1, int y1, int x2, int y2);
    friend void fl_polyline(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
    friend void fl_begin_points();
    friend void fl_begin_line();
    friend void fl_begin_loop();
    friend void fl_begin_polygon();
    friend void fl_vertex(double x, double y);
    friend void fl_curve(double x0, double y0, double x1, double y1, double x2, double y2, double x3,
```
double Y3);
00186 friend void fl_circle(double x, double y, double r);
00187 friend void fl_arc(double x, double y, double r, double start, double end);
00188 friend void fl_pie(int x, int y, int w, int h, double a1, double a2);
00189 friend void fl_end_points();
00190 friend void fl_end_line();
00191 friend void fl_end_loop();
00192 friend void fl_end_polygon();
00193 friend void fl_transformed_vertex(double xf, double yf);
00194 friend void fl_transformed_polygon(double x, double y);
00195 friend void fl_push_clip(int x, int y, int w, int h);
00196 friend int fl_clip_box(int x, int y, int w, int h, int x0, int y0, int x1, int y1);
00197 friend int fl_not_clipped(int x, int y, int w, int h);
00198 friend void fl_push_no_clip();
00199 friend void fl_pop_clip();
00200 friend void fl_begin_complex_polygon();
00201 friend void fl_gap();
00202 friend void fl_end_complex_polygon();
00203 friend void fl_push_matrix();
00204 friend void fl_pop_matrix();
00205 friend void fl_mult_matrix(double a, double b, double c, double d, double x, double y);
00206 friend void fl_scale(double x, double y);
00207 friend void fl_translate(double x, double y);
00208 friend void fl_rotate(double d);
00209 friend double fl_transform_x(double x, double y);
00210 friend double fl_transform_y(double x, double y);
00211 friend double fl_transform_dx(double x, double y);
00212 friend double fl_transform_dy(double x, double y);
00213 friend Fl_Region fl_clip_region();
00214 friend void fl_clip_region(Fl_Region r);
00215 friend void fl_restore_clip();
00216
00217 friend void fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L);
00218 friend void fl_draw_image_mono(const uchar * buf, int X, int Y, int W, int H, int D, int L);
00219 friend void fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D);
00220 friend void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D);
00221 friend FL_EXPORT void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D);
00222 friend FL_EXPORT void gl_start();
00223 friend FL_EXPORT void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
00224 matrix *fl_matrix;
00225 Fl_Graphics_Driver();
00226 virtual void rect(int x, int y, int w, int h);
00227 virtual void rectf(int x, int y, int w, int h);
00228 virtual void line_style(int style, int width=0, char * dashes=0);
00229 virtual void xyline(int x, int y, int x1);
00230 virtual void xyline(int x, int y, int x1, int y2);
00231 virtual void xyline(int x, int y, int x1, int y2, int x3);
00232 virtual void yxline(int x, int y, int y1);
00233 virtual void yxline(int x, int y, int y1, int x2);
00234 virtual void yxline(int x, int y, int y1, int x2, int y3);
00235 virtual void line(int x, int y, int x1, int y1);
00236 virtual void line(int x, int y, int x1, int y1, int x2, int y2);
00237 virtual void draw(const char *str, int n, int x, int y) {};
00238 #ifdef __APPLE__
00239 virtual void draw(const char *str, int n, float x, float y) { draw(str, n, (int)(x+0.5), (int)(y+0.5)); }
00240 #endif
00241 virtual void begin_points();
00242 virtual void begin_line();
00243 virtual void begin_loop();
00244 virtual void begin_polygon();
00245 virtual void vertex(double x, double y);
00246 virtual void curve(double X0, double Y0, double X1, double Y1, double X2, double Y2);
00247 virtual void circle(double x, double y, double r);
00248 virtual void arc(double x, double y, double r, double start, double end);
00249 virtual void pie(int x, int y, int w, int h, double a1, double a2);
00250 virtual void end_points();
00251 virtual void end_line();
00252 virtual void end_loop();
00253 virtual void end_polygon();
00254 virtual void transformed_vertex(double xf, double yf);
virtual void push_clip(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();

void push_matrix();
void pop_matrix();
void mult_matrix(double a, double b, double c, double d, double x, double y);
inline void scale(double x, double y) { mult_matrix(x,0,0,y,0); }
inline void translate(double x, double y) { mult_matrix(1,0,0,1,x,y); }
void rotate(double d);
void transform_x(double x, double y);
void transform_y(double x, double y);
double transform_dx(double x, double y);
double transform_dy(double x, double y);

void push_no_clip();

void draw_image(const uchar * buf, int X, int Y, int W, int H, int D=3, int L=0) {}
void draw_image_mono(const uchar * buf, int X, int Y, int W, int H, int D=1, int L=0) {}
void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=3) {}
void draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) {}
class FL_EXPORT Fl_GDI_Graphics_Driver : public Fl_Graphics_Driver {
public:
    static const char *class_id;
    const char *class_name() {return class_id;};
    void color(Fl_Color c);
    void color(uchar r, uchar g, uchar b);
    void draw(const char * str, int n, int x, int y);
    void rtl_draw(const char * str, int n, int x, int y);
    void font(Fl_Font face, Fl_Fontsize size);
    void draw(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw(Fl_Bitmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw(Fl_RGB_Image *img, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
    void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
    double width(const char *str, int n);
    double width(unsigned int c);
    void text_extents(const char *, int n, int& dx, int& dy, int& w, int& h);
    int height();
    int descent();
    void copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
};

class FL_EXPORT Fl_GDI_Printer_Graphics_Driver : public Fl_GDI_Graphics_Driver {
public:
    static const char *class_id;
    const char *class_name() {return class_id;};
    void draw(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy);
    int draw_scaled(Fl_Image *img, int XP, int YP, int WP, int HP, int cx, int cy);
};

class FL_EXPORT Fl_Xlib_Graphics_Driver : public Fl_Graphics_Driver {
public:
    static const char *class_id;
    const char *class_name() {return class_id;};
    void color(Fl_Color c);
    void color(uchar r, uchar g, uchar b);
    void draw(const char * str, int n, int x, int y);
    void rtl_draw(const char * str, int n, int x, int y);
    void font(Fl_Font face, Fl_Fontsize size);
    void draw(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw(Fl_Bitmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw(Fl_RGB_Image *img, int XP, int YP, int WP, int HP, int cx, int cy);
    void draw_image(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
    void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X, int Y, int W, int H, int D);
    double width(const char *str, int n);
    double width(unsigned int c);
    void text_extents(const char *, int n, int& dx, int& dy, int& w, int& h);
    int height();
    int descent();
    void copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
    void copy_offscreen_with_alpha(int x, int y, int w, int h, HBITMAP bitmap, int srcx, int srcy);
};

class FL_EXPORT Fl_Surface_Device : public Fl_Device {
    Fl_Graphics_Driver *_driver;
    static Fl_Surface_Device *_surface; // the surface that currently receives graphics output
    static Fl_Surface_Device *default_surface(); // create surface is none exists yet
protected:
    void set_current(void);
    inline void driver(Fl_Graphics_Driver *graphics_driver) { _driver = graphics_driver; };
class FL_EXPORT Fl_Display_Device : public Fl_Surface_Device {
static Fl_Display_Device * _display; // the platform display device
#ifdef __APPLE__
friend class Fl_X;
friend class Fl_Graphics_Driver;
static bool high_res_window_; // true when drawing to a window of a retina display (Mac OS X only)
static bool high_resolution() {return high_res_window_ ;}
#endif
public:
static const char * class_id;
const char * class_name() {return class_id ;};
Fl_Display_Device(Fl_Graphics_Driver * graphics_driver);
static Fl_Display_Device * display_device();
};

class FL_EXPORT Fl_Device_Plugin : public Fl_Plugin {
public:
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, int x, int y, int height) = 0;
#ifdef FL_LIBRARY
virtual
#endif
Fl_RGB_Image * rectangle_capture(Fl_Widget *widget, int x, int y, int w, int h) {return NULL;}
};
#endif // Fl_Device_H

// End of "$Id$".

10.31 Fl_Dial.H

#ifndef Fl_Dial_H
#define Fl_Dial_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

#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:
void draw(int X, int Y, int W, int H);
int handle(int event, int X, int Y, int W, int H);
void draw();

public:
short angle1() const {return a1 ;};
};

// File Fl_Dial widget . */
#endif // Fl_Dial_H
#define Fl_Dial_H
#define Fl_Valuator_H
#include "Fl_Valuator.H"
#endif // Fl_Valuator.H

// 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:
void draw(int X, int Y, int W, int H);
int handle(int event, int X, int Y, int W, int H);
void draw();

public:
short angle1() const {return a1 ;};
};

// these allow subclasses to put the dial in a smaller area:
void draw(int X, int Y, int W, int H);
int handle(int event, int X, int Y, int W, int H);
void draw();

// please report all bugs and problems on the following page:

// 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:

// http://www.fltk.org/COPYING.php

// Please report all bugs and problems on the following page:

// http://www.fltk.org/str.php

// Dec 1998 - Bill Spitzak

// Copyright 1998-2010 by Bill Spitzak and others.
10.32 Fl_Double_Window.H

```
00074  void angle1(short a) {a1 = a;}
00076  short angle2() const {return a2;}
00078  void angle2(short a) {a2 = a;}
00080  void angles(short a, short b) {a1 = a; a2 = b;}
00081
00082 };}
00083
00084 #endif
00085
00086 //
00087 // End of "$Id$",
00088 //
```

10.33 fl_draw.H File Reference

utility header to pull drawing functions together

```c
#include <FL/x.H>
#include <FL/Enumerations.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Device.H>
```

Macros

- `#define fl_clip fl_push_clip`

---

Generated by Doxygen
Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

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

- FL_EXPORT int fl_add_symbol (const char *name, void(*drawit)(Fl_Color), int scalable)
  Adds a symbol to the system.
- void fl_arc (double x, double y, double r, double start, double end)
  Adds 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 ()
  Starts drawing a complex filled polygon.
- void fl_begin_line ()
  Starts drawing a list of lines.
- void fl_begin_loop ()
  Starts drawing a closed sequence of lines.
- void fl_begin_points ()
  Starts drawing a list of points.
- void fl_begin_polygon ()
  Starts drawing a convex filled polygon.
- FL_EXPORT char fl_can_do_alpha_blending ()
  Checks whether platform supports true alpha blending for RGBA images.
- FL_EXPORT 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() is equivalent to fl_arc(x,y,r,0,360), but may be faster.
- int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)
  Intersects the rectangle with the current clip region and returns the bounding box of the result.
- Fl_Region fl_clip_region ()
  Returns the current clipping region.
- void fl_clip_region (Fl_Region r)
  Replaces the top of the clipping stack with a clipping region of any shape.
- Fl_Color fl_color ()
  Returns the last fl_color() that was set.
- void fl_color (Fl_Color c)
  Sets 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)
  Sets the color for all subsequent drawing operations.
- FL_EXPORT void fl_cursor (Fl_Cursor)
  Sets the cursor for the current window to the specified shape and colors.
- FL_EXPORT void fl_cursor (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)
- void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
  Adds a series of points on a Bezier curve to the path.
• int fl_descent ()
  Returns 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.

• FL_EXPORT void fl_draw (const char *str, int x, int y)
  Draws a null-terminated UTF-8 string starting at the given x, y location.

• 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)
  Fancy string drawing function which is used to draw all the labels.

• FL_EXPORT void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, void(*)(const char *, int, int, int), 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.

• FL_EXPORT void fl_draw (int angle, const char *str, int x, int y)
  Draws a null-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

• FL_EXPORT 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_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  Draws 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)
  Draws 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)
  Draws 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)
  Draws a gray-scale image using a callback function to generate image data.

• FL_EXPORT int fl_draw_pixmap (char *const *data, int x, int y, Fl_Color=FL_GRAY)
  Draw XPM image data, with the top-left corner at the given position.

• FL_EXPORT int fl_draw_pixmap (const char *const *cdata, int x, int y, Fl_Color=FL_GRAY)
  Draw XPM image data, with the top-left corner at the given position.

• FL_EXPORT 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 ()
  Ends complex filled polygon, and draws.

• void fl_end_line ()
  Ends list of lines, and draws.

• void fl_end_loop ()
  Ends closed sequence of lines, and draws.

• void fl_end_points ()
  Ends list of points, and draws.

• void fl_end_polygon ()
  Ends convex filled polygon, and draws.

• FL_EXPORT 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.

• FL_Font fl_font ()
  Returns 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.
• FL_EXPORT void fl_frame (const char *s, int x, int y, int w, int h)
  Draws a series of line segments around the given box.
• FL_EXPORT 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 ()
  Call fl_gap() to separate loops of the path.
• int fl_height ()
  Returns the recommended minimum line spacing for the current font.
• FL_EXPORT int fl_height (int font, int size)
  This function returns the actual height of the specified font and size.
• FL_EXPORT const char * fl_latin1_to_local (const char *t, int n=-1)
  Converts text from Windows/X11 latin1 character set to local encoding.
• void fl_line (int x, int y, int x1, int y1)
  Draws a line from (x,y) to (x1,y1)
• void fl_line (int x, int y, int x1, int y1, int x2, int y2)
  Draws 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)
  Sets how to draw lines (the "pen").
• FL_EXPORT const char * fl_local_to_latin1 (const char *t, int n=-1)
  Converts text from local encoding to Windows/X11 latin1 character set.
• FL_EXPORT const char * fl_local_to_mac_roman (const char *t, int n=-1)
  Converts text from local encoding to Mac Roman character set.
• void fl_loop (int x, int y, int x1, int y1, int x2, int y2)
  Outlines 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)
  Outlines a 4-sided polygon with lines.
• FL_EXPORT const char * fl_mac_roman_to_local (const char *t, int n=-1)
  Converts text from Mac Roman character set to local encoding.
• FL_EXPORT 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.
• FL_EXPORT int fl_measure_pixmap (char * const * data, int &w, int &h)
  Get the dimensions of a pixmap.
• FL_EXPORT 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)
  Concatenates 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?
• FL_EXPORT unsigned int fl_old_shortcut (const char *s)
  Emulation of XForms named shortcuts.
• FL_EXPORT void fl_overlay_clear ()
  Erase a selection rectangle without drawing a new one.
• FL_EXPORT void fl_overlay_rect (int x, int y, int w, int h)
  Draws a selection rectangle, erasing a previous one by XOR'ing it first.
• 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)
  Draws a single pixel at the given coordinates.
• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2)
  Fills a 3-sided polygon.
Fills a 4-sided polygon.

- void fl_pop_clip()
  Restores the previous clip region.

- void fl_pop_matrix()
  Restores the current transformation matrix from the stack.

- void fl_push_clip(int x, int y, int w, int h)
  Intersects the current clip region with a rectangle and pushes this new region onto the stack.

- void fl_push_matrix()
  Saves the current transformation matrix on the stack.

- void fl_push_no_clip()
  Pushes an empty clip region onto the stack so nothing will be clipped.

- FL_EXPORT 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(int x, int y, int w, int h)
  Draws a 1-pixel border inside the given bounding box.

- void fl_rect(int x, int y, int w, int h, Fl_Color c)
  Draws with passed color a 1-pixel border inside the given bounding box.

- void fl_rectf(int x, int y, int w, int h)
  Colors 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)
  Colors with passed color a rectangle that exactly fills the given bounding box.

- FLEXPORT void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)
  Colors a rectangle with "exactly" the passed r, g, b color.

- FL_EXPORT void fl_reset_spot()

- void fl_restore_clip()
  Undoes any clobbering of clip done by your program.

- void fl_rotate(double d)
  Concatenates rotation transformation onto the current one.

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

- void fl_scale(double x)
  Concatenates scaling transformation onto the current one.

- void fl_scale(double x, double y)
  Concatenates scaling transformation onto the current one.

- FL_EXPORT 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.

- 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_set_status(int X, int Y, int W, int H)

- FL_EXPORT const char* flShortcut_label(unsigned int shortcut)
  Get a human-readable string from a shortcut value.

- FL_EXPORT const char* flShortcut_label(unsigned int shortcut, const char** eom)

- Fl_Fontsize fl_size()
  Returns the size set by the most recent call to fl_font().

- FL_EXPORT void fl_text_extents(const char*, int &dx, int &dy, int &w, int &h)
  Determines the minimum pixel dimensions of a null-terminated string.

- void fl_text_extents(const char*t, int n, int &dx, int &dy, int &w, int &h)
  Determines the minimum pixel dimensions of a sequence of n characters.

- double fl_transform_dx(double x, double y)
  Transforms distance using current transformation matrix.
• double fl_transform_dy (double x, double y)
  Transforms distance using current transformation matrix.
• double fl_transform_x (double x, double y)
  Transforms coordinate using the current transformation matrix.
• double fl_transform_y (double x, double y)
  Transforms coordinate using the current transformation matrix.
• void fl_transformed_vertex (double xf, double yf)
  Adds coordinate pair to the vertex list without further transformations.
• void fl_translate (double x, double y)
  Concatenates translation transformation onto the current one.
• void fl_vertex (double x, double y)
  Adds a single vertex to the current path.
• FL_EXPORT double fl_width (const char *txt)
  Returns the typographical width of a null-terminated string using the current font face and size.
• double fl_width (const char *txt, int n)
  Returns the typographical width of a sequence of \( n \) characters using the current font face and size.
• double fl_width (unsigned int c)
  Returns the typographical width of a single character using the current font face and size.
• void fl_xyline (int x, int y, int x1)
  Draws a horizontal line from \((x,y)\) to \((x1,y)\)
• void fl_xyline (int x, int y, int x1, int y2)
  Draws 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)
  Draws 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)
  Draws a vertical line from \((x,y)\) to \((x,y1)\)
• void fl_yxline (int x, int y, int y1, int x2)
  Draws 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)
  Draws 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
• FL_EXPORT char fl_drawShortcut

10.33.1 Detailed Description
utility header to pull drawing functions together

10.34 fl_draw.H
Go to the documentation of this file.
#ifndef fl_draw_H
#define fl_draw_H

#include <FL/x.H> // for Fl_Region
#include <FL/Enumerations.H> // for the color names
#include <FL/Fl_Window.H> // for fl_set_spot()
#include <FL/Fl_Device.H> // for fl_graphics_driver

// Image class...
class Fl_Image;

// Label flags...
#define fl_draw_shortcut

// Colors:
inline void fl_color(Fl_Color c) {fl_graphics_driver->color(c); } // select indexed color
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); } // select actual color
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); }
#define fl_clip fl_push_clip
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 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_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_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(int x, int y, int w, int h, uchar r, uchar g, uchar 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);}

// 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);}
1666

00250
00251
00255
00259
00264
00265
00266
00270
00274
00279

File Documentation

// 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
inline void fl_yxline(int x, int y, int y1)
inline void fl_yxline(int x, int y, int y1,
inline void fl_yxline(int x, int y, int y1,
{fl_graphics_driver->yxline(x,y,y1,x2,y3);}

first:
{fl_graphics_driver->yxline(x,y,y1);}
int x2) {fl_graphics_driver->yxline(x,y,y1,x2);}
int x2, int y3)

00280
00281 // circular lines and pie slices (code in fl_arci.C):
00304 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); }
00317 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); }
00319 FL_EXPORT void fl_chord(int x, int y, int w, int h, double a1, double a2); // nyi
00320
00321 // scalable drawing code (code in fl_vertex.C and fl_arc.C):
00326 inline void fl_push_matrix() { fl_graphics_driver->push_matrix(); }
00330 inline void fl_pop_matrix() { fl_graphics_driver->pop_matrix(); }
00335 inline void fl_scale(double x, double y) { fl_graphics_driver->scale(x, y); }
00340 inline void fl_scale(double x) { fl_graphics_driver->scale(x, x); }
00345 inline void fl_translate(double x, double y) { fl_graphics_driver->translate(x, y); }
00350 inline void fl_rotate(double d) { fl_graphics_driver->rotate(d); }
00357 inline void fl_mult_matrix(double a, double b, double c, double d, double x,double y)
00358
{ fl_graphics_driver->mult_matrix(a, b, c, d, x, y); }
00362 inline void fl_begin_points() {fl_graphics_driver->begin_points(); }
00366 inline void fl_begin_line() {fl_graphics_driver->begin_line(); }
00370 inline void fl_begin_loop() {fl_graphics_driver->begin_loop(); }
00374 inline void fl_begin_polygon() {fl_graphics_driver->begin_polygon(); }
00379 inline void fl_vertex(double x, double y) {fl_graphics_driver->vertex(x,y); }
00388 inline void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3,
double Y3)
00389
{fl_graphics_driver->curve(X0,Y0,X1,Y1,X2,Y2,X3,Y3); }
00416 inline void fl_arc(double x, double y, double r, double start, double end)
{fl_graphics_driver->arc(x,y,r,start,end); }
00424 inline void fl_circle(double x, double y, double r) {fl_graphics_driver->circle(x,y,r); }
00428 inline void fl_end_points() {fl_graphics_driver->end_points(); }
00432 inline void fl_end_line() {fl_graphics_driver->end_line(); }
00436 inline void fl_end_loop() {fl_graphics_driver->end_loop(); }
00440 inline void fl_end_polygon() {fl_graphics_driver->end_polygon(); }
00455 inline void fl_begin_complex_polygon() {fl_graphics_driver->begin_complex_polygon(); }
00462 inline void fl_gap() {fl_graphics_driver->gap(); }
00466 inline void fl_end_complex_polygon() {fl_graphics_driver->end_complex_polygon(); }
00467 // get and use transformed positions:
00472 inline double fl_transform_x(double x, double y) {return fl_graphics_driver->transform_x(x, y); }
00477 inline double fl_transform_y(double x, double y) {return fl_graphics_driver->transform_y(x, y); }
00482 inline double fl_transform_dx(double x, double y) {return fl_graphics_driver->transform_dx(x, y); }
00487 inline double fl_transform_dy(double x, double y) {return fl_graphics_driver->transform_dy(x, y); }
00492 inline void fl_transformed_vertex(double xf, double yf)
{fl_graphics_driver->transformed_vertex(xf,yf); }
00497 /* NOTE: doxygen comments here to avoid triplication in os-specific sources */
00498
00499 // Fonts:
00509 inline void fl_font(Fl_Font face, Fl_Fontsize fsize) { fl_graphics_driver->font(face,fsize); }
00510
00515 inline Fl_Font fl_font() {return fl_graphics_driver->font();}
00520 inline Fl_Fontsize fl_size() {return fl_graphics_driver->size();}
00521
00522 // information you can get about the current font:
00527 inline int fl_height() {return fl_graphics_driver->height();}
00528 FL_EXPORT int fl_height(int font, int size);
00533 inline int fl_descent() {return fl_graphics_driver->descent();}
00536 FL_EXPORT double fl_width(const char* txt);
00539 inline double fl_width(const char* txt, int n) {return fl_graphics_driver->width(txt, n);}
00544 inline double fl_width(unsigned int c) {return fl_graphics_driver->width(c);}
00555 FL_EXPORT void fl_text_extents(const char*, int& dx, int& dy, int& w, int& h); // NO fltk symbol
expansion will be performed
00559 inline void fl_text_extents(const char *t, int n, int& dx, int& dy, int& w, int& h)
00560
{fl_graphics_driver->text_extents(t, n, dx, dy, w, h);}
00561
00562 // font encoding:
00563 // Note: doxygen comments here to avoid duplication for os-sepecific cases
00570 FL_EXPORT const char *fl_latin1_to_local(const char *t, int n=-1);
00577 FL_EXPORT const char *fl_local_to_latin1(const char *t, int n=-1);
00584 FL_EXPORT const char *fl_mac_roman_to_local(const char *t, int n=-1);
00591 FL_EXPORT const char *fl_local_to_mac_roman(const char *t, int n=-1);
00606 FL_EXPORT void fl_draw(const char* str, int x, int y);
00614 FL_EXPORT void fl_draw(int angle, const char* str, int x, int y);
00618 inline void fl_draw(const char* str, int n, int x, int y) {fl_graphics_driver->draw(str,n,x,y); }
00628 inline void fl_draw(int angle, const char* str, int n, int x, int y)
{fl_graphics_driver->draw(angle,str,n,x,y); }
00632 inline void fl_rtl_draw(const char* str, int n, int x, int y)
{fl_graphics_driver->rtl_draw(str,n,x,y); }

Generated by Doxygen


void (*callthis)(const char *, int, int, int, Fl_Color);

// boxtypes:

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);

// images:

inline 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);
}
inline 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, L);
}
inline 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);
}
inline 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);
}

/* note: doxygen comment here to avoid triplication in os-specific files */
FL_EXPORT char fl_can_do_alpha_blending();

/* note: doxygen comment here to avoid triplication in os-specific files */
FL_EXPORT uchar *fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha = 0);

// pixmaps:

FL_EXPORT int fl_draw_pixmap(const char* const* data, int x, int y, Fl_Color = FL_GRAY);
FL_EXPORT int fl_draw_pixmap(const char* const* cdata, int x, int y, Fl_Color = FL_GRAY);
FL_EXPORT int fl_measure_pixmap(const char* const* data, int &w, int &h);
FL_EXPORT int fl_measure_pixmap(const char* const* cdata, int &w, int &h);

// other:

FL_EXPORT 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);
FL_EXPORT const char* fl_shortcut_label(unsigned int shortcut);
FL_EXPORT const char* fl_shortcut_label(unsigned int shortcut, const char **eom);
FL_EXPORT unsigned int fl_oldShortcut(const char* s);
FL_EXPORT void fl_overlay_rect(int x, int y, int w, int h);
FL_EXPORT void fl_overlay_clear();
FL_EXPORT void fl_cursor(Fl_Cursor);
FL_EXPORT void fl_cursor(Fl_Cursor, Fl_Color fg, Fl_Color bg = FL_WHITE);
FL_EXPORT const char* fl_expand_text(const char* from, char* buf, int maxbuf, double maxw, int& n, double &width, int wrap, int draw_symbols = 0);

// XIM:

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);
FL_EXPORT int fl_add_symbol(const char* name, void (*drawit)(Fl_Color), int scalable);

#include <stdio.h>

*/

// End of "$Id$".

*/

10.35 Fl_Export.H

00001 /*
00002 "$Id$"
00003 */
00004 WIN32 DLL export.
00005 *
00006 Copyright 1998-2010 by Bill Spitzak and others.
00007 *
00008 This library is free software. Distribution and use rights are outlined in
00009 the file "COPYING" which should have been included with this file. If this
00010 file is missing or damaged, see the license at:
00011 *
00012 http://www.fltk.org/COPYING.php

Generated by Doxygen
10.36  Fl_File_Browser.H

// Fl_File_Browser widget.

enum { FILES, DIRECTORIES };

class FL_EXPORT Fl_File_Browser : public Fl_Browser {

int filetype_;  
const char *directory_;  
uchar iconsize_;  
const char *pattern_;  

int full_height() const;
int item_height(void *) const;
int item_width(void *) const;
void item_draw(void *, int, int, int, int) const;
int incr_height() const { return (item_height(0)); }  

public:
  enum { FILES, DIRECTORIES };
```cpp
00001 //
00002 // Fl_File_Chooser dialog for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2015 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 // http://www.fltk.org/COPYING.php
00011 //
00012 // Please report all bugs and problems on the following page:
00013 //
00014 // http://www.fltk.org/str.php
00015 //
00016 // =======================================================================
00017 // DO NOT EDIT FL/Fl_File_Chooser.H and src/Fl_File_Chooser.cxx !!!
00018 //
00019 // Please use fluid to change src/Fl_File_Chooser.fl interactively
00020 // and then use fluid to "write code" or edit and use fluid -c .
00021 //
00022 // =======================================================================
00023 //
00024 //
00025 // generated by Fast Light User Interface Designer (fluid) version 1.0309
00026 //
00027 #ifndef Fl_File_Chooser_H
00028 #define Fl_File_Chooser_H
00029
class FL_EXPORT Fl_File_Chooser {
00030 public:
00031 enum { SINGLE = 0, MULTI = 1, CREATE = 2, DIRECTORY = 4 };   
00032
00033 private:
00034 static Fl_Preferences *prefs_;    
00035 void favoritesButtonCB();
00036
00037 #endif  // !_Fl_File_Chooser_H_  
00038
00039 // End of "$Id$".
00040
00041 //
00042 // $Id$

10.37 Fl_File_Chooser.H

00001 //
00002 // Fl_File_Chooser dialog for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2015 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 // http://www.fltk.org/COPYING.php
00011 //
00012 // Please report all bugs and problems on the following page:
00013 //
00014 // http://www.fltk.org/str.php
00015 //
00016 // =======================================================================
00017 // DO NOT EDIT FL/Fl_File_Chooser.H and src/Fl_File_Chooser.cxx !!!
00018 //
00019 // Please use fluid to change src/Fl_File_Chooser.fl interactively
00020 // and then use fluid to "write code" or edit and use fluid -c .
00021 //
00022 // =======================================================================
00023 //
00024 //
00025 // generated by Fast Light User Interface Designer (fluid) version 1.0309
00026 //
00027 #ifndef Fl_File_Chooser_H
00028 #define Fl_File_Chooser_H
00029
class FL_EXPORT Fl_File_Chooser {
00030 public:
00031 enum { SINGLE = 0, MULTI = 1, CREATE = 2, DIRECTORY = 4 };   
00032
00033 private:
00034 static Fl_Preferences *prefs_;    
00035 void favoritesButtonCB();
00036
00037 #endif  // !_Fl_File_Chooser_H_  
00038
00039 // End of "$Id$".
00040
00041 //
00042 // $Id$
```
void favoritesCB(Fl_Widget *w);
void fileListCB();
void fileNameCB();
void newdir();
static void previewCB(Fl_File_Chooser *fc);
void showChoiceCB();
update_favorites();
update_preview();

public:
F1_File_Chooser(const char *d, const char *p, int t, const char *title);

private:
F1_Double_Window *window;
inline void cb_window_i(Fl_Double_Window*, void*);
static void cb_window(Fl_Double_Window*, void*);
F1_Choice *showChoice;
inline void cb_showChoice_i(Fl_Choice*, void*);
static void cb_showChoice(Fl_Choice*, void*);
F1_Menu_Button *favoritesButton;
inline void cb_favoritesButton_i(Fl_Menu_Button*, void*);
static void cb_favoritesButton(Fl_Menu_Button*, void*);
F1_Button *newButton;
inline void cb_newButton_i(Fl_Button*, void*);
static void cb_newButton(Fl_Button*, void*);
Fl_Tile *tile;
inline void cb_i(Fl_Tile*, void*);
static void cb_(Fl_Tile*, void*);
F1_File_Browser *fileList;
inline void cb_fileList_i(Fl_File_Browser*, void*);
static void cb_fileList(Fl_File_Browser*, void*);
F1_Box *previewBox;

public:
F1_Check_Button *previewButton;

private:
inline void cb_previewButton_i(Fl_Check_Button*, void*);
static void cb_previewButton(Fl_Check_Button*, void*);
F1_Check_Button *showHiddenButton;

private:
inline void cb_showHiddenButton_i(Fl_Check_Button*, void*);
static void cb_showHiddenButton(Fl_Check_Button*, void*);
F1_File_Input *fileName;
inline void cb_fileName_i(Fl_File_Input*, void*);
static void cb_fileName(Fl_File_Input*, void*);
F1_Return_Button *okButton;
inline void cb_okButton_i(Fl_Return_Button*, void*);
static void cb_okButton(Fl_Return_Button*, void*);
F1_Button *cancelButton;
inline void cb_cancelButton_i(Fl_Button*, void*);
static void cb_cancelButton(Fl_Button*, void*);
F1_Double_Window *favWindow;
F1_File_Browser *favList;
inline void cb_favList_i(Fl_File_Browser*, void*);
static void cb_favList(Fl_File_Browser*, void*);
F1_Button *favUpButton;
inline void cb_favUpButton_i(Fl_Button*, void*);
static void cb_favUpButton(Fl_Button*, void*);
F1_Button *favDeleteButton;
inline void cb_favDeleteButton_i(Fl_Button*, void*);
static void cb_favDeleteButton(Fl_Button*, void*);
F1_Button *favDownButton;
inline void cb_favDownButton_i(Fl_Button*, void*);
static void cb_favDownButton(Fl_Button*, void*);
F1_Button *favCancelButton;
inline void cb_favCancelButton_i(Fl_Button*, void*);
static void cb_favCancelButton(Fl_Button*, void*);
F1_Return_Button *favOkButton;
inline void cb_favOkButton_i(Fl_Return_Button*, void*);
static void cb_favOkButton(Fl_Return_Button*, void*);

~F1_File_Chooser();
void callback(void (*cb)(Fl_File_Chooser *, void *), void *d = 0);
void color(Fl_Color c);
Fl_Color color();
int count();
void directory(const char *d);
char * directory();
void filter(const char *p);
const char * filter();
int filter_value();
void filter_value(int f);
void hide();
void iconsize(uchar s);
uchar iconsize();
void label(const char *l);
const char * label();
void ok_label(const char *l);
const char *ok_label();
void preview(int e);
int preview() const { return previewButton->value(); };
private:
void showHidden(int e);
void remove_hidden_files();
public:
void rescan();
void rescan_keep_filename();
void show();
int show();
void texycolor(Fl_Color c);
void textfont(Fl_Font f);
void fontsize(Fl_Fontsize s);
void type(int t);
int type();
const char *user_data() const;
void user_data(void *d);
const char *value(int f = 1);
void value(const char *filename);
int visible();
static const char *add_favorites_label;
static const char *all_files_label;
static const char *custom_filter_label;
static const char *existing_file_label;
static const char *favorites_label;
static const char *filename_label;
static const char *filesystems_label;
static const char *manage_favorites_label;
static const char *new_directory_label;
static const char *new_directory_tooltip;
static const char *preview_label;
static const char *save_label;
static const char *show_label;
static const char *hidden_label;
static Fl_File_Sort_F *sort;
private:
Fl_Widget *ext_group;
public:
Fl_Widget* add_extra(Fl_Widget* gr);
};
FL_EXPORT char *fl_dir_chooser(const char *message,const char *fname,int relative=0);
FL_EXPORT char *fl_file_chooser(const char *message,const char *pat,const char *fname,int relative=0);
FL_EXPORT void fl_file_chooser_callback(void (*cb)(const char*));
FL_EXPORT void fl_file_chooser_ok_label(const char*l);
#endif

// End of "$Id$".

#include "Fl.H"

Generated by Doxygen
// Special color value for the icon color.
#define FL_ICON_COLOR (Fl_Color)0xffffffff

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:

  enum // File types
  {
    ANY, // Any kind of file
    PLAIN, // Only plain files
    FIFO, // Only named pipes
    DEVICE, // Only character and block devices
    LINK, // Only symbolic links
    DIRECTORY // Only directories
  };

  enum // Data opcodes
  {
    END, // End of primitive/icon
    COLOR, // Followed by color value (2 shorts)
    LINE, // Start of line
    CLOSEDLINE, // Start of closed line
    POLYGON, // Start of polygon
    OUTLINEPOLYGON, // Followed by outline color (2 shorts)
    VERTEX // Followed by scaled X,Y
  };

  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);

  void label(Fl_Widget *w);

  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);

  static Fl_File_ICON *find(const char *filename, int filetype = ANY);

  static Fl_File_ICON *first() { return (first_); }

  static void load_system_icons(void);

  Fl_File_ICON *next() { return (next_); }

  const char *pattern() { return (pattern_); }

  int size() { return (num_data_); }

  int type() { return (type_); }

  short *value() { return (data_); }

};
10.39 Fl_File_Input.H

ifndef Fl_File_Input_H
#define Fl_File_Input_H

#include <FL/Fl_Input.H>

class FL_EXPORT Fl_File_Input : public Fl_Input {

Fl_Color errorcolor_;  
char ok_entry_;  
uchar down_box_;  
short buttons_[200];  
short pressed_;  

void draw_buttons();  
int handle_button(int event);  
void update_buttons();  

public:
Fl_File_Input(int X, int Y, int W, int H, const char *L=0);  
virtual int handle(int event);  

protected:
virtual void draw();

public:
Fl_Boxtype down_box() const { return (Fl_Boxtype)down_box_; }  
void down_box(Fl_Boxtype b) { down_box_ = b; }  

Fl_Color errorcolor() const { return errorcolor_; }  
void errorcolor(Fl_Color c) { errorcolor_ = c; }  

int value(const char *str);  
int value(const char *str, int len);  
const char *value() { return Fl_Input_::value(); }  

};
#endif // !Fl_File_Input_H

10.40 Fl_Fill_Dial.H

ifndef Fl_Fill_Dial_H
#define Fl_Fill_Dial_H

Generated by Doxygen
/** \file Fl_Fill_Dial widget. */

#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

*/

#elif define Fl_FILL_Dial_H

#include "Fl_Fill_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

*/

FILED "Fl_Fill_Dial.H"

10.41 Fl_Fill_Slider.H

/** \file Fl_Fill_Slider widget. */

#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

*/

FILED "Fl_Fill_Slider.H"

10.42 Fl_Float_Input.H

/** \file Fl_Float_Input widget. */

#ifndef Fl_Float_Input_H
#define Fl_Float_Input_H

#include "Fl_Float_Input.H"

class FL_EXPORT Fl_Float_Input : public Fl_Float_Input {
    public:
    Fl_Float_Input(int X, int Y, int W, int H, const char *L=0);
}

#endif

*/

FILED "Fl_Float_Input.H"
10.43 Fl_FormsBitmap.H

00011 //
00012 // http://www.fltk.org/COPYING.php
00013 // Please report all bugs and problems on the following page:
00014 // http://www.fltk.org/str.php
00015 //
00016 /*
00017   file
00018   Fl_Float_Input widget . */
00019
00020 ifndef Fl_Float_Input_H
00021 #define Fl_Float_Input_H
00022
00023 #include "Fl_Input.H"
00024
00025 class FL_EXPORT Fl_Float_Input : public Fl_Input {
00026   public:
00027     Fl_Float_Input(int X, int Y, int W, int H, const char *l = 0);
00028     
00029 #endif
00030
00031 /*
00032   file
00033   Fl_Float_Input widget . */
00034
00035 #ifndef Fl_Float_Input_H
00036 #define Fl_Float_Input_H
00037
00038 #include "Fl_Bitmap.H"
00039
00040 class FL_EXPORT Fl_Float_Input : public Fl_Input {
00041   public:
00042     Fl_Float_Input(int X, int Y, int W, int H, const char *l = 0);
00043     
00044 #endif
00045 //
00046 // End of "$Id$".
00047 //

10.43 Fl_FormsBitmap.H

00001 //
00002 // "$Id$"
00003 //
00004 // Forms bitmap header file for the Fast Light Tool Kit (FLTK).
00005 // Copyright 1998-2010 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 // http://www.fltk.org/COPYING.php
00012 //
00013 // Please report all bugs and problems on the following page:
00014 //
00015 //
00016 //
00017 //
00018 //
00019 /*
00020   file
00021   Fl_Float_Input widget . */
00022
00023 ifndef Fl_Float_Input_H
00024 #define Fl_Float_Input_H
00025
00026 #include "Fl_Bitmap.H"
00027
00028 class FL_EXPORT Fl_Float_Input : public Fl_Input {
00029   Fl_Bitmap *b;
00030   protected:
00031     void draw();
00032     public:
00033     Fl_Float_Input(Fl_Boxtype, int, int, int, int, const char * = 0);
00034     void set(int W, int H, const uchar *bits);
00035     void bitmap(Fl_Bitmap *B) {b = B;}
00036     Fl_Bitmap *bitmap() const {return b;}
00037     
00038 #endif
00039 //
00040 // End of "$Id$".
00041 //

10.44 Fl_FormsPixmap.H

00001 //
00002 // "$Id$"
00003 //
00004 // Forms pixmap header file for the Fast Light Tool Kit (FLTK).
00005 // Copyright 1998-2010 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in

Generated by Doxygen
/* file Fl_FormsPixmap widget */

#ifndef Fl_FormsPixmap_H
#define Fl_FormsPixmap_H

#include "Fl_Pixmap.H"

class FL_EXPORT Fl_FormsPixmap : public Fl_Widget {
  Fl_Pixmap *b;
  protected:
    void draw();
  public:
    Fl_FormsPixmap(Fl_Boxtype t, int X, int Y, int W, int H, const char *L=0);
    void set(const char * const * bits);
    Fl_Pixmap *Pixmap() const {return b;}
};
#endif

// End of "$Id$".

10.45 Fl_Free.H

/* 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:
    int handle(int e);
    Fl_Free(uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl);
    ~Fl_Free();
};

Generated by Doxygen
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
#endif

// End of "$Id$".

/* 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);
};
#endif

// End of "$Id$".

/* file Fl_Gl_Window widget. */

#ifndef Fl_Gl_Window_H
#define Fl_Gl_Window_H
#include "Fl_Window.H"

class FL_EXPORT Fl_Gl_Window : public Fl_Window {
public:
  Fl_Gl_Window();
};
#endif

// End of "$Id$".
```cpp
#ifndef GLContext
typedef void* GLContext; // actually a GLXContext or HGLDC
#endif

class Fl_Gl_Choice; // structure to hold result of glXChooseVisual

class FL_EXPORT Fl_Gl_Window : public Fl_Window {
    int mode_; // GLPluginMode
    const int *alist;
    Fl_Gl_Choice *g;
    GLContext context_; // actually a GLXContext or HGLDC
    char valid_f_; // damage() of back buffer
    virtual void draw_overlay();
    void init();
    void *overlay;
    void make_overlay();
    friend class _Fl_Gl_Overlay;
    static int can_do(int, const int *);
    int mode(int, const int *);
    static int gl_plugin_linkage();

public:
    void show();
    void show(int a, char **b) {Fl_Window::show(a,b);}
    void flush();
    void hide();
    void resize(int,int,int,int);
    int handle(int);
    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);}
    Fl_Mode mode() const {return (Fl_Mode)mode_;
    int mode(int a) {return mode(a,0);}
    int mode(const int *a) {return mode(0, a);}
    void * context() const {return context_;
    void context(void *, int destroy_flag = 0);
    void make_current();
    void swap_buffers();
    void ortho();
    int can_do_overlay();
    void redraw_overlay();
    void hide_overlay();
    void make_overlay_current();
    virtual Fl_Gl_Window * as_gl_window() {return this;}
    virtual Fl_Gl_Window(int W, int H, const char *l=0) :
        Fl_Window(W,H,l) {init();}
    Fl_Gl_Window(int X, int Y, int W, int H, const char *l=0) :
        Fl_Window(X,Y,W,H,l) {init();}
    Fl_Gl_Window(int a, int b, int c, int d, int e)
    protected:
    virtual void draw();
};
#endif

// Note: Doxygen docs in Fl_Widget.H to avoid redundancy.
#define __APPLE__

float pixels_per_unit();
#endif
#endif
```
Group 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:

http://www.fltk.org/COPYING.php

Please report all bugs and problems on the following page:

http://www.fltk.org/str.php

file

 ifndef Fl_Group_H
 define Fl_Group_H
 ifndef Fl_Widget_H
 include "Fl_Widget.H"
 endif

class FL_EXPORT Fl_Group : public Fl_Widget {
 Fl_Widget ** array_,
 Fl_Widget * savedfocus_,
 Fl_Widget * resizable_,
 int children_,
 int *sizes_, // remembered initial sizes of children
 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();
 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;
 int *sizes();
 public:
 int handle(int);
 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 *) const;
 int find(const Fl_Widget& o) const {return find(&o);
 Fl_Widget * const* array() const;
 void resize(int,int,int,int);
 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();
 void resizable(Fl_Widget& o) {resizable_ = &o;
 void resizable(Fl_Widget& o) {resizable_ = o;
 void resizable() {const return resizable_;
 void add_resizable(Fl_Widget& o) {resizable_ = &o;
 void add_resizable(Fl_Widget& o) {resizable_ = 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) != 0; }
// Note: Doxygen docs in Fl_Widget.H to avoid redundancy.
virtual Fl_Group* as_group() { return this; }

// back compatibility functions:
void focus(Fl_Widget* W) { W->take_focus(); }

Fl_Widget* & _ddfdesign_kludge() { return resizable_; }

void forms_end();

// dummy class used to end child groups in constructors for complex
// subclasses of Fl_Group:
class FL_EXPORT Fl_End {
public:
  Fl_End() { Fl_Group::current()->end(); }

  // End of "$Id$".
}
inline void cb_larger__i(Fl_Button *, void*);
static void cb_larger_(Fl_Button *, void*);
inline void cb_find__i(Fl_Input *, void*);
static void cb_find_(Fl_Input *, void*);
Fl_Input *find_
inline void cb_view__i(Fl_Help_View *, void*);
static void cb_view_(Fl_Help_View *, void*);

public:
~Fl_Help_Dialog();
int h();
void hide();
void load(const char *f);
void position(int xx, int yy);
void resize(int xx, int yy, int ww, int hh);
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();
};
#endif

//
// End of "$Id$".
//

// Fl_Help_View widget . */
define Fl_Help_View_H
#define Fl_Help_func type - link callback function for files...
```c
struct Fl_Help_Block {
    const char *start, // Start of text
    *end; // End of text
    uchar border; // Draw border?
    Fl_Color bgcolor; // 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
};

struct Fl_Help_Link {
    char filename[192],
    name[32];
    int x,
    y,
    w,
    h;
};

/*
 * Fl_Help_View font stack opaque implementation
 */

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);
    }
    size_t count() const {return nfonts_;} // Gets the current number of fonts in the stack
};

protected:
    size_t nfonts_;
    Fl_Help_Font_Style elts_[100];
};

struct Fl_Help_Target {
    char name[32];
};

class FL_EXPORT Fl_Help_View : public Fl_Group { // Help viewer widget
    enum { RIGHT = -1, CENTER, LEFT };
    char title_[1024];
    Fl_Color defcolor_,
    bgcolor_,
    textcolor_,
    linkcolor_;
    Fl_Font textfont_
    Fl_Fontsize texsize_;
    const char *value_;
};
```
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); }

Fl_Help_Block *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);

void format();
void format_table(int *table_width, int *columns, const char *table);
const char *directory() const { if (directory_[0]) return (directory_); else return ((const char *)0); }
const char *filename() const { if (filename_[0]) return (filename_); else return ((const char *)0); }
int find(const char *s, int p = 0);  
void link(Fl_Help_Func *fn) { link_ = fn; }
int load(const char *f);
void resize(int X, int Y, int W, int H);
int size() const { return (size_); }
void size(int W, int H) { Fl_Widget::size(W, H); }
void textcolor(Fl_Color c) { if (textcolor_ == defcolor_) textcolor_ = c; defcolor_ = c; }
Fl_Color textcolor() const { return (defcolor_); }
void textfont(Fl_Font f) { textfont_ = f; format(); }
Fl_Font textfont() const { return (textfont_); }
void textsize(Fl_Fontsize s) { textsize_ = s; format(); }
Fl_Fontsize textsize() const { return (textsize_); }
const char *title() { return (title_); }
void topline(const char *n);
int topline() const { return (topline_); }
void leftline(int);
int leftline() const { return (leftline_); }
void value(const char *val);
const char *value() const { return (value_); }
clear_selection();
void select_all();
int scrollbar_size() const { return (scrollbar_size_); }
void scrollbar_size(int newSize) { scrollbar_size_ = newSize; }

10.51 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_Hold_Browser.H */

10.52 Fl_Hor_Fill_Slider.H
10.53 Fl_Hor_Nice_Slider.H

/* file Fl_Hor_Nice_Slider widget. */

#ifndef Fl_Hor_Nice_Slider_H
#define Fl_Hor_Nice_Slider_H

#include "Fl_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
  #endif

  // End of "$Id$".
  //

10.54 Fl_Hor_Slider.H

/* file Fl_Hor_Standard_Slider widget. */

#ifndef Fl_Hor_Slider_H
#define Fl_Hor_Slider_H

#include "Fl_Slider.H"

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);

  #endif
  #endif

  // End of "$Id$".
  //

Generated by Doxygen
10.55  Fl_Hor_Value_Slider.H

10.56  Fl_Image.H File Reference

Fl_Image, Fl_RGB_Image classes.
#include "Enumerations.H"
#include <stdlib.h>
Classes

- class Fl_Image
  Base class for image caching and drawing.
- 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.

10.56.1 Detailed Description

Fl_Image, Fl_RGB_Image classes.

10.56.2 Enumeration Type Documentation

10.56.2.1 Fl_RGB_Scaling
eenum Fl_RGB_Scaling
The scaling algorithm to use for RGB images.

Enumarator

<table>
<thead>
<tr>
<th>FL_RGB_SCALING_NEAREST</th>
<th>default RGB image scaling algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_RGB_SCALING_BILINEAR</td>
<td>more accurate, but slower RGB image scaling algorithm</td>
</tr>
</tbody>
</table>

10.57 Fl_Image.H

Go to the documentation of this file.

```c
#endif Fl_Image_H
#include "Enumerations.H"
#include <stdlib.h>

class Fl_Widget;
class Fl_Pixmap;
struct Fl_Menu_Item;
struct Fl_Label;

enum Fl_RGB_Scaling {
    FL_RGB_SCALING_NEAREST = 0,
    FL_RGB_SCALING_BILINEAR
};
```
public:
00058 static const int ERR_NO_IMAGE = -1;
00059 static const int ERR_FILE_ACCESS = -2;
00060 static const int ERR_FORMAT = -3;
00061 private:
00062 int w_, h_, d_, ld_, count_;  
00063 const char * const *data_;  
00064 static Fl_RGB_Scaling RGB_scaling_;  
00065 void draw_empty(int X, int Y);
00066 static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la);
00067 static void measure(const Fl_Label *lo, int &lw, int &lh);
00068 Fl_Image & operator=(const Fl_Image &);
00069 Fl_Image(const Fl_Image &);
00061 protected:
00072 void w(int W) {w_ = W;}
00073 void h(int H) {h_ = H;}
00074 void d(int D) {d_ = D;}
00075 void ld(int LD) {ld_ = LD;}
00076 void data(const char * const *p, int c) {data_ = p; count_ = c;}
00077 public:
00078 int w() const {return w_;
00079 int h() const {return h_;
00080 int d() const {return d_;
00081 int ld() const {return ld_;
00082 int count() const {return count_;
00083 int fail();
00084 Fl_Image(int W, int H, int D);
00085 virtual ~Fl_Image();
00086 virtual Fl_Image *copy(int W, int H);
00087 Fl_Image *copy() { return copy(w(), h()); }
00088 virtual void color_average(Fl_Color c, float i);
00089 virtual void desaturate();
00090 virtual void label(Fl_Widget *w);
00091 virtual void label(Fl_Menu_Item *m);
00092 virtual void draw(int X, int Y, int W, int H, int cx=0, int cy=0);
00093 virtual void uncache();
00094 class FL_EXPORT Fl_RGB_Image : public Fl_Image {
00095 friend class Fl_Quartz_Graphics_Driver;
00096 friend class Fl_GDI_Graphics_Driver;
00097 friend class Fl_GDI_Printer_Graphics_Driver;
00098 friend class Fl_Xlib_Graphics_Driver;
00099 static size_t max_size_;  
00100 const uchar *array;
00101 int alloc_array;
00102 private:
00103 #if defined(__APPLE__) || defined(WIN32)
00104 void *id_; // for internal use
00105 void *mask_; // for internal use (mask bitmap)
00106 #else
00107 unsigned id_; // for internal use
00108 unsigned mask_; // for internal use (mask bitmap)
00109 #endif // __APPLE__ || WIN32  
001010 public:
001011 Fl_RGB_Image(const uchar *bits, int W, int H, int D=3, int LD=0);
001012 Fl_RGB_Image(const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY);
001013 virtual ~Fl_RGB_Image();
001014 virtual void color_average(Fl_Color c, float i);
001015 virtual void desaturate();
001016 virtual void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);} // platform dependent
001017 virtual void uncache();
001018 // set RGB image scaling method
001019 static void RGB_scaling(Fl_RGB_Scaling);
001020 }; // get RGB image scaling method
001021 static Fl_RGB_Scaling RGB_scaling();
001022 class FL_EXPORT Fl_RGBA_Image : public Fl_Image {
001023 friend class Fl_Quartz_Graphics_Driver;
001024 friend class Fl_GDI_Graphics_Driver;
001025 friend class Fl_GDI_Printer_Graphics_Driver;
001026 friend class Fl_Xlib_Graphics_Driver;
001027 static size_t max_size_;  
001028 public:
001029 const uchar *array;
001030 int alloc_array;
001031 private:
001032 #if defined(__APPLE__) || defined(WIN32)
001033 void *id_; // for internal use
001034 void *mask_; // for internal use (mask bitmap)
001035 #else
001036 unsigned id_; // for internal use
001037 unsigned mask_; // for internal use (mask bitmap)
001038 #endif // __APPLE__ || WIN32  
001039 public:
001040 Fl_RGBA_Image(const uchar *bits, int W, int H, int D=4, int LD=0);
001041 Fl_RGBA_Image(const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY);
001042 virtual ~Fl_RGBA_Image();
001043 virtual Fl_Image *copy(int W, int H);
001044 Fl_Image *copy() { return copy(w(), h()); }
001045 virtual void color_average(Fl_Color c, float i);
001046 virtual void desaturate();
001047 virtual void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);} // platform dependent
001048 virtual void uncache();
001049 // Forbid use of copy constructor and assign operator
001050 Fl_Image & operator=(const Fl_Image &);
001051 Fl_Image(const Fl_Image &);
001052 protected:
001053 void w(int W) {w_ = W;}
001054 void h(int H) {h_ = H;}
001055 void d(int D) {d_ = D;}
001056 void ld(int LD) {ld_ = LD;}
001057 void data(const char * const *p, int c) {data_ = p; count_ = c;}
001058 void draw_empty(int X, int Y);
001059 static void labeltype(const Fl_Label *, int lx, int ly, int lw, int lh, Fl_Align);  
001060 static void measure(const Fl_Label *, int &lw, int &lh);  
001061 static Fl_RGB_Scaling RGB_scaling();
001062 static Fl_RGB_Scaling RGB_scaling();
001063 void inactive() { color_average(FL_GRAY, .33f); }
0237 void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}  
0238 virtual void label(Fl_Widget*w);  
0239 virtual void label(Fl_Menu_Item*m);  
0240 virtual void uncache();  
0250 static void max_size(size_t size) { max_size_ = size;}  
0255 static size_t max_size() {return max_size_;}  
0256 };  
0257 #endif  // !Fl_Image_H  
0259  
0260 //  
0261 // End of "$Id$".  
0262 //

10.58 Fl_Image_Surface.H

0001 //  
0002 // "$Id$"  
0003 //  
0004 // Draw-to-image code for the Fast Light Tool Kit (FLTK).  
0005 //  
0006 // Copyright 1998-2014 by Bill Spitzak and others.  
0007 //  
0008 // This library is free software. Distribution and use rights are outlined in  
0009 // the file "COPYING" which should have been included with this file. If this  
0010 // file is missing or damaged, see the license at:  
0011 //  
0012 // http://www.fltk.org/COPYING.php  
0013 //  
0014 // Please report all bugs and problems on the following page:  
0015 //  
0016 // http://www.fltk.org/str.php  
0017 //

0019 #ifndef Fl_Image_Surface_H  
0020 #define Fl_Image_Surface_H  
0021  
0022 #include <FL/Fl_Copy_Surface.H>  
0023 #include <FL/Fl_Image.H>  
0024 #include <FL/Fl_Shared_Image.H>  
0025  
0026 class FL_EXPORT Fl_Image_Surface : public Fl_Surface_Device {  
0027 private:  
0028 void prepare_(int w, int h, int highres);  
0029 Fl_Offscreen offscreen;  
0030 int width;  
0031 int height;  
0032 Fl_Paged_Device *helper;  
0034 #ifdef __APPLE__  
0035 #elif defined(WIN32)  
0036 HDC _sgc;  
0037 Window _sw;  
0038 Fl_Surface_Device *_ss;  
0039 int _savedc;  
0040 #else  
0041 Fl_Surface_Device *previous;  
0042 Window pre_window;  
0043 GC gc;  
0044 #endif  
0045 public:  
0046 static const char *class_id;  
0047 const char *class_name() {return class_id;};  
0048 #if FLTK_ABI_VERSION >= 10104 || defined(FL_DOXYGEN)  
0049 Fl_Image_Surface(int w, int h, int highres = 0);  
0050 #else  
0051 Fl_Image_Surface(int w, int h);  
0052 #endif  
0053 Fl_Image_Surface() {  
0054 Fl_Paged_Device *helper;  
0055 #endif  
0056 #ifdef __APPLE__  
0057 /* Mac class to implement translate()/untranslate() for a flipped bitmap graphics context */  
0058 class FL_EXPORT Fl_Quartz_Flipped_Surface_ : public Fl_Surface_Device {  
0059 private:  
0060 Fl_Quartz_Flipped_Surface_(int w, int h);  
0061 void translate(int x, int y);  
0062 void set_current();  
0063 void draw(Fl_Widget*, int delta_x = 0, int delta_y = 0);  
0064 void draw_decorated_window(Fl_Window* win, int delta_x = 0, int delta_y = 0);  
0065 Fl_RGB_Image *image();  
0066 Fl_Shared_Image *highres_image();  
0067 Fl_Quartz_Flipped_Surface_(int w, int h, int highres = 0);  
0068 Fl_Quartz_Flipped_Surface_(int w, int h);  
0069 #endif  
0070 public:  
0071 Fl_Quartz_Flipped_Surface_();  
0072 Fl_Quartz_Flipped_Surface_();  
0073 Fl_Quartz_Flipped_Surface_();  
0074 Fl_Quartz_Flipped_Surface_();  
0075 Fl_Quartz_Flipped_Surface_();  
0076 Fl_Quartz_Flipped_Surface_();  
0077 Fl_Quartz_Flipped_Surface_();  
0078 Fl_Quartz_Flipped_Surface_();  
0079 Fl_Quartz_Flipped_Surface_();
void untranslate();
virtual ~Fl_Quartz_Flipped_Surface_() {};
#endif

#endif // Fl_Image_Surface_H

//
// End of "$Id$".
//
10.59 Fl_Input.H

#ifndef Fl_Input_H
#define Fl_Input_H

#include "Fl_Input_.H"

class FL_EXPORT Fl_Input : public Fl_Input_ {
    int handle_key();
    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_page_up();
    int kf_page_down();
    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_delete_char();
    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:
    void draw();
public:
    int handle(int);
    Fl_Input(int,int,int,int,const char * = 0);
};
#endif

//
// Generated by Doxygen

void draw();
public:
int handle(int);
Fl_Input(int,int,int,int,const char * = 0);
}
10.60 Fl_Input_.H

10.60 Fl_Input_.H

00001 //
00002 // "FlInput".
00003 //
00004 // Input base class header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2015 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 /*
00019 \file
00020 Fl_Input_ widget . */
00021
00022 ifndef Fl_Input__H
00023 #define Fl_Input__H
00024
00025 ifndef Fl_Widget_H
00026 #include "Fl_Widget.H"
00027 #endif
00028
00029 #define FL_NORMAL_INPUT 0
00030 #define FL_FLOAT_INPUT 1
00031 #define FL_INT_INPUT 2
00032 #define FL_HIDDEN_INPUT 3
00033 #define FL_MULTILINE_INPUT 4
00034 #define FL_SECRET_INPUT 5
00035 #define FL_INPUT_TYPE 7
00036 #define FL_INPUT_READONLY 8
00037 #define FL_NORMAL_OUTPUT (FL_NORMAL_INPUT | FL_INPUT_READONLY)
00038 #define FL_MULTILINE_OUTPUT (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
00039 #define FL_INPUT_WRAP 16
00040 #define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)
00041 #define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)
00042
00043 class FL_EXPORT Fl_Input_ : public Fl_Widget {
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 00086 00087 00088 00089 00090 00091 00092 00093 00094 00095 00096 00097 00098 00099 00100 00101 00102 00103 00104 00105 00106 00107 00108 00109 00110 00111 00112 00113 00114 00115 00116 00117 00118 00119 00120 00121 00122 00123 00124 00125 00126 00127 00128 00129 00130 00131 00132 00133 00134 00135 00136 00137 00138 00139 00140 00141 00142 00143 00144 00145 00146 00147 00148 00149 00150 00151 00152 00153

/*
\file
Fl_Input_ widget . */

ifndef Fl_Input__H
#define Fl_Input__H

ifndef Fl_Widget_H
#include "Fl_Widget.H"
#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_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)

class FL_EXPORT Fl_Input_ : public Fl_Widget {

const char* value_;
char* buffer;
int size_;
int bufsize;
int position_; int mark_; int tab_nav_; int xscroll_, yscroll_; int maximum size_; int shortcut_; uchar erase_cursor_only;
Fl_Font textfont_; Fl_Fontsize textsize_; Fl_Color textcolor_; Fl_Color cursor_color_
static double up_down_pos;
static int was_up_down;

Generated by Doxygen
00154  /* Convert a given text segment into the text that will be rendered on screen. */
00155  const char * expand(const char *, char *) const;
00156
00157  /* Calculates the width in pixels of part of a text buffer. */
00158  double expandpos(const char *, const char *, const char *, int *) const;
00159
00160  /* Mark a range of characters for update. */
00161  void minimal_update(int, int);
00162
00163  /* Mark a range of characters for update. */
00164  void minimal_update(int p);
00165
00166  /* Copy the value from a possibly static entry into the internal buffer. */
00167  void put_in_buffer(int newsize);
00168
00169  /* Set the current font and font size. */
00170  void setfont() const;
00171
00172 protected:
00173
00174  /* Find the start of a word. */
00175  int word_start(int i) const;
00176
00177  /* Find the end of a word. */
00178  int word_end(int i) const;
00179
00180  /* Find the start of a line. */
00181  int line_start(int i) const;
00182
00183  /* Find the end of a line. */
00184  int line_end(int i) const;
00185
00186  /* Draw the text in the passed bounding box. */
00187  void drawtext(int, int, int, int);
00188
00189  /* Move the cursor to the column given by up_down_pos. */
00190  int up_down_position(int, int keepmark=0);
00191
00192  /* Handle mouse clicks and mouse moves. */
00193  void handle_mouse(int, int, int, int, int keepmark=0);
00194
00195  /* Handle all kinds of text field related events. */
00196  int handle_text(int, int, int, int, int);
00197
00198  /* Check the when() field and do a callback if indicated. */
00199  void maybe_do_callback();
00200
00201  int xscroll() const {return xscroll_;
00202
00203  int yscroll() const {return yscroll_;}
00204
00205  int yscroll(int yOffset) { yscroll_ = yOffset; damage(FL_DAMAGE_EXPOSE);}
00206
00207  /* Return the number of lines displayed on a single page. */
00208  int linePerPage();
00209
00210  public:
00211
00212  /* Change the size of the widget. */
00213  void resize(int, int, int, int);
00214
00215  /* Constructor */
00216  FL_Input_(int, int, int, const char * = 0);
00217
00218  /* Destructor */
00219  ~FL_Input_();
00220
00221  /* Changes the widget text. */
00222  int value(const char *);
00223
00224  /* Changes the widget text. */
00225  int value(const char *, int);
00226
00227  /* Changes the widget text. */
00228  int static_value(const char *);
00229
00230  /* Changes the widget text. */
00231  int static_value(const char *, int);
00232
00233  const char * value() const {return value_;
00234
00235  Fl_Char index(int i) const;
00236
00237  int size() const {return size_;
00238
00239  void size(int W, int H) { Fl_Widget::size(W, H); }
```cpp
int maximum_size() const { return maximum_size_; }

void maximum_size(int m) { maximum_size_ = m; }

int position() const { return position_; }

int mark() const { return mark_; }

int position(int p, int m) { return position(position_, position_, p, m); }

int position(int p) { return position(position_, p, 0); }

int position(int a, int b, const char *text, int ilen=0); 

int cut() { return replace(position_, mark_, 0); }

int cut(int n) { return replace(position_, position_+n, 0); }

int cut(int a, int b) { return replace(a, b, 0); }

int insert(const char * t, int l=0) { return replace(position_, mark_, t, l); }

int copy(int clipboard);

int undo();

int replace(int b, int e, const char *text, int ilen=0);

int shortcut() const { return shortcut_; }

void shortcut(int s) { shortcut_ = s; }

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; }

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)); }

int tab_nav() const { return tab_nav_; }

void tab_nav(int val) {
    tab_nav_ = val;
}

int tab_nav() const {
    return tab_nav_; }

#endif
```

Generated by Doxygen
10.61  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>
#include <FL/fl_draw.H>
#include <string.h>

class FL_EXPORT Fl_Input_Choice : public Fl_Group {
  // Private class to handle slightly 'special' behavior of menu button

  // note: this is used by the Fl_Input_Choice ctor defined in Fl_Group.
  static void menu_cb(Fl_Widget*, void *data) {
    Fl_Input_Choice *o=(Fl_Input_Choice *)data;
    Fl_Widget_Tracker wp(o);
    const Fl_Menu_Item *item = o->menubutton()->mvalue();
    if (!item || !item->flags & (FL_SUBMENU|FL_SUBMENU_POINTER)) return; // ignore submenus
    o->do_callback();
    if (!wp.deleted()) return;
    if (o->callback() != default_callback) {
      o->clear_changed();
    }
    if (wp.deleted()) return;
    if (o->when() & FL_WHEN_NOT_CHANGED) o->do_callback();
    if (o->when() & FL_WHEN_CHANGED) o->do_callback();
  }

  // note: this is used by the Fl_Input_Choice ctor defined in Fl_Group.

  Fl_Input *inp_;  // Fl_Input* menu_;
  InputMenuButton *menu_;

public:
  InputMenuButton(int X,int Y,int W,int H,const char *L=0) :
    Fl_Menu_Button(X, Y, W, H, L) { box(FL_UP_BOX); }

  Fl_Input_Choice() {
    inp_ = new Fl_Input();
    menu_ = new InputMenuButton(x(),y(),w(),h(),L);
    box(FL_UP_BOX);
    inp_->value(menu_->text());
    inp_->set_changed();
    this->Fl_Widget::set_changed();
    if (when() & (FL_WHEN_CHANGED|FL_WHEN_RELEASE))
      do_callback();
  }

protected:
  Fl_Menu_Item *item;

private:
  void draw() {
    draw_box(FL_UP_BOX, color());
    fl_color(active_r() ? labelcolor() : fl_inactive(labelcolor()));
    int xc = x()+w()/2, yc=y()+h()/2;
    fl_polygon(xc-5,yc-3,xc+5,yc-3,xc,yc+3);
    if (Fl::focus() == this) draw_focus();
  }
};
```

Generated by Doxygen
static void inp_cb(Fl_Widget*, void *data) {
    Fl_Input_Choice *o=(Fl_Input_Choice *)data;
    Fl_Widget_Tracker wp(o);
    if (o->inp_->changed()) {
        o->Fl_Widget::set_changed();
        if (o->when() & (FL_WHEN_CHANGED|FL_WHEN_RELEASE))
            o->do_callback();
        else {
            o->Fl_Widget::clear_changed();
            if (o->when() & FL_WHEN_NOT_CHANGED)
                o->do_callback();
        }
    } else {
        o->Fl_Widget::clear_changed();
        if (wp.deleted()) return;
    }
    if (o->callback() != default_callback)
        o->Fl_Widget::clear_changed();
}

// Custom resize behavior -- input stretches, menu button doesn't
inline int inp_x() { return(x() + Fl::box_dx(box())); }
inline int inp_y() { return(y() + Fl::box_dy(box())); }
inline int inp_w() { return(w() - Fl::box_dw(box()) - 20); }
inline int inp_h() { return(h() - Fl::box_dh(box())); }
inline int menu_x() { return(x() + w() - 20 - Fl::box_dx(box())); }
inline int menu_y() { return(y() + Fl::box_dy(box())); }
inline int menu_w() { return(20); }
inline int menu_h() { return(h() - Fl::box_dh(box())); }

public:
    Fl_Input_Choice(int X,int Y,int W,int H,const char *L=0);
    void add(const char *s) { menu_->add(s); }
    int changed() const { return inp_->changed() | Fl_Widget::changed(); }
    void clear_changed() {
        inp_->clear_changed();
        Fl_Widget::clear_changed();
    }
    void set_changed() {
        inp_->set_changed();
        // no need to call Fl_Widget::set_changed()
    }
    void clear() { menu_->clear(); }
    void resize(int X, int Y, int W, int H) {
        menu_->resize(menu_x(), menu_y(), menu_w(), menu_h());
        menu_->resize(menu_x(), menu_y(), menu_w(), menu_h());
        // need to call Fl_Widget::resize()
    }
    Fl_Boxtype down_box() const { return (menu_->down_box()); }
    const Fl_Menu_Item *menu() { return (menu_->menu()); }
    void menu(const Fl_Menu_Item *m) { menu_->menu(m); }
    void resize(int X, int Y, int W, int H) {
        inp_->resize(inp_x(), inp_y(), inp_w(), inp_h());
        menu_->resize(menu_x(), menu_y(), menu_w(), menu_h());
    }
    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); }
    void value(const char *val) { inp_->value(val); }
    void value(int val) {
        menu_->value(val);
        inp_->value(menu_->text(val));
    }
    Fl_Menu_Button *menubutton() { return menu_; }
    Fl_Input *input() { return inp_; }
};
10.63 Fl_JPEG_Image.H

10.64 Fl_Light_Button.H
10.65 Fl_Line_Dial.H

```c
#ifndef Fl_Line_Dial_H
#define Fl_Line_Dial_H

#include "Fl_Dial.H"

class FL_EXPORT Fl_Line_Dial : public Fl_Dial {
public:
    Fl_Line_Dial(int X, int Y, int W, int H, const char *L = 0);
};
#endif

// End of "$Id$".
```

10.66 Fl_Menu.H

```c
#ifndef Fl_Menu_H
#define Fl_Menu_H

#include "Fl_Button.H"

class FL_EXPORT Fl_Menu : public Fl_Button {
protected:
    virtual void draw();
public:
    virtual int handle(int);
    Fl_Menu(int x, int y, int w, int h, const char *l = 0);
};
#endif

// End of "$Id$".
```
10.6.7  Fl_Menu_.H

```cpp
class FL_EXPORT Fl_Menu_ : public Fl_Widget {

Fl_Menu_Item *menu_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
uchar alloc; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
uchar down_box_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Font textfont_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Fontsize textsize_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Color textcolor_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)

protected:

uchar alloc; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
uchar down_box_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Font textfont_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Fontsize textsize_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
Fl_Color textcolor_; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)

public:

int item_pathname(char *name, int namelen, const Fl_Menu_Item *finditem) const;
const Fl_Menu_Item * picked(const Fl_Menu_Item *item);
const Fl_Menu_Item * find_item(const char *name); const;
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_;
}
void menu(const Fl_Menu_Item *m);
int insert(int index, const char *, int shortcut, Fl_Callback*, void* = 0, int = 0);
int add(const char *, int shortcut, Fl_Callback*, void* = 0, int = 0); // see src/Fl_Menu_add.cxx
int add(const char* a, const char* b, Fl_Callback* c, void* d = 0, int e = 0) {;
  return add(a,fl_old_shortcut(b),c,d,e);
}

};
```

Generated by Doxygen
```c
int add(const char *);  
int size() const;  
void size(int W, int H) { Fl_Widget::size(W, H); }  
void clear();  
int clear_submenu(int index);  
void replace(int, const char *);  
void remove(int);  
void shortcut(int i, int s) {menu_[i].shortcut(s);};  
void mode(int i, int fl) {menu_[i].flags = fl;};  
int mode(int i) const {return menu_[i].flags;};  
const Fl_Menu_Item *mvalue() const {return 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_Color down_color() const {return selection_color();};
void down_color(unsigned c) {selection_color(c);};  
void setonly(Fl_Menu_Item * item);  
};
```

---

```c
#endif
#endif

// End of "$Id$".

// End of "$Id$".

// $Id$

```

---

### Fl_Menu_Bar.H

```c
 ifndef Fl_Menu_Bar_H
 define Fl_Menu_Bar_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Menu_Bar : public Fl_Menu_ {
 protected:
   void draw();

 public:
   int handle(int);

 Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);

 };  
#endif
```

---

```
*/

// Fl_Menu_Bar widget . */

 ifndef Fl_Menu_Bar_H
 define Fl_Menu_Bar_H

#define Fl_Menu_Bar_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Menu_Bar : public Fl_Menu_ {
 protected:
   void draw();

 public:
   int handle(int);

 Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);

 };  
#endif
```

---

```
Generated by Doxygen
```
10.69 Fl_Menu_Button.H

```c
#ifndef Fl_Menu_Button_H
#define Fl_Menu_Button_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Menu_Button : public Fl_Menu_ {
protected:
    void draw();
public:
    enum popup_buttons {POPUP1 = 1,
                        POPUP2,
                        POPUP12,
                        POPUP3,
                        POPUP13,
                        POPUP23,
                        POPUP123
                    };
    int handle(int);
    const Fl_Menu_Item * popup();
    Fl_Menu_Button(int,int,int,int,const char * =0);
};

#endif
```

10.70 Fl_Menu_Item.H File Reference

```c
#include "Fl_Widget.H"
#include "Fl_Image.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

```c
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
}
```
• enum {
    FL_PUP_NONE = 0 , FL_PUP_GREY = FL_MENU_INACTIVE , FL_PUP_GRAY = FL_MENU_INACTIVE , FL_PUP_BOX = FL_MENU_TOGGLE , FL_PUP_CHECK = FL_MENU_VALUE , FL_PUP_RADIO = FL_MENU_RADIO , FL_PUP_INVISIBLE = FL_MENU_INVISIBLE , FL_PUP_SUBMENU = FL_SUBMENU_POINTER }

Functions

• FL_EXPORT Fl_Shortcut fl_oldShortcut (const char *)

  Emulation of XForms named shortcuts.

10.70.1 Enumeration Type Documentation

10.70.1.1 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>This 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</td>
</tr>
</tbody>
</table>

10.71 Fl_Menu_Item.H

Go to the documentation of this file.

```c
#ifndef Fl_Menu_Item_H
#define Fl_Menu_Item_H

#include "Fl_Widget.H"
#include "Fl_Image.H"

#ifdef __APPLE__ && defined(check)
#undef check
#endif

// doxygen needs the following line to enable e.g.:FL_MENU_TOGGLE to link to the enums
enum { // values for flags:
    FL_MENU_INACTIVE = 1 ,
    FL_MENU_TOGGLE = 2 ,
```

Generated by Doxygen
FL_MENU_VALUE = 4,
FL_MENU_RADIO = 8,
FL_MENU_INVISIBLE = 0x10,
FL_SUBMENU_POINTER = 0x80,
FL_SUBMENU = 0x40,
FL_MENU_DIVIDER = 0x80,
FL_MENU_HORIZONTAL = 0x100;

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);
class Fl_Menu;

struct FL_EXPORT Fl_Menu_Item {
    const char *text;
    int shortcut_;  
    Fl_Callback *callback_;  
    void *user_data_;  
    int flags;
    uchar labeltype_;  
    Fl_Font labelfont_;  
    Fl_Fontsize labelsize_;  
    Fl_Color labelcolor_;  
}

const Fl_Menu_Item *next(int i=1) const {
    return (Fl_Menu_Item *)((const Fl_Menu_Item*)this)->next(i);
}

const Fl_Menu_Item *first() const { return next(0); }
Fl_Menu_Item *first() { return next(0); }

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;}
Fl_Labeltype labeltype() const {return (Fl_Labeltype)labeltype_;

void labeltype(Fl_Labeltype a) {labeltype_ = 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_=c; user_data_=p;}

void callback(Fl_Callback * c) {callback_=c;}
void callback(Fl_Callback0 *c) {callback_=(Fl_Callback*)c;

void callback(Fl_Callback1 *c, long p=0) {callback_=(Fl_Callback*)c;

void argument(long v) {user_data_ = (void *)(fl_intptr_t)v;}

int shortcut() const {return shortcut_;}

int submenu(int s) {shortcut_ = s;}

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);}
void activate() {flags &= ~FL_MENU_INACTIVE;}
void deactivate() {flags |= FL_MENU_INACTIVE;}
int activevisible() const {return !(flags & (FL_MENU_INACTIVE|FL_MENU_INVISIBLE));}

void image(Fl_Image* a) {a->label(this);}
void image(Fl_Image& a) {a.label(this);}

// compatibility for FLUID so it can set the image of a menu item...

int measure(int* h, const Fl_Menu_* a) const;
void draw(int x, int y, int w, int h, const Fl_Menu_*, int t=0) const;

// popup menus without using an Fl_Menu_widget:
void popup(int x, int y, const char* title = 0, const Fl_Menu_Item* picked=0, const Fl_Menu_* a = 0);

int checked() const {return flags&FL_MENU_VALUE;}
void check() {flags |= FL_MENU_VALUE;}
void uncheck() {flags &= ~FL_MENU_VALUE;}
int insert(int,const char*,int,Fl_Callback*,void* =0, int =0);
int add(const char*, int shortcut, Fl_Callback*, void* =0, int = 0);
int add(const char*a, const char* b, Fl_Callback* c,
void* d = 0, int e = 0) {
    return add(a,fl_old_shortcut(b),c,d,e);
}

// back-compatibility, do not use:
int checked() const {return flags&FL_MENU_VALUE;}
void check() {flags |= FL_MENU_VALUE;}
void uncheck() {flags &= ~FL_MENU_VALUE;}
int insert(int,const char*,int,Fl_Callback*,void* =0, int =0);
int add(const char*, int shortcut, Fl_Callback*, void* =0, int = 0);
int add(const char*a, const char* b, Fl_Callback* c,
void* d = 0, int e = 0) {
    return add(a,fl_old_shortcut(b),c,d,e);
}

int size() const;

typedef Fl_Menu_Item Fl_Menu; // back compatibility
enum { // back-compatibility enum:
    FL_PUP_NONE = 0,
    FL_PUP_GREY = FL_MENU_INACTIVE,
    FL_PUP_BOX = FL_MENU_TOGGLE,
    FL_PUP_CHECK = FL_MENU_VALUE,
    FL_PUP_RADIO = FL_MENU_RADIO,
    FL_PUP_SUBMENU = FL_SUBMENU_POINTER
};

} // Fl_Menu_Widget.

// End of "$Id$".

// End of "$Id$".

// Generated by Doxygen

10.72 Fl_Menu_Window.H
Menu window 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:

http://www.fltk.org/COPYING.php

Please report all bugs and problems on the following page:

http://www.fltk.org/str.php

file

Fl_Menu_Window widget . */

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:
  void show();
  void erase();
  void flush();
  void hide();
  unsigned int overlay() {return !(flags()&NO_OVERLAY);}
  void set_overlay() {clear_flag(NO_OVERLAY);}
  void clear_overlay() {set_flag(NO_OVERLAY);}
  ~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

End of "$Id$".

---

Standard message 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:

http://www.fltk.org/COPYING.php

Please report all bugs and problems on the following page:

http://www.fltk.org/str.php

$Id$

---

Multi browser 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:

$Id$
Fl_Multi_Browser widget.

```cpp
#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
```

Fl_Multi_Label widget.

```cpp
#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
```

Fl_Multiline_Input widget.

```cpp
#ifndef Fl_Multiline_Input_H
#define Fl_Multiline_Input_H

class Fl_Widget;
struct Fl_Menu_Item;

struct FL_EXPORT Fl_Multiline_Input {
  const char * labela;
  const char * labelb;
  uchar typea;
  uchar typeb;
  void label(Fl_Widget *);
  void label(Fl_Menu_Item *);
};
#endif
```

---

Generated by Doxygen
/* Fl_Multiline_Input widget. */

#ifndef Fl_Multiline_Input_H
#define Fl_Multiline_Input_H

#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);
};

#endif

// End of "$Id$".

10.77 Fl_Multiline_Output.H

/* Fl_Multiline_Output widget. */

#ifndef Fl_Multiline_Output_H
#define Fl_Multiline_Output_H

#include "Fl_Output.H"

class FL_EXPORT Fl_Multiline_Output : public Fl_Output {
public:
  Fl_Multiline_Output(int X, int Y, int W, int H, const char *l = 0);
};

#endif

// End of "$Id$".

10.78 Fl_Native_File_Chooser.H File Reference

Fl_Native_File_Chooser widget.
#include <FL/Fl_Native_File_Chooser.H>

Classes

• class Fl_FLTK_File_Chooser
• class Fl_GTK_File_Chooser
• class Fl_Native_File_Chooser

This class lets an FLTK application easily and consistently access the operating system’s native file chooser.

Generated by Doxygen
Fl_NativeFileChooser.H

Go to the documentation of this file.

Fl_NativeFileChooser.H

Detailed Description

Fl_NativeFileChooser widget.
int type() const;
void options(int o);
int options() const;
int count() const;
const char *filename() 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() ;

#ifdef WIN32
private:
int _btype; // kind-of browser to show()
int _options; // general options
#ifdef FLTK_ABI_VERSION >= 10304
fl_OPENFILENAMEW *_ofn_ptr; // GetOpenFileName() & GetSaveFileName() struct
fl_BROWSEINFOW *_binf_ptr; // SHBrowseForFolder() struct
WCHAR *_wpattern; // pattern buffer for filter
#else
fl_OPENFILENAMEW _ofn;
fl_BROWSEINFOW _binf;
#endif
char **_pathnames; // array of pathnames
int _tpathnames; // total pathnames
char * _directory; // default pathname to use
char * _title; // title for window
char * _filter; // user-side search filter
char * _parsedfilter; // filter parsed for Windows dialog
int _nfilters; // number of filters parse_filter counted
char * _preset_file; // the file to preselect
char * _errmsg; // error message
#endif
#endif

#ifdef __APPLE__
private:
int _btype; // kind-of browser to show()
int _options; // general options
void *_panel;
char **_pathnames; // array of pathnames
int _tpathnames; // total pathnames
char * _directory; // default pathname to use
char * _title; // title for window
char * _preset_file; // the 'save as' filename
char * _filter; // user-side search filter, eg:
// C Files|txt|.*[ch]|

// filter names (tab delimited)
// eg. "C Files|Text Files"
// array of filter patterns, eg:
// _filt_conn[0]=".*\.cxx\.
// _filt_conn[1]=".*\txt"
char * _filt_patt[MAXFILTERS];
#endif
private:
int _btype; // kind-of browser to show()
int _options; // general options
void * _panel;
char ** _pathnames; // array of pathnames
int _tpathnames; // total pathnames
char * _directory; // default pathname to use
char * _title; // title for window
char * _preset_file; // the 'save as' filename
char * _filter; // user-side search filter, eg:
// C Files|.*[ch]|Text Files|.*.txt
char * _filt_patt[MAXFILTERS];
// array of filter patterns, eg:
// _filt_conn[0]=".*[ch]\.
// _filt_conn[1]=".*\txt"
int _filt_total; // parse_filter() # of filters loaded
int _filt_value; // index of the selected filter
char * _errmsg; // error message

Generated by Doxygen
void errmsg(const char *msg);
void clear_pathnames();
void set_single_pathname(const char *s);
int get_saveas_basename(void);
void clear_filters();
void add_filter(const char *, const char *);
void parse_filter(const char *from);
int post();
int runmodal();
#endif
#if ! defined(__APPLE__) && !defined(WIN32)
private:
#if FLTK_ABI_VERSION <= 10302
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
#endif
static int have_looked_for_GTK_libs;
union {
    Fl_FLTK_File_Chooser *_x11_file_chooser;
    Fl_GTK_File_Chooser *_gtk_file_chooser;
};
#endif

class FL_EXPORT Fl_FLTK_File_Chooser {
friend class Fl_Native_File_Chooser;
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_FLTK_File_Chooser(int val);
virtual ~Fl_FLTK_File_Chooser();
void errmsg(const char *msg);
int type_fl_file(int);
void parse_filter();
int exist_dialog();
Fl_File_Chooser *_file_chooser;
type(int);
type() const;
options(int);
options() const;
count() const;
filename() const;
filename(int i) const;
directory(const char *);
directory() const;
title(const char *);
title() const;
filter() const;
filter(const char *);
filters() const;
filter_value(int i);
filter_value() const;
preset_file(const char *);
preset_file() const;
errmsg() const;
show();
};

class FL_EXPORT Fl_GTK_File_Chooser : public Fl_FLTK_File_Chooser {
friend class Fl_Native_File_Chooser;
private:
typedef struct _GtkWidget GtkWidget;
typedef struct _GtkFileFilterInfo GtkFileFilterInfo;
struct pair {
    Fl_GTK_File_Chooser * running; // the running Fl_GTK_File_Chooser
    const char * filter; // a filter string of the chooser
    pair(Fl_GTK_File_Chooser * c, const char *f) {
};

0224 // Private methods
0225 void errmsg(const char *msg);
0226 void clear_pathnames();
0227 void set_single_pathname(const char *s);
0228 int get_saveas_basename(void);
0229 void clear_filters();
0230 void add_filter(const char *, const char *);
0231 void parse_filter(const char *from);
0232 int post();
0233 int runmodal();
0234 #endif
0235
0236 #if ! defined(__APPLE__) && !defined(WIN32)
0237 private:
0238 #if FLTK_ABI_VERSION <= 10302
0239 int _btype; // kind-of browser to show()
0240 int _options; // general options
0241 int _filters;
0242 char _filter; // user supplied filter
0243 char _parsedfilt; // parsed filter
0244 int _filtvalue; // selected filter
0245 char _preset_file;
0246 char _prevvalue; // Returned filename
0247 char _directory;
0248 char _errmsg; // error message
0249 #endif
0250 static int have_looked_for_GTK_libs;
0251 union {
    Fl_FLTK_File_Chooser *_x11_file_chooser;
    Fl_GTK_File_Chooser *_gtk_file_chooser;
};
0252 Fl_FLTK_File_Chooser *x11_file_chooser;
0253 Fl_GTK_File_Chooser *_gtk_file_chooser;
0254 }
0255 #endif
0256
0257 #if !defined(__APPLE__) && !defined(WIN32)
0258 class FL_EXPORT Fl_FLTK_File_Chooser {
0259 friend class Fl_Native_File_Chooser;
0260 protected:
0261 int _btype; // kind-of browser to show()
0262 int _options; // general options
0263 int _filters;
0264 char _filter; // user supplied filter
0265 char _parsedfilt; // parsed filter
0266 int _filtvalue; // selected filter
0267 char _preset_file;
0268 char _prevvalue; // Returned filename
0269 char _directory;
0270 char _errmsg; // error message
0271 Fl_FLTK_File_Chooser(int val);
0272 virtual ~Fl_FLTK_File_Chooser();
0273 void errmsg(const char *msg);
0274 int type_fl_file(int);
0275 void parse_filter();
0276 int exist_dialog();
0277 Fl_File_Chooser *_file_chooser;
0278 virtual void type(int);
0279 int type() const;
0280 virtual options(int);
0281 int options() const;
0282 virtual int count() const;
0283 virtual const char *filename() const;
0284 virtual const char *filename(int i) const;
0285 virtual const char *directory() const;
0286 virtual void title(const char *);
0287 virtual const char * title() const;
0288 virtual const char * filename() const;
0289 virtual const char * title() const;
0290 virtual const char * filter() const;
0291 virtual void filter(const char *);
0292 int filters() const;
0293 void filter_value(int i);
0294 int filter_value() const;
0295 void preset_file(char *);
0296 virtual const char * preset_file() const;
0297 virtual const char *errmsg() const;
0298 virtual int show();
0299 }
0300
0301
class FL_EXPORT Fl_GTK_File_Chooser : public Fl_FLTK_File_Chooser {
0302 friend class Fl_Native_File_Chooser;
0303 private:
0304 typedef struct _GtkWidget GtkWidget;
0305 typedef struct _GtkFileFilterInfo GtkFileFilterInfo;
0306 struct pair {
    Fl_GTK_File_Chooser * running; // the running Fl_GTK_File_Chooser
    const char * filter; // a filter string of the chooser
    pair(Fl_GTK_File_Chooser * c, const char *f) {
running = c;
filter = strdup(f);
}
pair() {
free((char*)filter);
};
}

GtkWidget *gtkw_ptr; // used to hold a GtkWidget* without pulling GTK into everything...
unsigned gtkw_count; // number of files read back - if any
char *gtkw_filename; // last name we read back
char *gtkw_title; // the title to be applied to the dialog
const char *previous_filter;

int fl_gtk_chooser_wrapper(); // method that wraps the GTK widget
Fl_GTK_File_Chooser(int val);
virtual ~Fl_GTK_File_Chooser();
static int did_find_GTK_libs;
static void probe_for_GTK_libs(void);
virtual void type(int);
virtual int count() const;
virtual const char *filename() const;
virtual const char *filename(int i) const;
virtual void title(const char *);
virtual const char * title() const;
virtual int show();
void changed_output_type(const char *filter);

static int custom_gtk_filter_function(constGtkFileFilterInfo *, Fl_GTK_File_Chooser::pair*);
static void free_pair(pair *p);
10.82 Fl_Output.H

#ifndef Fl_Object
#define Fl_Object Fl_Widget
#endif
#include "Fl_Widget.H"

// End of "$Id$".

10.82 Fl_Output.H

#ifndef Fl_Output_H
#define Fl_Output_H

#include "Fl_Input.H"
class FL_EXPORT Fl_Output : public Fl_Input {
public:
    Fl_Output(int X, int Y, int W, int H, const char *l = 0);
};
#endif

// End of "$Id$".

10.83 Fl_Overlay_Window.H

#ifndef Fl_Overlay_Window_H
#define Fl_Overlay_Window_H
#include "Fl_Input.H"
class FL_EXPORT Fl_Overlay_Window : public Fl_Input {
public:
    Fl_Overlay_Window(int X, int Y, int W, int H, const char *l = 0);
};
#endif

// End of "$Id$".
# Fl_Overlay_Window class

```c
#ifndef Fl_Overlay_Window_H
#define Fl_Overlay_Window_H

#include "Fl_Double_Window.H"

class FL_EXPORT Fl_Overlay_Window : public Fl_Double_Window {
    #ifndef FL_DOXYGEN
    friend class _Fl_Overlay;
    #endif
    protected:
        virtual void draw_overlay() = 0;
    private:
        Fl_Window *overlay_
    public:
        void show();
        void flush();
        void hide();
        void resize(int,int,int,int);
        // -Fl_Overlay_Window();
        int can_do_overlay();
        void redraw_overlay();
    protected:
        Fl_OverlayWindow(int W, int H, const char *l=0);
        Fl_OverlayWindow(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);}
    }
    #endif
    #endif
// End of "$Id$".
```

---

# Fl_Pack.H

```c
#ifndef Fl_Pack_H
#define Fl_Pack_H

#include <FL/Fl_Group.H>

class FL_EXPORT Fl_Pack : public Fl_Group {
    int spacing_

    public:
        enum { // values for type(int)
            VERTICAL = 0,
            HORIZONTAL = 1
            };
    public:
        protected:
        virtual void draw();
    public:
        Fl_Pack(int x,int y,int w ,int h, const char *l = 0);
        int spacing() const {return spacing_;
        void spacing(int i) {spacing_ = i;
        uchar horizontal() const {return type();
```

10.85 Fl_Paged_Device.H File Reference

Declaration of class Fl_Paged_Device.
#include <FL/Fl_Device.H>
#include <FL/Fl_Window.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.

10.85.1 Detailed Description

Declaration of class Fl_Paged_Device.

10.86 Fl_Paged_Device.H

Go to the documentation of this file.
#ifndef Fl_Paged_Device_H
#define Fl_Paged_Device_H

#include <FL/Fl_Device.H>
#include <FL/Fl_Window.H>

#define NO_PAGE_FORMATS 30 /* MSVC6 compilation fix */

class Fl_EXPORT Fl_Paged_Device : public Fl_Surface_Device {
  #ifndef __APPLE__
  friend class Fl_Copy_Surface;
  friend class Fl_Image_Surface;
  void draw_decorated_window(Fl_Window *win, int x_offset, int y_offset, Fl_Surface_Device *toset);
  #endif
  public:
  enum Page_Format {
    A0 = 0,
    A1,
    A2,
    A3,
    ...
# Fl_Pixmap.H

Generated by Doxygen
Fl_Pixmap widget. */

#ifndef Fl_Pixmap_H
#define Fl_Pixmap_H

#include "Fl_Image.H"

#ifdef WIN32
#include "x.H"
#endif

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_Quartz_Graphics_Driver;
friend class Fl_GDI_Graphics_Driver;
friend class Fl_GDI_Printer_Graphics_Driver;
friend class Fl_Xlib_Graphics_Driver;
void copy_data();
void delete_data();
void set_data(const char * const *);
int prepare(int XP, int YP, int WP, int HP, int &cx, int &cy,
int &X, int &Y, int &W, int &H);

protected:
void measure();

public:
int alloc_data; // Non-zero if data was allocated

private:
#if defined(WIN32)
#if FLTK_ABI_VERSION < 10301
static // a static member is needed for ABI compatibility
#endif
UINT pixmap_bg_color; // RGB color used for pixmap background
#endif // WIN32

void *id_; // for internal use
void *mask_; // for internal use (mask bitmap)
#else
unsigned id_; // for internal use
unsigned mask_; // for internal use (mask bitmap)
#endif // __APPLE__ || WIN32

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();}

virtual ~Fl_Pixmap();
Fl_Image *copy(int W, int H);
Fl_Image *copy() { return copy(w(), h()); };

virtual void color_average(Fl_Color c, float i);

virtual void desaturate();

virtual void draw(int X, int Y, int W, int H, int cx=0, int cy=0);

virtual void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }

virtual void label(Fl_Widget *w);
virtual void label(Fl_Menu_Item *m);
virtual void uncache();
};

#endif

@

---

10.88  Fl_Plugin.H

ifndef Fl_Plugin_H
#define Fl_Plugin_H

#include "Fl_Preferences.H"

class FL_EXPORT Fl_Plugin {
 Fl_Preferences::ID id;
 public:
 Fl_Plugin(const char *klass, const char *name);
 virtual ~Fl_Plugin();

 Fl_Plugin_Manager(const char *klass);
 ~Fl_Plugin_Manager();

 int plugins() { 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 *filepath, const char *pattern=0);
}

#endif // !Fl_Preferences_H

---

10.89  Fl_PNG_Image.H

ifndef Fl_PNG_Image_H
#define Fl_PNG_Image_H

PNG image 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:

Generated by Doxygen
```
/*
file Fl_PNM_Image class . */

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H
#include "Fl_Image.H"

class FL_EXPORT Fl_PNM_Image : public Fl_RGB_Image {
 public:
 Fl_PNM_Image(const char * filename);
 Fl_PNM_Image (const char *name_png, const unsigned char *buffer, int datasize);
 private:
 void load_png_(const char *name_png, const unsigned char *buffer_png, int datasize);

#endif

// End of "$Id$".
```

```
10.90 Fl_PNM_Image.H

```

```
10.91 Fl_Positioner.H

```

Generated by Doxygen
class Fl_Positioner : public Fl_Widget {

  double xmin, ymin;
  double xmax, ymax;
  double xvalue_, yvalue_;
  double xstep_, ystep_;

protected:

  void draw(int, int, int, int);
  int handle(int, int, int, int, int);
  void draw();

public:

  int handle(int);

  Fl_Positioner(int x, int y, int w, int h, const char *l=0);

  double xvalue() const {return xvalue_;

  double yvalue() const {return yvalue_;

  int xvalue(double);

  int yvalue(double);

  int value(double,double);

  void xbounds(double, double);

  double xminint() const {return xmin;

  void xminimum(double a) {xbounds(a, xmax);

  double xmaxint() const {return xmax;

  void xmaximum(double a) {xbounds(xmin, a);

  void yminimum(double a) {ybounds(a, ymax);

  double ymaximum() const {return ymaximum;

  void ymaximum(double a) {ybounds(ymin, a);

  void xsstep(double a) {xstep_ = a;

  void ystep(double a) {ystep_ = a;


  #endif

};

#endif

};

#endif

// End of "$Id$".

10.92 Fl_PostScript.H File Reference

#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>

Classes

- class Fl_PostScript_File_Device
  
  To send graphical output to a PostScript file.

- class Fl_PostScript_Graphics_Driver
  
  PostScript graphical backend.
### Typedefs
- typedef int() Fl_PostScript_Close_Command(FILE *)

### 10.92.1 Detailed Description


### 10.93 Fl_PostScript.H

Go to the documentation of this file.

```c
#ifndef Fl_PostScript_H
#define Fl_PostScript_H

#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>

/* Signature of Fl_PostScript::close_command() functions passed as parameters. */
extern "C" {
  typedef int (Fl_PostScript_Close_Command)(FILE *
}

class FL_EXPORT Fl_PostScript_Graphics_Driver : public Fl_Graphics_Driver {
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_PostScript_Graphics_Driver();
  #ifndef FL_DOXYGEN
  enum SHAPE{NONE=0, LINE, LOOP, POLYGON, POINTS};
  
  class Clip {  
    public:
      int x, y, w, h;
      Clip *prev;  
    };  
  Clip *clip_; 
  int gap_;  
  int pages_; 
  double width_;  
  double height_;  
  int shape_;  
  int linewidth_;  
  int linestyle_;  
  unsigned char cr_,cg_,cb_;  
  void concat();  
  void reconcat();  
  void recover();  
  void reset();

Generated by Doxygen
```
uchar * mask;
int mx; // width of mask;
int my; // mask lines
// Fl_Color bg_;
Fl_PostScript_Close_Command * close_cmd_; 
int page_policy_; 
nPages;
int orientation_; 
float scale_x;
float scale_y;
float angle;
int left_margin;
int top_margin;
FILE *output;
double px_, ph_; 
uchar bg_r, bg_g, bg_b;

int start_postscript (int pagecount, enum Fl_Paged_Device::Page_Format format, enum Fl_Paged_Device::Page_Layout layout);
/* int alpha_mask(const uchar * data, int w, int h, int D, int LD=0);
void transformed_draw(const char * s, int n, double x, double y); // precise text placing
void transformed_draw(const char * s, double x, double y);
int alpha_mask(const uchar * data, int w, int h, int D, int LD=0);

enum Fl_Paged_Device::Page_Format page_format_; 
char *ps_filename_;

void page_policy(int p);
int page_policy(){return page_policy_;};
void close_command(Fl_PostScript_Close_Command * cmd){close_cmd_=cmd;};
FILE * file() {return output;};
//void orientation (int o);
//Fl_PostScript_Graphics_Driver(FILE *o, int lang_level, int pages = 0); // ps (also multi-page)
//Fl_PostScript_Graphics_Driver(FILE *o, int lang_level, int x, int y, int w, int h); //eps
constructor
void interpolate(int i){interpolate_=i;};
int interpolate(){return interpolate_} ;
void page(double pw, double ph, int media = 0);
void page (int format);
#endif // FL_DOXYGEN
// implementation of drawing methods
void color(Fl_Color c);
void color(uchar r, uchar g, uchar b);
void color(uchar x, uchar y, int w, int h);
int clip_box(int x, int y, int w, int h, int 6x, int 6y, int 6w, int 6h);
int not_clipped(int x, int y, int w, int h);
void push_no_clip();
void push_clip(int x, int y, int w, int h);
int clip_box(int x, int y, int w, int h, int 6x, int 6y, int 6w, int 6h);
int not_clipped(int x, int y, int w, int h);
void push_no_clip();
void pop_clip();
void line_style(int style, int width=0, char * dashes=0);
void rect(int x, int y, int w, int h);
void rectf(int x, int y, int w, int h);
void xylene(int x, int y, int x1);
void xyline(int x, int y, int x2, int y2);
void xyline(int x, int y, int x2, int y2, int x3);
void yxline(int x, int y, int y1);
void yxline(int x, int y, int y2, int x2);
void yxline(int x, int y, int y2, int x2, int x3);
void xyline(int x, int y, int y2, int x2, int y3);
void line(int x1, int y1, int x2, int y2);
void line(int x1, int y1, int x2, int y2, int x3, int y3);
void loop(int x0, int y0, int x1, int y1, int x2, int y2);
void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
void point(int x, int y);
void begin_points();
void begin_line();
void begin_loop();
void begin_polygon();
void vertex(double x, double y);
void curve(double x, double y, double x1, double y1, double x2, double y2, double x3, double y3);
void circle(double x, double y, double r);
void arc(double x, double y, double r, double start, double a);
void arc(int x, int y, int w, int h, double a1, double a2);
void pie(int x, int y, int w, int h, double a1, double a2);
void end_points();
void end_line();
void end_loop();
void end_polygon();
void begin_complex_polygon(){begin_polygon();};
void gap(){gap_=1;};
void end_complex_polygon(){end_polygon();};
void transformed_vertex(double x, double y);

void draw_image(const uchar * d, int x,int y,int w,int h, int delta=3, int ldelta=0);
void draw_image_mono(const uchar * d, int x,int y,int w,int h, int delta=1, int id=0);
void draw_image(Fl_Draw_Image_Cb call, void* data, int x,int y, int w, int h, int delta=3);
void draw_image_mono(Fl_Draw_Image_Cb call, void* data, int x,int y, int w, int h, int delta=1);

void draw(const char * s, int nBytes, int x, int y) {transformed_draw(s,nBytes,x,y); }; #ifdef __APPLE__
void draw(const char * s, int nBytes, float x, float y) {transformed_draw(s,nBytes,x,y); }; #endif
draw(int angle, const char *str, int n, int x, int y);
void rtl_draw(const char * s, int n, int x, int y);
void font(int face, int size);
double width(const char *, int);
double width(unsigned int u);
void text_extents(const char *c, int n, int &dx, int &dy, int &w, int &h);
int height();
double height();
double descent();
int descent();
void draw(Fl_Pixmap * pxm,int XP, int YP, int WP, int HP, int cx, int cy);
void draw(Fl_Bitmap * bitmap,int XP, int YP, int WP, int HP, int cx, int cy);
void draw(Fl_RGB_Image * rgb,int XP, int YP, int WP, int HP, int cx, int cy);
void draw_scaled(Fl_Image *img, int XP, int YP, int WP, int HP);

int draw_scaled(Fl_Image *img, int XP, int YP, int WP, int HP);
clocale_printf(const char *format, ...);
~Fl_PostScript_Graphics_Driver();

class FL_EXPORT Fl_PostScript_File_Device : public Fl_Paged_Device {
static const char *class_id;
const char *class_name() {return class_id;};
class Fl_PostScript_File_Device();
int start_job(int pagecount, int * from, int* to);
int start_job(int pagecount, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);
int start_page (void);
int printable_rect(int *w, int *h);
void margins(int *left, int *top, int *right, int *bottom);
void origin(int *x, int *y);
void origin(int x, int y);
void scale (float scale_x, float_y = 0.);
void rotate(float angle);
void translate(int x, int y);
void untranslate(void);
int end_page (void);
void end_job(void);

static const char *file_chooser_title;
/**
 * Fl_Preferences class.
 */

#ifndef Fl_Preferences_H
#define Fl_Preferences_H

#include <stdio.h>
#include "Fl_Export.H"

class FL_EXPORT Fl_PREFERENCES {
public:
    enum Root {
        SYSTEM=0,
        USER
    };

typedef void *ID;

static const char *newUUID();

static char remove(ID id_); 

const char *name(); 
const char *path(); 
int groups(); 
const char *group(int num_group); 
char groupExists(const char *key); 
char deleteGroup(const char *group); 
char deleteAllGroups();

int entries(); 
const char *entry(int index); 
char entryExists(const char *key); 
char deleteEntry(const char *entry); 
char deleteAllEntries();

char clear();

int size(const char *entry); 

char getUserdataPath(char *path, int pathlen);

virtual ~Fl_PREFERENCES();

ID id() { return (ID)node; }

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 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); 

void flush();

};
class FL_EXPORT Name {
    char *data_;    // char export( const char *filename, Type fileFormat );
    char import( const char *filename );

    public:
    Name( unsigned int n );
    Name( const char *format, ... );
    
    operator const char *() { return data_; }
    ~Name();
}

struct Entry {
    char *name, *value;
}

private:
Fl_Preferences() : node(0), rootNode(0) { }
Fl_Preferences &operator=(const Fl_Preferences&);

static char nameBuffer[128];
static char uuidBuffer[40];
static Fl_Preferences *runtimePrefs;

public: // older Sun compilers need this (public definition of the following classes)
    class RootNode;

    class FL_EXPORT Node { // a node contains a list to all its entries
        // and all means to manage the tree structure
        Node *child_, *next_; // these two are mutually exclusive
        Node *parent_; // top_ bit clear
        RootNode *root_; // top_ bit set
        char *path_; // path
        Entry *entry_; // and methods
        int nEntry_, NEntry_; // indexing routines
        Node *index_; // these two are mutually exclusive
        int nIndex_, NIndex_; // indexing routines
        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 );
        Node *addChild( const char *path );
        void setParent( Node *parent );
        Node *parent() { return top_?0L:parent_; }
        void setRoot(RootNode *r) { root_ = r; top_ = 1; }
        RootNode *findRoot();
        char remove();
        char dirty();
        void deleteAllChildren();
        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 );
        int getEntry( const char *name );
        char deleteEntry( const char *name );
        void deleteAllEntries();
        int nEntry();
        Entry *entry(int i) { return entry_[i]; }
    }
    friend class Node;

    class FL_EXPORT RootNode { // the root node manages file paths and basic reading and
        // writing
    }

Generated by Doxygen
```cpp
char *filename_;   
char *vendor_, *application_; 

public:
RootNode( Fl_Preferences *, Root root, const char *vendor, const char *application );
RootNode( Fl_Preferences *, const char *path, const char *vendor, const char *application );
RootNode( Fl_Preferences * );
~RootNode();
int read();
int write();
char getPath( char *path, int pathlen );
};

friend class RootNode;

protected:
Node *node;
RootNode *rootNode;

#endif // !Fl_Preferences_H

// End of "$Id$".

10.95 Fl_Printer.H File Reference

declaration of classes Fl_Printer, Fl_System_Printer and Fl_PostScript_Printer.
#include <FL/x.H>
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Pixmap.H>
#include <FL/fl_RGB_Image.H>
#include <FL/Fl_Bitmap.H>
#include <stdio.h>
#include <FL/Fl_PostScript.H>

Classes

- class Fl_PostScript_Printer
  Print support under Unix/Linux.
- class Fl_Printer
  OS-independent print support.
- class Fl_System_Printer
  Print support under MSWindows and Mac OS.

10.95.1 Detailed Description

declaration of classes Fl_Printer, Fl_System_Printer and Fl_PostScript_Printer.

10.96 Fl_Printer.H

Go to the documentation of this file.
```

Generated by Doxygen
```c
#ifndef Fl_Printer_H
#define Fl_Printer_H

#include <FL/x.H>
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Pixmap.H>
#include <FL/Fl_RGB_Image.H>
#include <FL/Fl_Bitmap.H>
#include <stdio.h>
#if !(defined(__APPLE__) || defined(WIN32))
#include <FL/Fl_PostScript.H>
#elif defined(WIN32)
#include <commdlg.h>
#endif

#endif

class Fl_System_Printer : public Fl_Paged_Device {
friend class Fl_Printer;
private:
  void *gc;
  void set_current(void);
#endif __APPLE__
  float scale_x;
  float scale_y;
  float angle; // rotation angle in radians
  PMPrintSession printSession;
  PMPageFormat pageFormat;
  PMPrintSettings printSettings;
#elif defined(WIN32)
  int abortPrint;
  PRINTDLG pd;
  HDC hDC;
  int prerr;
  int left_margin;
  int top_margin;
  void absolute_printable_rect(int *x, int *y, int *w, int *h);
#endif __APPLE__
  void margins(int *left, int *top, int *right, int *bottom);
  void origin(int *x, int *y);
  void origin(int x, int y);
  void scale(float scale_x, float scale_y = 0.);
  void rotate(float angle);
  void translate(int x, int y);
  void untranslate(void);
  void untranslate(void);
  void end_page(void);
  void end_job(void);
#endif __APPLE__
  void print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x, int delta_y);

public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_System_Printer(void);
protected:
  Fl_System_Printer(void);
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_Printer(void);
  int start_job(int pagecount, int *frompage = NULL, int *topage = NULL);
  int start_page(void);
  int printable_rect(int *w, int *h);
  void margins(int *left, int *top, int *right, int *bottom);
  void origin(int x, int y);
  void scale(float scale_x, float scale_y = 0.);
  void rotate(float angle);
  void translate(int x, int y);
  void untranslate(void);
  void end_page(void);
  void end_job(void);
#ifdef __APPLE__
  void print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x, int delta_y);
#endif __APPLE__
  void print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x, int delta_y);

protected:
  Fl_System_Printer(void);
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_PostScript_Printer(void);
protected:
  Fl_PostScript_Printer(void);
public:
  static const char *class_id;
  const char *class_name() {return class_id;};
  Fl_PostScript_Printer(void);
  int start_job(int pages, int *firstpage = NULL, int *lastpage = NULL);
  }; // class Fl_System_Printer
  };

  class FL_EXPORT Fl_Printer : public Fl_Paged_Device {
  public:
    static const char *class_id;
    const char *class_name() {return class_id;};
    Fl_Printer(void);
    int start_job(int pagecount, int *frompage = NULL, int *topage = NULL);
    int start_page(void);
    int printable_rect(int *w, int *h);
    void margins(int *left, int *top, int *right, int *bottom);
```
00186 void origin(int *x, int *y);
00187 void origin(int x, int y);
00188 void scale(float scale_x, float scale_y = 0.);
00189 void rotate(float angle);
00190 void translate(int x, int y);
00191 void untranslate(void);
00192 int end_page (void);
00193 void end_job (void);
00194 void print_widget(Fl_Widget * widget, int delta_x=0, int delta_y=0);
00195 void print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0);
00196 void set_current(void);
00197 Fl_Graphics_Driver* driver(void);
00199 ~Fl_Printer(void);
00200
00204 static const char *dialog_title;
00205 static const char *dialog_printer;
00206 static const char *dialog_range;
00207 static const char *dialog_copy;
00208 static const char *dialog_all;
00209 static const char *dialog_pages;
00210 static const char *dialog_from;
00211 static const char *dialog_to;
00212 static const char *dialog_properties;
00213 static const char *dialog_copyNo;
00214 static const char *dialog_cancel_button;
00215 static const char *dialog_print_to_file;
00217 static const char *property_title;
00218 static const char *property_pagesize;
00219 static const char *property_mode;
00220 static const char *property_use;
00221 static const char *property_save;
00222 static const char *property_cancel;
00224 private:
00225 #if defined(WIN32) || defined(__APPLE__)
00226 Fl_System_Printer *printer;
00227 #else
00228 Fl_PostScript_Printer *printer;
00229 #endif
00230 }
00231
00232 //
00233 // End of "$Id$"
00234 //
00236

10.97 Fl_Progress.H

00001 //
00002 /* $Id$ */
00003 //
00004 // Progress bar widget definitions.
00005 //
00006 // Copyright 2000-2010 by Michael Sweet.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 /* \file
00020 Fl_Progress widget . */
00021 #ifndef _FL_PROGRESS_H_
00022 # define _FL_PROGRESS_H_
00023 # define _FL_PROGRESS_H_
00024 //
00025 // Include necessary headers.
00026 //
00027 //
00028 #include "FL_Widget.H"
00029 class FL_EXPORT Fl_Progress : public Fl_Widget {
float value_,
minimum_,
maximum_;

protected:

virtual void draw();

public:

Fl_Progress(int x, int y, int w, int h, const char *l = 0);

void maximum(float v) { maximum_ = v; redraw(); }
float maximum() const { return (maximum_); }

void minimum(float v) { minimum_ = v; redraw(); }
float minimum() const { return (minimum_); }

void value(float v) { value_ = v; redraw(); }
float value() const { return (value_); }

};

#endif // !_Fl_Progress_H_

Fl_Radio_Button.H

#include "Fl_Button.H"

class FL_EXPORT Fl_Radio_Button : public Fl_Button {

public:

Fl_Radio_Button(int X,int Y,int W,int H,const char *L=0);

}

#endif

Fl_Radio_Light_Button.H

#include "Fl_Button.H"

class FL_EXPORT Fl_Radio_Light_Button : public Fl_Button {

public:

Fl_Radio_Light_Button(int X,int Y,int W,int H, const char *L=0);

};

#endif

Fl_Radio_Light_Button.H
10.100  Fl_Radio_Round_Button.H

10.101  Fl_Repeat_Button.H
# Fl_Return_Button.H

```c
#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();
public:
  int handle(int);
};
#endif
```

---

# Fl_RGB_Image.H

```c
#ifndef Fl_RGB_Image_H
#define Fl_RGB_Image_H
#include "Fl_Button.H"

class FL_EXPORT Fl_RGB_Image : public Fl_Button {
protected:
  void draw();
public:
  Fl_RGB_Image(int X, int Y, int W, int H);
};
#endif
```

---

Generated by Doxygen
10.104 Fl_Roller.H

```c
#ifndef Fl_Roller_H
#define Fl_Roller_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Roller : public Fl_Valuator {
    protected:
        void draw();
    public:
        int handle(int);
        Fl_Roller(int X,int Y,int W,int H,const char * L=0);
};
#endif
```

10.105 Fl_Round_Button.H

```c
#ifndef Fl_Round_Button_H
#define Fl_Round_Button_H

class FL_EXPORT Fl_Round_Button : public Fl_Valuator {
    protected:
        void draw();
    public:
        int handle();
        Fl_Round_Button(int X,int Y,int W,int H, const char * L=0);
    }
#endif
```

*Generated by Doxygen*
10.106 Fl_Round_Clock.H

00001 //
00002 /* $Id$ */
00003 //
00004 // Round clock header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2010 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 /*
00020 File Fl_Round_Clock widget */
00021 */
00022 ifndef Fl_Round_Clock_H
00023 #define Fl_Round_Clock_H
00024
00025 #include "Fl_Clock.H"
00026
00027 class FL_EXPORT Fl_Round_Clock : public Fl_Clock {
00028
00029 public:
00030 Fl_Round_Clock(int X,int Y,int W,int H, const char *L = 0);
00031
00032 #endif
00033
00034 // End of "$Id$".
00035
00036 //

10.107 Fl_Scroll.H

00001 //
00002 // Scroll header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2021 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 File Fl_Scroll widget */
00019 */
00020 ifndef Fl_Scroll_H
00021 #define Fl_Scroll_H
00022
00023 #include "Fl_Group.H"
00024
00025 #include "Fl_Scrollbar.H"
00026
00027 class FL_EXPORT Fl_Scroll : public Fl_Group {
00028
00029  int xposition_, yposition_;
00030
00031 // End of "$Id$".
int oldx, oldy;
int scrollbar_size_;  
static void hscrollbar_cb(Fl_Widget*, void*);
static void vscrollbar_cb(Fl_Widget*, void*);
void fix_scrollbar_order();
static void draw_clip(void*,int,int,int,int);

#if FLTK_ABI_VERSION >= 10303
protected:  // NEW (STR#1895)
#else
private:  // OLD
#endif

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);

protected:
void bbox(int&,int&,int&,int&);
void draw();

public:
Fl_Scrollbar scrollbar;
Fl_Scrollbar hscrollbar;
void resize(int X, int Y, int W, int H);
int handle(int);
Fl_Scroll(int X,int Y,int W,int H,const char *l=0);
enum { // values for type()
HORIZONTAL = 1,
VERTICAL = 2,
BOTH = 3,
ALWAYS_ON = 4,
HORIZONTAL_ALWAYS = 5,
VERTICAL_ALWAYS = 6,
BOTH_ALWAYS = 7
};

int xposition() const {return xposition_;}  
int yposition() const {return yposition_;}  
void scroll_to(int, int);
void clear();
int scrollbar_size() const {
return(scrollbar_size_);
return (scrollbar_size_ = newSize);
if ( newSize != scrollbar_size_ ) redraw();
scrollbar_size_ = newSize;
}

};

#endif
10.108 Fl_Scrollbar.H

```
/*
** Fl_Scrollbar widget . */

#ifndef Fl_Scrollbar_H
#define Fl_Scrollbar_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Scrollbar : public Fl_Slider {

  int linesize_;  
  int pushed_;    

  static void timeout_cb(void *);
  void increment_cb();

protected:
  void draw();

public:

  Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);
  ~Fl_Scrollbar();
  int handle(int);

  int value() const {return int(Fl_Slider::value());}
  int value(int p) {return int(Fl_Slider::value((double)p));}
  int value(int pos, int windowSize, int first, int total) {
    return scrollvalue(pos, windowSize, first, total);
  }
  int linesize() const {return linesize_;}
  void linesize(int i) {linesize_ = i;}

};

#endif
```

10.109 Fl_Secret_Input.H

```
/*
** Secret input header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2011 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
*/
```
# Fl_Secret_Input widget

```c
#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);
    }
#endif
```

---

## 10.110 Fl_Select_Browser.H

```c
#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
```

---

## 10.111 Fl_Shared_Image.H File Reference

The `Fl_Shared_Image` class.

```c
#include "Fl_Image.H"
```

### Classes
- `class Fl_Shared_Image`
  
  This class supports caching, loading, scaling, and drawing of image files.

### Typedefs
- `typedef Fl_Image *(FL_SHARED_HANDLER) (const char *name, uchar *header, int headerlen)`
Functions

- FL_EXPORT void fl_register_images()
  
  Register the image formats.

10.111.1 Detailed Description

**Fl_Shared_Image** class.

10.111.2 Function Documentation

10.111.2.1 fl_register_images()

```c
FL_EXPORT void fl_register_images () [extern]
```

Register the image formats.

This function is provided in the fltk_images library and registers all of the "extra" image file formats that are not part of the core FLTK library.

10.112 Fl_Shared_Image.H

Go to the documentation of this file.

```
#ifndef Fl_Shared_Image_H
#define Fl_Shared_Image_H

#include "Fl_Image.H"

// Test function for adding new formats
typedef Fl_Image *(*Fl_Shared_Handler)(const char *name, uchar *header,
                                     int headerlen);

// Shared images class.
class FL_EXPORT Fl_Shared_Image : public Fl_Image {
private:
  static Fl_RGB_Scaling scaling_algorithm_; // method used to rescale RGB source images
  static Fl_Shared_Handler *handlers_; // Additional format handlers
public:
  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);
};
```

Generated by Doxygen
// Use get() and release() to load/delete images in memory...

Fl_Shared_Image();
Fl_Shared_Image(const char *n, Fl_Image *img = 0);

virtual ~Fl_Shared_Image();
void add();
void update();

public:

const char *name() { return name_; }
int refcount() { return refcount_; }
int original() { return original_; }

void release();
void reload();

virtual Fl_Image *copy(int W, int H);
Fl_Image *copy() { return copy(w(), h()); }

virtual void color_average(Fl_Color c, float i);
virtual void desaturate();

virtual void draw(int X, int Y, int W, int H, int cx, int cy);
void draw(int X, int Y) { draw(X, Y, W(), H(), 0, 0); }
void scale(int width, int height, int proportional = 1, int can_expand = 0);

virtual void uncache();

static Fl_Shared_Image *find(const char *name, int W = 0, int H = 0); 
static Fl_Shared_Image *get(const char *name, int W = 0, int H = 0);
static Fl_Shared_Image *get(Fl_RGB_Image *rgb, int own_it = 1);
static Fl_Shared_Image **images();
static int num_images();
static void add_handler(Fl_Shared_Handler f);
static void remove_handler(Fl_Shared_Handler f);
static void scaling_algorithm(Fl_RGB_Scaling algorithm) {scaling_algorithm_ = algorithm; }


10.113  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_EXPORT Fl_Color fl_show_colormap (Fl_Color oldcol)

  Pops up a window to let the user pick a colormap entry.

10.113.1  Detailed Description

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

10.114  fl_show_colormap.H

Go to the documentation of this file.
#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

// End of "$Id$".

#include "fl_ask.H"

// End of "$Id$".

/* file Fl_Simple_Counter widget. */

#ifndef Fl_Simple_Counter_H
#define Fl_Simple_Counter_H

#include "Fl_Counter.H"

class FL_EXPORT Fl_Simple_Counter : public Fl_Counter {
public:
    Fl_Simple_Counter(int X,int Y,int W,int H, const char *L = 0);
};

Generated by Doxygen
# Fl_Single_Window.H

class FL_EXPORT Fl_Single_Window : public Fl_Window {
    public:
    void show();
    void show(int a, char **b) {Fl_Window::show(a,b);}
    void flush();
    Fl_Single_Window(int W, int H, const char *l=0);
    Fl_Single_Window(int X, int Y, int W, int H, const char *l=0);
    int make_current();
};

# Fl_Slider.H

class FL_EXPORT Fl_Slider : public Fl_Valuator {
    public:
    void show();
    void show(int a, char **b) {Fl_Valuator::show(a,b);}
    void flush();
    Fl_Slider(int X, int Y, int W, int H, const char *l=0);
    int make_current();
};
// values for type(), lowest bit indicate horizontal:
#define FL_VERT_SLIDER 0
#define FL_HOR_SLIDER 1
#define FL_VERT_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, int);  
  void draw();  

public:  
  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, const char *L);  
    int scrollvalue(int pos,int size,int first,int total);  
    void bounds(double a, double b);  
    float slider_size() const {return slider_size_;}  
    void slider_size(double v);  
    Fl_Boxtype slider() const {return (Fl_Boxtype)slider_;}  
    void slider(Fl_Boxtype c) {slider_ = c;}  
    };  

#endif
} // End of "$Id$.

// Spinner widget 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:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems on the following page:  
// http://www.fltk.org/str.php

// Include necessary headers...  

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
  
  #if FLTK_ABI_VERSION >= 10301
  // NEW
  protected:
  #endif
  Fl_Input input_; // Input field for the value
  Fl_Repeat_Button up_button_, // Up button
  down_button_; // Down button
  
  private:
  static void sb_cb(Fl_Widget *w, Fl_Spinner *sb) {
    double v; // New value
    if (w == &(sb->input_)) {
      // Something changed in the input field...
      v = atof(sb->input_.value());
      if (v < sb->minimum_) {
        sb->value_ = sb->minimum_;
        sb->update();
      } else if (v > sb->maximum_) {
        sb->value_ = sb->maximum_;
        sb->update();
      } else sb->value_ = v;
      sb->set_changed();
      sb->do_callback();
    } else if (w == &(sb->up_button_)) {
      // Up button pressed...
      v = sb->value_ + sb->step_;
      if (v > sb->maximum_) sb->value_ = sb->minimum_;
      else sb->value_ = v;
      sb->update();
    } else if (w == &(sb->down_button_)) {
      // Down button pressed...
      v = sb->value_ - sb->step_;
      if (v < sb->minimum_) sb->value_ = sb->maximum_;
      else sb->value_ = v;
      sb->update();
    }
  }
  
  public:
  Fl_Spinner(int X, int Y, int W, int H, const char *L = 0);
  
  const char *format() { return (format_); }
  void format(const char *f) { format_ = f; update(); }
  
  int handle(int event) {
    switch (event) {
    case FL_KEYDOWN :
    case FL_SHORTCUT :
      if (Fl::event_key() == FL_Up) {
        up_button_.do_callback();
        return 1;
      }
    default: break;
    }
    return 0;
  }
```c
00139     } else if (Fl::event_key() == FL_Down) {
00140         down_button_.do_callback();
00141         return 1;
00142     } else return 0;
00143
00144     case FL_FOCUS :
00145         if (input_.take_focus()) return 1;
00146         else return 0;
00147     }
00148
00149     return Fl_Group::handle(event);
00150 }
00151
00152 double maximum() const { return (maximum_); }
00153 double maximum() const { return (maximum_); }
00154 void maximum(double m) { maximum_ = m; }
00155 double minimum() const { return (minimum_); }
00156 double minimum() const { return (minimum_); }
00157 void minimum(double m) { minimum_ = m; }
00158 void range(double a, double b) { minimum_ = a; maximum_ = b; }
00159 void resize(int X, int Y, int W, int H) {
00160     Fl_Group::resize(X, Y, W, H);
00161     input_.resize(X, Y, W - H / 2 - 2, H);
00162     up_button_.resize(X + W - H / 2 - 2, Y, H / 2 + 2, H / 2);
00163     down_button_.resize(X + W - H / 2 - 2, Y + H - H / 2,
00164                           H / 2 + 2, H / 2);
00165 }
00166
double step() const { return (step_); }
00167 void step(double s) { step_ = s; }
00168
double minimum() const { return (minimum_); }
00169 void minimum(double m) { minimum_ = m; }
00170
double maximum() const { return (maximum_); }
00171 void maximum(double m) { maximum_ = m; }
00172
00173 double maximum() const { return (maximum_); }
00174 double maximum() const { return (maximum_); }
00175 void maximum(double m) { maximum_ = m; }
00176 double minimum() const { return (minimum_); }
00177 void minimum(double m) { minimum_ = m; }
00178
double step() const { return (step_); }
00179 void step(double s) { step_ = s; }
00180
double minimum() const { return (minimum_); }
00181 void minimum(double m) { minimum_ = m; }
00182
00183 double maximum() const { return (maximum_); }
00184 void maximum(double m) { maximum_ = m; }
00185 void resize(int X, int Y, int W, int H) {
00186     Fl_Group::resize(X, Y, W, H);
00187     input_.resize(X, Y, W - H / 2 - 2, H);
00188     up_button_.resize(X + W - H / 2 - 2, Y, H / 2 + 2, H / 2);
00189     down_button_.resize(X + W - H / 2 - 2, Y + H - H / 2,
00190                           H / 2 + 2, H / 2);
00191 }
00192
00193 double maximum() const { return (maximum_); }
00194 void maximum(double m) { maximum_ = m; }
00195
double minimum() const { return (minimum_); }
00196 void minimum(double m) { minimum_ = m; }
00197
00198 double step() const { return (step_); }
00199 void step(double s) { step_ = s; }
00200
00201 double maximum() const { return (maximum_); }
00202 void maximum(double m) { maximum_ = m; }
00203 double minimum() const { return (minimum_); }
00204 void minimum(double m) { minimum_ = m; }
00205
00206 double maximum() const { return (maximum_); }
00207 void maximum(double m) { maximum_ = m; }
00208
double minimum() const { return (minimum_); }
00209 void minimum(double m) { minimum_ = m; }
00210
00211 uchar type() const { return (input_.type()); }
00212 void type(uchar v) {
00213     if (v==FL_FLOAT_INPUT) {
00214         format("%. *f");
00215     } else {
00216         format("%.0f");
00217     }
00218     input_.type(v);
00219 }
00220
00221 double step() const { return (step_); }
00222 void step(double s) { step_ = s; }
00223
double value() const { return (value_); }
00224 void value(double v) { value_ = v; update(); }
00225
00226 Fl_Color textcolor() const {
00227     return (input_.textcolor());
00228 }
00229 void textcolor(Fl_Color c) {
00230     input_.textcolor(c);
00231 }
00232
00233 Fl_Font textfont() const {
00234     return (input_.textfont());
00235 }
00236 void textfont(Fl_Font f) {
00237     input_.textfont(f);
00238 }
00239
00240 Fl_Fontsize textsize() const {
00241     return (input_.textsize());
00242 }
00243 void textsize(Fl_Fontsize s) {
00244     input_.textsize(s);
00245 }
00246
00247 uchar type() const { return (input_.type()); }
00248 void type(uchar v) {
00249     if (v==FL_FLOAT_INPUT) {
00250         format("%. *f");
00251     } else {
00252         format("%.0f");
00253     }
00254     input_.type(v);
00255 }
00256
00257 #endif // FL_Spinner_H
00258
00259 // End of "$Id$".
00260
00261 //
00262 // Copyright 1998-2010 by Bill Spitzak and others.
00263 //
00264
```

#ifndef Fl_Sys_Menu_Bar_H
#define Fl_Sys_Menu_Bar_H

#include "Fl_Menu_Bar.H"
#include "x.H"

#if defined(__APPLE__) || defined(FL_DOXYGEN)

class FL_EXPORT Fl_Sys_Menu_Bar : public Fl_Menu_Bar {

#if FLTK_ABI_VERSION >= 10304
    // NEW -- update() public (STR#3317)
    public:
        void update();
#else
    // OLD -- update() protected
    protected:
        void update();
#endif

    public:
        Fl_Sys_Menu_Bar(int x,int y,int w,int h,const char *l=0);
        ~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) {
            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 global() {};
        void mode (int i, int fl) { Fl_Menu::::mode(i, fl);
            update();
        }
        int mode(int i) const { return Fl_Menu::::mode(i); }
        void shortcut (int i, int s) { Fl_Menu::::shortcut(i, s); update(); }
        void setonly (Fl_Menu_Item *item) { Fl_Menu::::setonly(item); update(); }

    }
#else
    // NEW -- small class for update()
    class FL_EXPORT Fl_Sys_Menu_Bar : public Fl_Menu_Bar {
        public:
            Fl_Sys_Menu_Bar(int x,int y,int w,int h,const char *l=0) : Fl_Menu::::menu() {
                return Fl_Menu::::menu();
            }
            inline void update() {} 
    }
#else
    // OLD -- simple typedef
    typedef Fl_Menu_Bar Fl_Sys_Menu_Bar;
#endif

#endif // defined(__APPLE__) || defined(FL_DOXYGEN)

#endif // Fl_Sys_Menu_Bar_H

.Generated by Doxygen
#ifndef _FL_TABLE_H
#define _FL_TABLE_H

#include <sys/types.h>
#include <string.h> // memcpy
#ifdef WIN32
#include <malloc.h> // WINDOWS: malloc_REALLOC
#else /*WIN32*/
#include <stdlib.h> // UNIX: malloc_REALLOC
#endif /*WIN32*/

#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scroll.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Scrollbar.H>

class FL_EXPORT Fl_Table : public Fl_Group {
public:
    enum TableContext {
        CONTEXTNONE = 0,
        CONTEXTSTARTPAGE = 0x01,
        CONTEXTENDPAGE = 0x02,
        CONTEXTROWHEADER = 0x04,
        CONTEXTCOLHEADER = 0x08,
        CONTEXTCELL = 0x10,
        CONTEXTTABLE = 0x20,
        CONTEXTRCRESIZE = 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;

    #if FLTK_ABI_VERSION >= 10301
    int _scrollbar_size;
    #endif

    #if FLTK_ABI_VERSION >= 10303
    enum {
        TABCELLNAV = 1«0,
    };
    unsigned int flags_;
    #endif

    int _auto_drag;
    int _selecting;
    if FLTK_ABI_VERSION >= 10301
    int _scrollbar_size;
    #endif
    #if FLTK_ABI_VERSION >= 10303
    enum {
        TABCELLNAV = 1«0,
        #endif
    ];
    unsigned int flags_;
    #endif

    class FL_EXPORT IntVector {
int *arr;
unsigned int _size;
void init() {
    arr = NULL;
    _size = 0;
}
void copy(int *newarr, unsigned int newsize) {
    memcpy(arr, newarr, newsize * sizeof(int));
}

public:

-IntVector() {
    init();
} // CTOR
-IntVector( ) { _size = 0; } // DTOR
IntVector(IntVector& o) { copy(o.arr, o._size); } // COPY CTOR
IntVector& operator=(IntVector& o) { // ASSIGN
    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() { return(_size); }
void size(unsigned int count) {
    if ( count != _size ) {
        arr = (int *)realloc(arr, count * sizeof(int));
        _size = 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() { return(arr[_size-1]); }


IntVector _colwidths; // column widths in pixels
IntVector _rowheights; // row heights in pixels

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);

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, table_h; // table's virtual size (in pixels)
int toprow, bottom, leftcol, rightcol; // four corners of viewable table

// selection
int current_row, current_col;
int select_row, select_col;

// OPTIMIZATION: Precomputed scroll positions for the toprow/leftcol
int toprow_scrollpos;
int leftcol_scrollpos;

// Dimensions
int tix, tiy, tiw, tih; // data table inner dimension xywh
int tox, toy, tow, toh; // data table outer dimension xywh
int wix, wiw, wiw, wh; // widget inner dimension xywh

Fl_Scroll *table; // container for child fltk widgets (if any)
10.121 Fl_Table.H 1745

00311 Fl_Scrollbar *hscrollbar; // horizontal scrollbar
00312
00313 // Fltk
00314 int handle(int e); // fltk handle() override
00315
00316 // Class maintenance
00317 void recalc_dimensions();
00318 void table_resized(); // table resized; recalc
00319 void table_scrolled(); // table scrolled; recalc
00320 void get_bounds(TableContext context, // return x/y/w/h bounds for context
00321 int X, int Y, int W, int H);
00322 void change_cursor(Fl_Cursor newcursor); // change mouse cursor to some other shape
00323 TableContext cursor2rowcol(int &R, int &C, ResizeFlag &resizeflag);
00324 // find r/c given current x/y event
00325 int find_cell(TableContext context, // find cell's x/y/w/h given r/c
00326 int R, int C, int X, int Y, int W, int H);
00327 int row_col_clamp(TableContext context, int &R, int &C);
00328 // clamp r/c to known universe
00329
00330 virtual void draw_cell(TableContext context, int R=0, int C=0,
00331 int X=0, int Y=0, int W=0, int H=0) {
00332  // overridden by deriving class
00333 }
00334
00335 long row_scroll_position(int row); // find scroll position of row (in pixels)
00336 long col_scroll_position(int col); // find scroll position of col (in pixels)
00337
00338 int is_fltk_container() { // does table contain fltk widgets?
00339  return( Fl_Group::children() > 3 ); // (ie. more than box and 2 scrollbars?)
00340 }
00341
00342 static void scroll_cb(Fl_Widget *,void*); // h/v scrollbar callback
00343
00344 void damage_zone(int r1, int c1, int r2, int c2, int r3 = 0, int c3 = 0); // damage(FL_DAMAGE_CHILD);
00345
00346 void redraw_range(int topRow, int botRow, int leftCol, int rightCol) {
00347  if (topRow == -1) {
00348    // Initialize redraw range
00349    _redraw_toprow = topRow;
00350    _redraw_botrow = botRow;
00351    _redraw_leftcol = leftCol;
00352    _redraw_rightcol = rightCol;
00353  } else {
00354    // Extend redraw range
00355    if (topRow < _redraw_toprow) _redraw_toprow = topRow;
00356    if (botRow > _redraw_botrow) _redraw_botrow = botRow;
00357    if (leftCol < _redraw_leftcol) _redraw_leftcol = leftCol;
00358    if (rightCol > _redraw_rightcol) _redraw_rightcol = rightCol;
00359  }
00360  // Indicate partial redraw needed of some cells
00361  damage(FL_DAMAGE_CHILD);
00362 }
00363
00364 public:
00365  Fl_Table(int X, int Y, int W, int H, const char *l=0);
00366  ~Fl_Table();
00367
00368  virtual void clear() { rows(0); cols(0); table->clear(); } // todo: add topline(), middleline(), bottomline()
00369
00370  Fl_Boxtype table_box(Fl_Widget*,void*); // h/v scrollbar callback
00371
00372  inline void visible_cells(int& r1, int& r2, int& c1, int& c2) {
00373    r1 = toprow;
00374    r2 = botrow;
00375    c1 = leftcol;
00376    c2 = rightcol;
00377  }
00570  c2 = rightcol;
00571 }
00572
00577  int is_interactive_resize() {
00578    return(_resizing_row != -1 || _resizing_col != -1);
00579 }
00580
00584  inline int row_resize() {
00585    return(_row_resize);
00586 }
00587
00594  inline int row_resize() {
00595    _row_resize = flag;
00596  }
00597
00601  void row_resize(int flag) { // enable row resizing
00602    _row_resize = flag;
00603  }
00604
00610  void col_resize(int flag) { // enable col resizing
00611    _col_resize = flag;
00612  }
00613
00617  inline int col_resize_min() { // column minimum resizing width
00618    return(_col_resize_min);
00619 }
00620
00626  void col_resize_min(int val) {
00627    _col_resize_min = (val < 1) ? 1 : val;
00628 }
00629
00633  inline int row_resize_min() { // column minimum resizing width
00634    return(_row_resize_min);
00635 }
00636
00642  void row_resize_min(int val) {
00643    _row_resize_min = (val < 1) ? 1 : val;
00644 }
00645
00649  inline int row_header() { // set/get row header enable flag
00650    return(_row_header);
00651 }
00652
00657  void row_header(int flag) {
00658    _row_header = flag;
00659    table_resized();
00660    redraw();
00661 }
00662
00666  inline int col_header() { // set/get col header enable flag
00667    return(_col_header);
00668 }
00669
00674  void col_header(int flag) {
00675    _col_header = flag;
00676    table_resized();
00677    redraw();
00678 }
00679
00683  inline void col_header_height(int height) { // set/get col header height
00684    _col_header_h = height;
00685    table_resized();
00686    redraw();
00687 }
00688
00692  inline int col_header_height() { // set/get col header enable flag
00693    return(_col_header_h);
00694 }
00695
00699  inline void row_header_width(int width) { // set/get row header width
00700    _row_header_w = width;
00701    table_resized();
00702    redraw();
00703 }
00704
00708  inline int row_header_width() {
00709    return(_row_header_w);
00710 }
00711
00715  inline void row_header_color(Fl_Color val) { // set/get row header color
00716    _row_header_color = val;
00717    redraw();
00718 }
00719
00723  inline Fl_Color row_header_color() { // set/get row header enable flag
00724    return(_row_header_color);
00725 }
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/get row height

inline int row_height(int row) {
    return((row<0 || row>=(_rowheights.size()) ? 0 : _rowheights[row]));
}

void col_width(int col, int width); // set/get a column's width

inline int col_width(int col) {
    return((col<0 || col>=(_colwidths.size()) ? 0 : _colwidths[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);

int move_cursor(int R, int C, int shiftselect);

int move_cursor(int R, int C);

void resize(int X, int Y, int W, int H); // fltk resize() override

void draw(void); // fltk draw() override

void box(Fl_Boxtype val) {
    Fl_Group::box(val);
}

int row_position(); // current row position

int col_position(); // current col position

inline void top_row(int row) { // set/get top row (deprecated)
    row_position(row);
}

inline int top_row() {
    return(row_position());
}

void init_sizes() {
    table->init_sizes();
    table->redraw();
}

void add(Fl_Widget& wgt) {
    table->add(wgt);
    if ( table->children() > 2 ) {
        table->show();
    } else {
        table->hide();
    }
}

void add(Fl_Widget * wgt) {
    add( *wgt);
}

// Child group
void init_sizes() {
    table->init_sizes();
    table->redraw();
}

void add(Fl_Widget& wgt) {
    table->add(wgt);
}

if ( table->children() > 2 ) {
    table->show();
} else {
    table->hide();
}

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_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();
}

#if FLTK_ABI_VERSION >= 10301
    int scrollbar_size() const {
        return(_scrollbar_size);
    }
    void scrollbar_size(int newSize) {
        if (_scrollbar_size != newSize) redraw();
        _scrollbar_size = newSize;
    }
    #endif

#if FLTK_ABI_VERSION >= 10303
    void tab_cell_nav(int val) {
        if (val) flags_ |= TABCELLNAV;
        else flags_ &= ~TABCELLNAV;
    }
    #endif

Generated by Doxygen
int tab_cell_nav() const {
    return(flags_ & TABCELLNAV ? 1 : 0);
}
#endif

#endif /*_FL_TABLE_H*/

//
// End of "$Id$".
//
// Fl_Table_Row.H

#ifndef _FL_TABLE_ROW_H
#define _FL_TABLE_ROW_H

// Fl_Table_Row -- A row oriented table widget
// A class specializing in a table of rows.
// Handles row-specific selection behavior.

// Copyright 2002 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:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems to "erco at seriss dot com".

#include "Fl_Table.H"

class FL_EXPORT Fl_Table_Row : public Fl_Table {
public:
    enum TableRowSelectMode {
        SELECT_NONE,  // no selection allowed
        SELECT_SINGLE, // single row selection
        SELECT_MULTI  // multiple row selection (default)
    };

private:
    // An STL-ish vector without templates
    class FL_EXPORT CharVector {
        char *arr;
        int _size;
        void init() {
            arr = NULL;
            _size = 0;
        }
        void copy(char *newarr, int newsize) {
            size(newsize);
            memcpy(arr, newarr, newsize * sizeof(char));
        }
        public:
        CharVector() { // CTOR
            init();
        }
        ~CharVector() { // DTOR
            if ( arr ) free(arr);
            arr = NULL;
        }
        CharVector(CharVector&o) { // COPY CTOR
            init();
            copy(o.arr, o._size);
        }
        CharVector& operator=(CharVector&o) { // ASSIGN
            init();
            copy(o.arr, o._size);
            return(*this);
        }
        char operator[](int x) const {
            return(arr[x]);
        }
        char* operator[](int x) {
            return(arr[x]);
        }
        int size() const {
            return(_size);
        }
    };
};
void size(int count) {
  if (count != _size) {
    arr = (char*)realloc(arr, count * sizeof(char));
    _size = count;
  }
}

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

// handle() state variables.
// Put here instead of local statics in handle(), so more
// than one instance can exist without crosstalk between.
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);
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) {
  int r, int c, int WX, int WY, int WX, int WY; // Fl_Table::find_cell()
  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;
}

~Fl_Table_Row() { }

void rows(int val); // set number of rows
int rows() { // get number of rows
  return(Fl_Table::rows());
}

void type(TableRowSelectMode val); // set selection mode
TableRowSelectMode type() const { // get selection mode
  return(_selectmode);
}

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() { // implies clearing selection
  rows[0];
  cols[0];
  Fl_Table::clear(); // clear the table
}

};
#endif /*_FL_TABLE_ROW_H*/

// End of "$Id$".
10.123 Fl_Tabs.H

```c
/* file
   Fl_Tabs widget */

#ifndef Fl_Tabs_H
#define Fl_Tabs_H

#include "Fl_Group.H"

class FL_EXPORT Fl_Tabs : public Fl_Group {
#if FLTK_ABI_VERSION >= 10304
// NEW (nothing)
#else
// OLD (maintained for ABI compat)
#endif

int *tab_pos; // array of x-offsets of tabs per child + 1
int *tab_width; // array of widths of tabs per child + 1
int tab_count; // array size
int tab_positions(); // allocate and calculate tab positions
void clear_tab_positions();
int tab_height();
void draw_tab(int x1, int x2, int W, int H, Fl_Widget *o, int sel=0);
protected:
void redraw_tabs();
void draw();

public:
int handle(int);
Fl_Widget *value();
int value(Fl_Widget *);
Fl_Widget *push() const {return push_};
int push(Fl_Widget *);
Fl_Tabs(int,int,int,int,const char * = 0);
Fl_Widget *which(int event_x, int event_y);
~Fl_Tabs();
void client_area(int &rx, int &ry, int &rw, int &rh, int tabh=0);
};
#endif

// End of "$Id$".
```

10.124 Fl_Text_Buffer.H

```c
/* file
   Fl_Text_Buffer widget */

#ifndef Fl_Text_Buffer_H
#define Fl_Text_Buffer_H

#include "Fl_Group.H"

class FL_EXPORT Fl_Text_Buffer : public Fl_Group {
#if FLTK_ABI_VERSION >= 10304
// NEW (nothing)
#else
// OLD (maintained for ABI compat)
#endif

int *cursor_pos; // array of cursor positions per child + 1
int *cursor_width; // array of widths of cursor per child + 1
int cursor_count; // array size
int cursor_positions(); // allocate and calculate cursor positions
void clear_cursor_positions();
int cursor_height();
void draw_cursor(int x, int W, int H, Fl_Widget *o, int sel=0);
protected:
void redraw_cursor();
void draw();

public:
int handle(int);
Fl_Widget *value();
int value(Fl_Widget *);
Fl_Widget *cur() const {return cur_};
int cur(Fl_Widget *);
Fl_Text_Buffer(int,int,int,int,const char * = 0);
Fl_Widget *which(int event_x, int event_y);
~Fl_Text_Buffer();
void client_area(int &rx, int &ry, int &rw, int &rh, int tabh=0);
};
#endif

// End of "$Id$".
```
/* File Documentation */

#ifndef FL_TEXT_BUFFER_H
#define FL_TEXT_BUFFER_H

#endif

#ifndef ASSERT_UTF8
#define IS_UTF8_ALIGNED(a) if (a && *a) assert(fl_utf8len(*(a))>0);
#define IS_UTF8_ALIGNED2(a, b) if (b>=0 && b<a->length()) assert(fl_utf8len(a->byte_at(b))>0);
#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

#include "Fl_Export.H"

class FL_EXPORT Fl_Text_Selection {
friend class Fl_Text_Buffer;
public:
    void set(int start, int end);
    void update(int pos, int nDeleted, int nInserted);
    int start() const { return mStart; }
    int end() const { return mEnd; }
    bool selected() const { return mSelected; }
    void selected(bool b) { mSelected = b; }
    int includes(int pos) const;
    int position(int * start, int* end) const;
protected:
    int mStart;
    int mEnd;
    bool mSelected;
};

typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,
int nRestyled, const char * deletedText,
void* cbArg);

typedef void (*Fl_Text_Predelete_Cb)(int pos, int nDeleted, void* cbArg);

class FL_EXPORT Fl_Text_Buffer {
public:
    Fl_Text_Buffer(int requestedSize = 0, int preferredGapSize = 1024);
int length() const { return mLength; }
char* text() const;
char* text_range(int start, int end) const;
unsigned int char_at(int pos) const;
char byte_at(int pos) const;
const char* address(int pos) const
{ return (pos < mGapStart) ? mBuf+pos : mBuf+pos+mGapEnd-mGapStart; }
char* address(int pos)
{ return (pos < mGapStart) ? mBuf+pos : mBuf+pos+mGapEnd-mGapStart; }
void insert(int pos, const char* text);
void append(const char* t) { insert(length(), t); }
void remove(int start, int end);
void replace(int start, int end, const char* text);
void copy(Fl_Text_Buffer* fromBuf, int fromStart, int fromEnd, int toPos);
int undo(int* cp=0);
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);
char* 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 predelCB, 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 countDisplayedCharacters(int lineStartPos, int targetPos) const;
int skipDisplayedCharacters(int lineStartPos, int nChars);
int countLines(int startPos, int endPos) const;
int skipLines(int startPos, int nLines);
int rewindLines(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; }
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;

const Fl_Text_Selection* primary_selection() { return &mPrimary; }
const Fl_Text_Selection* secondary_selection() { return &mSecondary; }
const Fl_Text_Selection* highlight_selection() { 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*);

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);
void remove_(int start, int end);
void reallocate_with_gap(int newGapStart, int newGapLen);
char* selection_text_(Fl_Text_Selection* sel) const;

void move_gap(int pos);
void reallocate_with_gap(int newGapStart, int newGapLen);
char* selection_text_(Fl_Text_Selection* sel) const;

void move_gap(int pos);
void reallocate_with_gap(int newGapStart, int newGapLen);
void* selection_text_(Fl_Text_Selection* sel) const;

void move_gap(int pos);
void reallocate_with_gap(int newGapStart, int newGapLen);

void move_gap(int pos);
void reallocate_with_gap(int newGapStart, int newGapLen);
void* selection_text_(Fl_Text_Selection* sel) const;

void move_gap(int pos);
void reallocate_with_gap(int newGapStart, int newGapLen);
void* selection_text_(Fl_Text_Selection* sel) const;
void replace_selection_(Fl_Text_Selection * sel, const char* text);
void update_selections(int pos, int nDeleted, int nInserted);

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;
int mModifyProcs;
Fl_Text_Modify_Cb *mModifyProcs;
void * mModifyProcArg;
int mPredeleteProcs;
Fl_Text_Predelete_Cb *mPredeleteProcs;
void * mPredeleteProcArg;
int mCursorPosHint;
char mCanUndo;
int mPreferredGapSize;

};

// End of "$Id$".

/*
file
Fl_Text_Display widget . */

class FL_EXPORT Fl_Text_Display: public Fl_Group {

public:

enum {
  NORMAL_CURSOR,
  CARET_CURSOR,
  DIM_CURSOR,
  BLOCK_CURSOR,
  HEAVY_CURSOR,
  SIMPLE_CURSOR
};

enum {
  CURSOR_POS,
  CHARACTER_POS
};

enum {
  FILE
};

FILE *fl_text_display_widget . */

endif

// End of "$Id$".

// Copyright 2001-2016 by Bill Spitzak and others.
// Original code Copyright Mark Edel. Permission to distribute under
// the LGPL for the FLTK library granted by Mark Edel.
// 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

Generated by Doxygen
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 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_Text_Display(int X, int Y, int W, int H, const char *l = 0);

virtual int handle(int e);

void buffer(Fl_Text_Buffer * buf);

void buffer(Fl_Text_Buffer& buf) { buffer(&buf); }

Fl_Text_Buffer * buffer() const { return mBuffer; }

void redisplay_range(int start, int end);

void scroll(int toplineNum, int hozOffset);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

int move_right();

int move_left();

int move_up();

int move_down();

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);

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);

int next_word(void);

int previous_word(void);
void *cbArg);

int position_style(int lineStartPos, int lineLen, int lineIndex) const;

int shortcut() const {return shortcut_;

int shortcut(int s) {shortcut_ = s;

textfont() const {return textfont_;

textfont(Fl_Font s) {textfont_ = s; mColumnScale = 0;

textSize() const {return textsize_;

textsize(Fl_Fontsize s) {textsize_ = s; mColumnScale = 0;

Fl_Font textfont() const {return textfont_;

textfont(Fl_Font s) {textfont_ = s; mColumnScale = 0;

Fl_Fontsize textsize() const {return textsize_;

textsize(Fl_Fontsize s) {textsize_ = s; mColumnScale = 0;

Fl_Color textcolor() const {return textcolor_;

textcolor(Fl_Color c) {textcolor_ = c;

int wrapped_column(int row, int column) const;

int wrapped_row(int row) const;

void wrap_mode(int wrap, int wrap_margin);

virtual void resize(int X, int Y, int W, int H);

void draw();

draw_text(int X, int Y, int W, int H);

draw_range(int start, int end);

draw_cursor(int, int);

void draw_string(int style, int x, int y, int toX, const char *string,

int find_x(const char *s, int len, int style, int x) const;

enum {
  DRAW_LINE,
  FIND_INDEX,
  FIND_INDEX_FROM_ZERO,
  GET_WIDTH
};

int handle_vline(int mode,

int lineStart, int lineLen, int leftChar, rightChar,

topClip, int bottomClip,

int leftClip, int rightClip) const;

void draw_line_numbers(bool clearAll);

void clear_rect(int style, int x, int y, int width, int height) const;

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();
void update_h_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;
maintain_absolute_top_line_number(int oldFirstChar);
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);
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;
int wrap_uses_character(int lineEndPos) const;
damaged_range1_start, damaged_range1_end;
damaged_range2_start, damaged_range2_end;
mCursorPos;
mCursorOn;
mCursorOldY;
*/ Y pos. of cursor for blanking */
mCursorToHint; /* Tells the buffer modified callback
where to move the cursor, to reduce
the number of redraw calls */
mCursorStyle; /* One of enum cursorStyles above */
mCursorPreferredXPos; /* Pixel position for vert. cursor movement */
mVisibleLines; /* # of visible (displayed) lines */
mBufferedLines; /* # of newlines in the buffer */
FL_Text_Buffer mBuffer; /* Contains text to be displayed */
FL_Text_Buffer mStyleBuffer; /* Optional parallel buffer containing
color and font information */
mFirstChar, mLastChar; /* Buffer positions of first and last
displayed character (lastChar points
either to a newline or one character
beyond the end of the buffer) */
mContinuousWrap; /* Wrap long lines when displaying */
mWrapMarginFix; /* Margin in # of pixels for
wrapping in continuousWrap mode */
//mLineStarts;
mTopLineNum; /* Line number of top displayed line
of file (first line of file is 1) */
mAbsTopLineNum; /* In continuous wrap mode, the line
number of the top line if the text
were not wrapped (note that this is
only maintained as needed). */
mNeedAbsTopLineNum; /* Externally settable flag to continue
maintaining absTopLineNum even if
it isn't needed for line # display */
mHorizOffset; /* Horizontal scroll pos. in pixels */
mTopLineNumHint; /* Line number of top displayed line
define file (first line of file is 1) */
mHorizOffsetHint; /* Horizontal scroll pos. in pixels */
mNStyles; /* Number of entries in styleTable */
const Style_Table_Entry *mStyleTable; /* Table of fonts and colors for

00487 char mUnfinishedStyle; /* Style buffer entry which triggers
00489 Unfinished_Style_Cb mUnfinishedHighlightCB; /* Callback to parse "unfinished" */
00491 regions */
00492 void mHighlightCBArg; /* Arg to unfinishedHighlightCB */
00493
00494 int mMaxsize;
00495
00496 int mSuppressResync; /* Suppress resynchronization of line
00497 starts during buffer updates */
00498 int mLinesDeleted; /* Number of lines deleted during
00499 buffer modification (only used
00500 when resynchronization is suppressed) */
00501 int mModifyingTabDistance; /* Whether tab distance is being
00502 modified */
00503
00504 mutable double mColumnScale; /* Width in pixels of an average character. This
00505 value is calculated as needed (lazy eval); it
00506 needs to be mutable so that it can be calculated
00507 within a method marked as "const" */
00508
00509 Fl_Color mCursor_color;
00510 Fl_Scrollbar* mHScrollBar;
00512 Fl_Scrollbar* mVScrollBar;
00513 int scrollbar_width_;
00514 int dragPos, dragType, dragging;
00516 int display_insert_position_hint;
00517 struct { int x, y, w, h; | text_area;
00518
00519 int shortcut_;}
00520
00521 Fl_Font textfont_;}
00522 Fl_Fontsize textsize_;}
00523 Fl_Color textcolor_;}
00527
00529 // Line number margin and width
00526 int mLineNumLeft, mLineNumWidth;
00527
00528 // Line number fonts/colors
00529 #if FLTK_ABI_VERSION >= 10303
00530 Fl_Font linenumber_font_;}
00531 Fl_Fontsize linenumber_size_;}
00532 Fl_Color linenumber_fgcolor_;}
00533 Fl_Color linenumber_bgcolor_;}
00534 Fl_Align linenumber_align_;}
00535 const char* linenumber_format_;}
00536 #endif
00537);
00538
00541 //
00542 // End of "$Id$".
00543 //

10.126  Fl_Text_Editor.H

00001 //
00002 "$Id$"
00003 //
00004 // Header file for Fl_Text_Editor class.
00005 //
00006 // Copyright 2001-2010 by Bill Spitzak and others.
00007 // Original code Copyright Mark Edel. Permission to distribute under
00008 // the LGPL for the FLTK library granted by Mark Edel.
00009 //
00010 // This library is free software. Distribution and use rights are outlined in
00011 // the file "COPYING" which should have been included with this file. If this
00012 // file is missing or damaged, see the license at:
00013 //
00014 // http://www.fltk.org/COPYING.php
00015 //
00016 // Please report all bugs and problems on the following page:
00017 //
00018 // http://www.fltk.org/str.php
00019 //
00020 //
00021 // $file
00022 Fl_Text_Editor widget , */
00023
00025 ifndef FL_TEXT_EDITOR_H
#define FL_TEXT_EDITOR_H

#include "Fl_Text_Display.H"

// key will match in any state
#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);

void insert_mode(int b) { insert_mode_ = b; }

void default_key_function(Key_Func f) { default_key_function_ = f; }

protected:
}

// key will match in any state
#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);

void insert_mode(int b) { insert_mode_ = b; }

void default_key_function(Key_Func f) { default_key_function_ = f; }

protected:
}

// key will match in any state
#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);

void insert_mode(int b) { insert_mode_ = b; }

void default_key_function(Key_Func f) { default_key_function_ = f; }

protected:
}

// key will match in any state
#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);

void insert_mode(int b) { insert_mode_ = b; }

void default_key_function(Key_Func f) { default_key_function_ = f; }

protected:
}
00145  static Key_Binding* global_key_bindings;
00146
00147 #ifndef FL_DOXYGEN
00148  Key_Func default_key_function;
00149 #endif
00150 
00151 #endif
00152 #endif
00153
00154 //
00155 // End of "$Id$",
00156 //
00157

10.127  Fl_Tile.H

00001 //
00002 // "$Id$"
00003 //
00004 // Tile header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2016 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 //  http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 //  http://www.fltk.org/str.php
00017 //
00018 #ifndef Fl_Tile_H
00019 #define Fl_Tile_H
00020 #define Fl_Tile_H
00021 #include "Fl_Group.H"
00022/*
00023 The Fl_Tile class lets you resize its children by dragging
00024 the border between them.
00025 */
00026
class FL_EXPORT Fl_Tile : public Fl_Group {
00027 public:
00028  int handle(int event);
00029  Fl_Tile(int X, int Y, int W, int H, const char *L=0);
00030  void resize(int X, int Y, int W, int H);
00031  void position(int oldx, int oldy, int newx, int newy);
00032 
00033 #endif
00034 #endif
00035 //
00036 // End of "$Id$",
00037 //

10.128  Fl_Tiled_Image.H

00001 //
00002 // "$Id$"
00003 //
00004 // Tiled image header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2015 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 //  http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 //  http://www.fltk.org/str.php
00017 //
00018 /*
00019 \file
00020 Fl_Tiled_Image widget . */
00021 #ifndef Fl_Tiled_Image_H
00022

Generated by Doxygen
class FL_EXPORT Fl_Tiled_Image : public Fl_Image {
protected:
  Fl_Image *image_; // The image that is tiled
  int alloc_image_; // Did we allocate this image?

public:
  Fl_Tiled_Image(Fl_Image *i, int W = 0, int H = 0);
  virtual ~Fl_Tiled_Image();
  virtual Fl_Image *copy(int W, int H);
  Fl_Image *copy() { return copy(w(), h()); }
  virtual void color_average(Fl_Color c, float i);
  virtual void desaturate();
  virtual void draw(int X, int Y, int W, int H, int cx, int cy);
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  Fl_Image *image() { return image_; }
};
10.130 Fl_Toggle_Button.H

...
10.132  Fl_Toggle_Round_Button.H

```c
#ifndef Fl_Toggle_Round_Button
#define Fl_Toggle_Round_Button Fl_Round_Button
#endif
```

10.133  Fl_Tooltip.H

```c
class Fl_Tooltip {
  static float delay() { return delay_; }
  static void delay(float f) { delay_ = f; }
  static float hoverdelay() { return hoverdelay_; }
  static void hoverdelay(float f) { hoverdelay_ = f; }
  static int enabled() { return Fl::option(Fl::OPTION_SHOW_TOOLTIPS); }
  static void enable(int b = 1) { Fl::option(Fl::OPTION_SHOW_TOOLTIPS, (b!=0)); }
  static void disable() { enable(0); }
  static Fl_Widget* current() { return widget_; }
  static void current(Fl_Widget* w);
  static Fl_Font font() { return font_; }
  static void font(Fl_Font i) { font_ = i; }
  static Fl_Fontsize size() { return (size_ == -1 ? Fl_NORMAL_SIZE : size_); }
  static void size(Fl_Fontsize s) { size_ = s; }
  static Fl_Color color() { return color_; }
  static void color(Fl_Color c) { color_ = c; }
  static Fl_Color textcolor() { return textcolor_; }
  static void textcolor(Fl_Color c) { textcolor_ = c; }
  #if FLTK_ABI_VERSION >= 10301
  static Fl_Font font();
  static void font(Fl_Font i) {
    font_ = i;
  }
  static Fl_Fontsize size() {
    return (size_ == -1 ? FL_NORMAL_SIZE : size_);
  }
  static void size(Fl_Fontsize s) {
    size_ = s;
  }
  static Fl_Color color() {
    return color_;
  }
  static void color(Fl_Color c) {
    color_ = c;
  }
  static Fl_Color textcolor() {
    return textcolor_;}
  static void textcolor(Fl_Color c) {
    textcolor_ = c;
  }
  #endif
};
```
0083  static int margin_width() { return margin_width_;  
0085  static void margin_width(int v) { margin_width_ = v;  
0087  static int margin_height() { return margin_height_;  
0089  static void margin_height(int v) { margin_height_ = v;  
0091  static int wrap_width() { return wrap_width_;  
0093  static void wrap_width(int v) { wrap_width_ = v;  
0094  #else  
0095  static int margin_width() { return 3;  
0096  static int margin_height() { return 3;  
0097  static int wrap_width() { return 400;  
0098  #endif  
0099  
0100  #ifdef __APPLE__  
0101  // the unique tooltip window  
0102  static Fl_Window* current_window(void);  
0103  #endif  
0104  
0105  // These should not be public, but Fl_Widget::tooltip() needs them...  
0106  private:  
0107  private:  
0108  friend void Fl_Widget::tooltip(const char *);  
0109  friend void Fl_Widget::copy_tooltip(const char *);  
0110  static void enter_(Fl_Widget* w);  
0111  static void exit_(Fl_Widget* w);  
0112  static void set_enter_exit_once_();  
0113  
0114  private:  
0115  static float delay_;  
0116  static float hoverdelay_;  
0117  static Fl_Color color_;  
0118  static Fl_Color textcolor_;  
0119  static Fl_Font font_;  
0120  static Fl_Fontsize size_;  
0121  static Fl_Widget* widget_;  
0122  #if FLTK_ABI_VERSION >= 10401  
0123  static int margin_width_;  
0124  static int margin_height_;  
0125  static int wrap_width_;  
0126  #endif  
0127  };  
0128  #endif  
0129  #endif  
0130  #elif  
0131  // End of "$Id$".  
0132  // End of "$Id$".  
0133  //

10.134  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 {
     FL_TREE_REASON_NONE =0 , FL_TREE_REASON_SELECTED , FL_TREE_REASON_DESELECTED ,
     FL_TREE_REASON_RESELECTED ,
     FL_TREE_REASON_OPENED , FL_TREE_REASON_CLOSED , FL_TREE_REASON_DRAGGED 
   }

   The reason the callback was invoked.
10.134.1 Detailed Description

This file contains the definitions of the Fl_Tree class.

10.134.2 Enumeration Type Documentation

10.134.2.1 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 (e.g. double-clicked)</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>

10.135 Fl_Tree.H

Go to the documentation of this file.
00317 Fl_Tree_Item * _callback_item;        // item invoked during callback (can be NULL)
00318 Fl_Tree_Reason _callback_reason;     // reason for the callback
00319 Fl_Tree_Prefs _prefs;                // all the tree's settings
00320 int _scrollbar_size;                 // size of scrollbar trough
00321 #if FLTK_ABI_VERSION >= 10301
00322 // NEW:
00323 Fl_Tree_Item * _lastselect;           // used to extend selections
00324 #else /*FLTK_ABI_VERSION*/
00325 // OLD: static data inside handle() method
00326 #endif /*FLTK_ABI_VERSION*/
00327 void fix_scrollbar_order();
00328
00329 protected:
00330 Fl_Scrollbar *_vscroll;
00331 #if FLTK_ABI_VERSION >= 10303
00332 Fl_Scrollbar *_hscroll;
00333 int _tox,_toy,_tow,_toh;
00334 int _tix,_tiy,_tiw,_tih;
00335 int _tree_w;
00336 int _tree_h;
00337 #endif
00338 void item_clicked(Fl_Tree_Item * val);
00339 void do_callback_for_item(Fl_Tree_Item * item, Fl_Tree_Reason reason);
00340 // next_visible_item() and extend_selection() moved to 'public' in ABI 1.3.3
00341 // undocmented draw_tree() dropped -- draw() does all the work now
00342 #ifndef FLTK_ABI_VERSION >= 10303
00343 #endif
00344 Fl_Tree_Item *next_visible_item(Fl_Tree_Item *start, int dir);
00345 void extend_selection(Fl_Tree_Item *from, Fl_Tree_Item *to);
00346 int draw_tree();
00347 #endif
00348
00349 public:
00350 Fl_Tree(int X, int Y, int W, int H, const char *L=0);
00351 ~Fl_Tree();
00352 int handle(int e);
00353 void draw();
00354 void show_self();
00355 void resize(int,int,int,int);
00356 // root methods
00357 void root_label(const char *new_label);
00358 Fl_Tree_Item * root();
00359 void root(Fl_Tree_Item *newitem);
00360 const Fl_Tree_Prefs& prefs() const { return _prefs; }
00361
00362 // Item creation/removal methods
00363 #if FLTK_ABI_VERSION >= 10303
00364 Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem=0);
00365 #else
00366 Fl_Tree_Item *add(const char *path);
00367 Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem);
00368 #endif
00369 Fl_Tree_Item * add(Fl_Tree_Item *parent_item, const char *name);
00370 Fl_Tree_Item *insert_above(Fl_Tree_Item *above, const char *name);
00371 Fl_Tree_Item * insert(Fl_Tree_Item *item, const char *name, int pos);
00372 int remove(Fl_Tree_Item *item);
00373 void clear();
00374 void clear_children(Fl_Tree_Item *item);
00375
00376 // Item lookup methods
00377 #if FLTK_ABI_VERSION >= 10303
00378 Fl_Tree_Item *find_item(const char *path, int yonly=0) const;
00379 Fl_Tree_Item *find_item(const char *path) const;
00380 Fl_Tree_Item *find_item(const char *path, int yonly=0) const;
00381 Fl_Tree_Item *find_item(const char *path);  
00382 #else
00383 Fl_Tree_Item *find_item(const char *path, int yonly=0) const;
00384 Fl_Tree_Item *find_item(const char *path);  
00385 #endif
00386 Fl_Tree_Item *first();
00387 Fl_Tree_Item *last();
00388 Fl_Tree_Item *first_selected_item();
00389 Fl_Tree_Item *last_selected_item();
00390 Fl_Tree_Item *next_item(Fl_Tree_Item *item, int dir=FL_Down, bool visible=false);
00422 #if FLTK_ABI_VERSION >= 10303
00423 Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item=0, int dir=FL_Down);
00424 int get_selected_items(Fl_Tree_Item_Array &ret_items);
00425 #else
00426 Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item=0);
00427 Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item, int dir);
00428 #endif

00441 // Item open/close methods
00443 int open(Fl_Tree_Item *item, int docallback=1);
00444 int open(const char *path, int docallback=1);
00445 void open_toggle(Fl_Tree_Item *item, int docallback=1);
00446 int close(Fl_Tree_Item *item, int docallback=1);
00447 int close(const char *path, int docallback=1);
00448 #if FLTK_ABI_VERSION >= 10303
00449 int is_open(Fl_Tree_Item *item) const;
00450 int is_close(const char *path) const;
00451 #else
00452 int is_open(const char *path) const;
00453 int is_close(Fl_Tree_Item *item) const;
00454 #endif

00444 // Item selection methods
00445 int select(Fl_Tree_Item *item, int docallback=1);
00446 int select(const char *path, int docallback=1);
00447 void select_toggle(Fl_Tree_Item *item, int docallback=1);
00448 int deselect(Fl_Tree_Item *item, int docallback=1);
00449 int deselect(const char *path, int docallback=1);
00450 int select_all(Fl_Tree_Item *item=0, int docallback=1);
00451 int select_only(Fl_Tree_Item *selitem, int docallback=1);
00452 int extend_selection_dir(Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible);
00453 int extend_selection(Fl_Tree_Item *from, Fl_Tree_Item *to, int val, bool visible=false);
00454 #if FLTK_ABI_VERSION >= 10303
00455 int extend_selection__(Fl_Tree_Item *from, Fl_Tree_Item *to, int val, bool visible=false);
00456 #endif

00466 // Item attribute related methods
00467 Fl_Font item_labelfont() const;
00468 void item_labelfont(Fl_Font val);
00469 Fl_Fontsize item_labelsize() const;
00470 void item_labelsize(Fl_Fontsize val);
00471 Fl_Color item_labelfgcolor(void) const;
00472 void item_labelfgcolor(Fl_Color val);
00473 Fl_Color item_labelbgcolor(void) const;
00474 void item_labelbgcolor(Fl_Color val);
00475 Fl_Color connectorcolor() const;
00476 void connectorcolor(Fl_Color val);
00477 int marginleft() const;
00478 void marginleft(int val);
00479 int margintop() const;
00480 void margintop(int val);
00481 int marginbottom() const;
00482 void marginbottom(int val);
00483 int openchild_marginbottom() const;
00484 void openchild_marginbottom(int val);
00485 int usericonmarginleft() const;
00486 void usericonmarginleft(int val);
00487 int labelmarginleft() const;
00488 #if FLTK_ABI_VERSION >= 10301
00489 int widgetmarginleft() const;
00490 void widgetmarginleft(int val);
00491 #endif /*FLTK_ABI_VERSION*/
00492 int connectorwidth() const;
00493 void connectorwidth(int val);
00494 Fl_Image * usericon() const;
#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Draw.H>
#include <FL/Fl_Image.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item_Array.H>
#include <FL/Fl_Tree_Prefs.H>

This file contains the definitions for Fl_Tree_Item.

### Classes

- **Fl_Tree_Item**

  A tree widget item.

---

Generated by Doxygen
10.136.1 Detailed Description

This file contains the definitions for Fl_Tree_Item.

10.137 Fl_Tree_Item.H

Go to the documentation of this file.

---

```c
// 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:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems on the following page:
// http://www.fltk.org/str.php

class Fl_Tree;

class FL_EXPORT Fl_Tree_Item {
  #if FLTK_ABI_VERSION >= 10303
    Fl_Tree *_tree; // parent tree
  #endif
  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’)
  #if FLTK_ABI_VERSION >= 10303
    enum Fl_Tree_Item_Flags {
      OPEN = 1«0,
      VISIBLE = 1«1,
      ACTIVE = 1«2,
      SELECTED = 1«3
    };
  #else /*FLTK_ABI_VERSION*/
    // OLD: this will go away after 1.3.x
    char _open; // item is open?
    char _visible; // item is visible?
    char _active; // item activated?
    char _selected; // item selected?
  #endif /*FLTK_ABI_VERSION*/
  int _xywh[4]; // xywh of this widget (if visible)
  int _collapse_xywh[4]; // xywh of collapse icon (if visible)
  int _label_xywh[4]; // xywh of label
  Fl_Widget *_widget; // item’s label widget (optional)
  Fl_Image *_usericon; // item’s user-specific icon (optional)
  #if FLTK_ABI_VERSION >= 10304
    Fl_Image *_userdeicon; // deactivated usericon
  #endif
  Fl_Tree_Item_Array _children; // array of child items
  Fl_Tree_Item *_parent; // parent item (=0 if root)
  void *_userdata; // user data that can be associated with an item
  #if FLTK_ABI_VERSION >= 10301
    Fl_Tree_Item *_prev_sibling; // previous sibling (same level)
    Fl_Tree_Item *_next_sibling; // next sibling (same level)
  #endif /*FLTK_ABI_VERSION*/
};
```

---
10.137 Fl_Tree_Item.H

00113 // Protected methods
00114 protected:
00115 void _Init(const Fl_Tree_Prefs &prefs, Fl_Tree *tree);
00116 void show_widgets();
00117 void hide_widgets();
00118 void draw_vertical_connector(int x, int y1, int y2, const Fl_Tree_Prefs &prefs);
00119 void draw_horizontal_connector(int x1, int x2, int y, const Fl_Tree_Prefs &prefs);
00120 void recalc_tree();
00121 int calc_item_height(const Fl_Tree_Prefs &prefs) const;
00122 #if FLTK_ABI_VERSION >= 10303
00123 Fl_Color drawfgcolor() const;
00124 Fl_Color drawbgcolor() const;
00125 #endif
00126 00127 public:
00128 Fl_Tree_Item(const Fl_Tree_Prefs &prefs); // CTOR -- backwards compatible
00129 #if FLTK_ABI_VERSION >= 10303
00130 Fl_Tree_Item(Fl_Tree *tree); // CTOR -- ABI 1.3.3+
00131 virtual ~Fl_Tree_Item(); // DTOR -- ABI 1.3.3+
00132 #else
00133 ~Fl_Tree_Item(); // DTOR -- backwards compatible
00134 #endif
00135 Fl_Tree_Item(const Fl_Tree_Item &o); // COPY CTOR
00136 int x() const { return(_xywh[0]); }
00137 int y() const { return(_xywh[1]); }
00138 int w() const { return(_xywh[2]); }
00139 int h() const { return(_xywh[3]); }
00140 int label_x() const { return(_label_xywh[0]); }
00141 int label_y() const { return(_label_xywh[1]); }
00142 int label_w() const { return(_label_xywh[2]); }
00143 int label_h() const { return(_label_xywh[3]); }
00144 #if FLTK_ABI_VERSION >= 10303
00145 virtual int draw_item_content(int render);
00146 void draw(int X, int Y, int W, Fl_Tree_Item *itemfocus,
00147 int &tree_item_xmax, int lastchild=1, int render=1);
00148 #else
00149 void draw(int X, int Y, int W, Fl_Widget *tree,
00150 Fl_Tree_Item *itemfocus, const Fl_Tree_Prefs &prefs, int lastchild=1);
00151 #endif
00152 show_self(const char *indent = "") const;
00153 void label(const char *val);
00154 const char *label() const;
00155 inline void user_data( void * data ) { _userdata = data; }
00156 inline void * user_data() const { return _userdata; }
00157 void labelfont(Fl_Font val) {
00158 _labelfont = val;
00159 recalc_tree(); // may change tree geometry
00160 }
00161 Fl_Font labelfont() const {
00162 return(_labelfont);
00163 }
00164 void labelsize(Fl_Fontsize val) {
00165 _labelsize = val;
00166 recalc_tree(); // may change tree geometry
00167 }
00168 Fl_Fontsize labelsize() const {
00169 return(_labelsize);
00170 }
00171 void labelbgcolor(Fl_Color val) {
00172 _labelbgcolor = val;
00173 }
00174 Fl_Color labelbgcolor() const {
00175 return(_labelbgcolor);
00176 }
00177 void labelcolor(Fl_Color val) {
00178 _labelcolor = val;
00179 }
00180 Fl_Color labelcolor() const {
00181 return(_labelcolor);
00182 }
00183 void widget(Fl_Widget *val) {
00184 _widget = val;
00185 recalc_tree(); // may change tree geometry
00186 }
00187 Fl_Widget *widget() const {
00188 return(_widget);
00189 }
00190 int children() const {
return(_children.total());
}
Fl_Tree_Item *child(int index) {
  return(_children[index]);
}
const Fl_Tree_Item *child(int t) const;
int has_children() const {
  return(children());
}
int find_child(const char *name);
int find_child(Fl_Tree_Item *item);
int remove_child(Fl_Tree_Item *item);
int remove_child(const char *new_label);
void clear_children();
void swap_children(int ax, int bx);
int swap_children(Fl_Tree_Item *a, Fl_Tree_Item *b);
const Fl_Tree_Item *find_child(const char *name) const;
const Fl_Tree_Item *find_child(Fl_Tree_Item *item) const;
int remove_child(Fl_Tree_Item *item);
int remove_child(const char *new_label);
void clear_children();
void swap_children(int ax, int bx);
int swap_children(Fl_Tree_Item *a, Fl_Tree_Item *b);
const Fl_Tree_Item *find_child(char **arr) const;
const Fl_Tree_Item *find_item(char **arr) const;
Fl_Tree_Item *find_item(const char *name);
Fl_Tree_Item *find_item(Fl_Tree_Item *item);
const Fl_Tree_Item *find_item(const char *name);
Fl_Tree_Item *find_item(char **arr);
Fl_Tree_Item *find_item(const char *name);
Fl_Tree_Item *find_item(Fl_Tree_Item *item);
Fl_Tree_Item *find_item(char **arr);
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                   const char *new_label,
                   Fl_Tree_Item *newitem);
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                   const char *new_label);
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                   char **arr,
                   Fl_Tree_Item *newitem);
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                   char **arr);
#if FLTK_ABI_VERSION >= 10303
Fl_Tree_Item *replace(Fl_Tree_Item *new_item);
Fl_Tree_Item *replace_child(Fl_Tree_Item *olditem, Fl_Tree_Item *newitem);
#endif
Fl_Tree_Item *insert(const Fl_Tree_Prefs &prefs, const char *new_label, int pos=0);
Fl_Tree_Item *insert_above(const Fl_Tree_Prefs &prefs, const char *new_label);
int deparent(int index);
int reparent(Fl_Tree_Item *newchild, int index);
int move(int to, int from);
int move(Fl_Tree_Item *item, int op=0, int pos=0);
int move_above(Fl_Tree_Item *item);
int move_below(Fl_Tree_Item *item);
int move_into(Fl_Tree_Item *item, int pos=0);
int depth() const;
Fl_Tree_Item *prev();
Fl_Tree_Item *next();
Fl_Tree_Item *next_sibling();
Fl_Tree_Item *prev_sibling();
void update_prev_next(int index);
Fl_Tree_Item *next_displayed(Fl_Tree_Prefs &prefs); // deprecated
Fl_Tree_Item *prev_displayed(Fl_Tree_Prefs &prefs); // deprecated
Fl_Tree_Item *next_visible(Fl_Tree_Prefs &prefs);
Fl_Tree_Item *prev_visible(Fl_Tree_Prefs &prefs);

Fl_Tree_Item *parent() {
  return(_parent);
}
const Fl_Tree_Item *parent() const {
  return(_parent);
}
void parent(Fl_Tree_Item *val) {
  _parent = val;
}
#if FLTK_ABI_VERSION >= 10303
const Fl_Tree_Prefs& prefs() const;
#endif
Fl_Tree *tree() {
  return(_tree);
}
#if FLTK_ABI_VERSION >= 10304
Fl_Tree *tree() {
  return(_tree);
}
#endif
// State
void open();
void close();
int is_open() const {
  return(is_flag(OPEN));
}
int is_close() const {
  return(is_flag(OPEN)?0:1);
}
void open_toggle() {

// Adding items

} // State

Generated by Doxygen
is_open()?close():open();  // handles calling recal_tree()
void select(int val=1) {
    set_flag(SELECTED, val);
}
void select_toggle() {
    if ( is_selected() ) {
        deselect();  // deselect if selected
    } else {
        select();  // select if deselected
    }
}
void select(int val=1) {
    set_flag(SELECTED, val);
}

void select_toggle() {
    if ( is_selected() ) {
        deselect();  // deselect if selected
    } else {
        select();  // select if deselected
    }
}

int select_all() {
    int count = 0;
    if ( ! is_selected() ) {
        select();
        ++count;
    }
    for ( int t=0; t<children(); t++ ) {
        count += child(t)->select_all();
    }
    return(count);
}

void deselect() {
    set_flag(SELECTED, 0);
}

int deselect_all() {
    int count = 0;
    if ( is_selected() ) {
        deselect();
        ++count;
    }
    for ( int t=0; t<children(); t++ ) {
        count += child(t)->deselect_all();
    }
    return(count);
}

char is_selected() const {
    return(is_flag(SELECTED));
}

void activate(int val=1) {
    set_flag(ACTIVE, val);
    if ( _widget && val != (int)_widget->active() ) {
        if ( val ) {
            _widget->activate();
        } else {
            _widget->deactivate();
        }
        _widget->redraw();
    }
}

void deactivate() {
    activate(0);
}

char is_activated() const {
    return(is_flag(ACTIVE));
}

char is_active() const {
    return(is_activated());
}

int visible() const {
    return(is_visible());
}

int is_visible() const {
    return(is_flag(VISIBLE));
}

void usericon(Fl_Image *val) {
    _usericon = val;
    recal_tree();  // may change tree geometry
}

Fl_Image *usericon() const {
    return(_usericon);
}

#if FLTK_ABI_VERSION >= 10304
void userdeicon(Fl_Image * val) {
    _userdeicon = val;
}
#endif

Fl_Image * userdeicon() const {
    return _userdeicon;
}

// Events
#if FLTK_ABI_VERSION >= 10303
const Fl_Tree_Item* find_clicked(const Fl_Tree_Prefs &prefs, int yonly=0) const;
Fl_Tree_Item* find_clicked(const Fl_Tree_Prefs &prefs, int yonly=0);
00495 #else
00496 const Fl_Tree_Item* find_clicked(const Fl_Tree_Prefs &prefs) const;
00497 Fl_Tree_Item * find_clicked(const Fl_Tree_Prefs &prefs);  
00498 #endif
00499 int event_onCollapse_icon(const Fl_Tree_Prefs &prefs) const;
00500 int event_on_label(const Fl_Tree_Prefs &prefs) const;
00502 int is_root() const {
00503    return (_parent == nullptr);
00504 }
00505 // Protected methods
00507 // TODO: move these to top 'protected:' section
00508 protected:
00509 #if FLTK_ABI_VERSION >= 10301
00511 inline void set_flag(unsigned short flag, int val) {
00512    if (flag==OPEN || flag==VISIBLE) {
00513        recalc_tree(); // may change tree geometry
00515    } else _flags &= ~flag;
00516 }
00518 inline int is_flag(unsigned short flag) const {
00519    return (_flags & flag ? 1 : 0);
00520 }
00523 void set_flag(unsigned short flag, int val) {
00525    switch(flag) {
00526        case OPEN:    _open = val; break;
00527        case VISIBLE: _visible = val; break;
00528        case SELECTED: _selected = val; break;
00530    }
00532    int is_flag(unsigned short flag) const {
00533        switch(flag) {
00534            case OPEN: return (_open ? 1 : 0);
00535            case VISIBLE: return (_visible ? 1 : 0);
00536            case ACTIVE: return (_active ? 1 : 0);
00537            default: return (0);
00539        }
00542    }  
00541 #endif /*FLTK_ABI_VERSION*/
00542 #else /*FLTK_ABI_VERSION*/
00543    int is_flag(unsigned short flag) const {
00544        switch(flag) {
00545            case OPEN: return (_open ? 1 : 0);
00546            case VISIBLE: return (_visible ? 1 : 0);
00547            case ACTIVE: return (_active ? 1 : 0);
00549     // End of "$Id$"
00549

10.138 Fl_Tree_Item_Array.H File Reference

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.

10.138.1 Detailed Description

This file defines a class that manages an array of Fl_Tree_Item pointers.

10.139 Fl_Tree_Item_Array.H

Go to the documentation of this file.
# define _FL_TREE_ITEM_ARRAY_H

#include <FL/Fl.H>
#include "Fl_Export.H"

class 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:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems on the following page:
// http://www.fltk.org/str.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

#if FLTK_ABI_VERSION >= 10303
enum {
    MANAGE_ITEM = 1,
};
#else /*FLTK_ABI_VERSION*/
#endif

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);
}

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);
int remove(Fl_Tree_Item *item);

#if FLTK_ABI_VERSION >= 10303
void manage_item_destroy(int val) {
    if ( val ) _flags |= MANAGE_ITEM; else _flags &= ~MANAGE_ITEM;
}
#else /*FLTK_ABI_VERSION*/
#endif

int manage_item_destroy() const {
    return _flags & MANAGE_ITEM ? 1 : 0;
}

};

#endif /*_FL_TREE_ITEM_ARRAY_H*/

// End of "$Id$".

Generated by Doxygen
10.140 Fl_Tree_Prefs.H File Reference

This file contains the definitions for Fl_Tree’s preferences.

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

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_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.

10.140.1 Detailed Description

This file contains the definitions for Fl_Tree’s preferences.

```
Fl_Tree_Prefs :
    ............
    |
Fl_Tree :
    |____ Fl_Tree_Item
```

10.140.2 Enumeration Type Documentation

10.140.2.1 Fl_Tree_Connector

```c
enum Fl_Tree_Connector
```

Defines the style of connection lines between items.

<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>
10.140.2.2  Fl_Tree_Item_Draw_Mode

enum Fl_Tree_Item_Draw_Mode
Bit flags that control how item's labels and widget()s are drawn in the tree via item_draw_mode().

Enumerator

<table>
<thead>
<tr>
<th>Enum</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>
<tr>
<td>FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET</td>
<td>If widget() defined, include label to the left of the widget.</td>
</tr>
<tr>
<td>FL_TREE_ITEM_HEIGHT_FROM_WIDGET</td>
<td>If widget() defined, widget()'s height controls item's height.</td>
</tr>
</tbody>
</table>

10.140.2.3  Fl_Tree_Item_Reselect_Mode

enum Fl_Tree_Item_Reselect_Mode
Defines the ways an item can be (re) selected via item_reselect_mode().

Enumerator

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECTABLE_ONCE</td>
<td>Item can only be selected once (default)</td>
</tr>
<tr>
<td>FL_TREE_SELECTABLE_ALWAYS</td>
<td>Enables FL_TREE_REASON_RESELECTED events for callbacks.</td>
</tr>
</tbody>
</table>

10.140.2.4  Fl_Tree_Select

enum Fl_Tree_Select
Tree selection style.

Enumerator

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECT_NONE</td>
<td>Nothing selected when items are clicked.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE</td>
<td>Single item selected when item is clicked (default)</td>
</tr>
<tr>
<td>FL_TREE_SELECT_MULTI</td>
<td>Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE_DRAGGABLE</td>
<td>Single items may be selected, and they may be reordered by mouse drag.</td>
</tr>
</tbody>
</table>

10.140.2.5  Fl_Tree_Sort

enum Fl_Tree_Sort
Sort order options for items added to the tree.

Enumerator

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SORT_NONE</td>
<td>No sorting; items are added in the order defined (default).</td>
</tr>
<tr>
<td>FL_TREE_SORT_ASCENDING</td>
<td>Add items in ascending sort order.</td>
</tr>
<tr>
<td>FL_TREE_SORT_DESCENDING</td>
<td>Add items in descending sort order.</td>
</tr>
</tbody>
</table>
```c
#include <FL/Fl.H> // needed for ABI version features (via Enumerations.H)

// This file is part of the Fl_Tree widget for FLTK

// Copyright (C) 2009-2010 by Greg Ercolano.

// 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

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
};

#if FLTK_ABI_VERSION >= 10301
enum Fl_Tree_Item_Reselect_Mode {
    FL_TREE_SELECTABLE_ONCE=0,
    FL_TREE_SELECTABLE_ALWAYS,
};
#endif /*FLTK_ABI_VERSION*/

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
};

// Colors

int Fl_Color _labelbgcolor; // label's background color
int Fl_Color _connectorcolor; // connector dotted line color

Fl_Image *openimage; // the 'open' icon [+]
Fl_Image *closeimage; // the 'close' icon [-]:
Fl_Image *userimage; // user's own icon
```

Generated by Doxygen

```
00136 Fl_Image *_openimage;  // deactivated 'open' icon
00137 Fl_Image *_closedimage; // deactivated 'close' icon
00138 Fl_Image *_userimage;  // deactivated user icon
00139 #endif
00140 char _showcollapse; // 1=show collapse icons, 0=don’t
00141 char _showroot; // show the root item as part of the tree
00142 Fl_Tree_Sort _sortorder; // none, ascending, descending, etc.
00143 Fl_Boxtype _selectbox; // selection box type
00144 Fl_Tree_Select _selectmode; // selection mode
00145 #: if FLTK_ABI_VERSION >= 10301
00146 Fl_Tree_Item_Reselect_Mode _itemreselectmode; // controls item selection callback() behavior
00147 Fl_Tree_Item_Draw_Mode _itemdrawmode; // controls how items draw label + widget()
00148 #: endif
00149 #: if FLTK_ABI_VERSION >= 10303
00150 Fl_Tree_Item_Draw_Callback *_itemdrawcallback; // callback to handle drawing items (0=none)
00151 void *_itemdrawuserdata; // data for drawing items (0=none)
00152 #: endif
00153 public:
00154 Fl_Tree_Prefs();
00155 #: if FLTK_ABI_VERSION >= 10304
00156 ~Fl_Tree_Prefs();
00157 #: endif
00158
00160 // Labels
00163 inline Fl_Font item_labelfont() const { return(_labelfont); }
00164 inline void item_labelfont(Fl_Font val) { _labelfont = val; }
00165 inline Fl_Fontsize item_labelsize() const { return(_labelsize); }
00166 inline void item_labelsize(Fl_Fontsize val) { _labelsize = val; }
00167 inline Fl_Color item_labelfgcolor() const { return(_labelfgcolor); }
00168 inline void item_labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
00169 #if FLTK_ABI_VERSION >= 10301
00170 inline Fl_Color item_labelbgcolor() const { return(_labelbgcolor); }
00171 inline void item_labelbgcolor(Fl_Color val) { _labelbgcolor = val; }
00172 #: endif /*FLTK_ABI_VERSION*/
00173 #else /*FLTK_ABI_VERSION*/
00174 inline Fl_Color item_labelbgcolor() const { return(_labelbgcolor); }
00175 inline void item_labelbgcolor(Fl_Color val) { _labelbgcolor = val; }
00176 #endif /*FLTK_ABI_VERSION*/
00177
00180 // Margins
00183 inline int marginleft() const { return(_marginleft); }
00185 inline void marginleft(int val) { _marginleft = val; }
00187 inline int margintop() const { return(_margintop); }
00189 inline void margintop(int val) { _margintop = val; }
00192 #if FLTK_ABI_VERSION >= 10301
00193 inline int marginbottom() const { return(_marginbottom); }
00195 inline void marginbottom(int val) { _marginbottom = val; }
00196 #: endif /*FLTK_ABI_VERSION*/
00197
00200 // Obsolete names - for 1.3.0 backwards compat
00203 inline Fl_Font labelfont() const { return(_labelfont); }
00205 inline void labelfont(Fl_Font val) { _labelfont = val; }
00207 inline Fl_Fontsize labelsize() const { return(_labelsize); }
00209 inline void labelsize(Fl_Fontsize val) { _labelsize = val; }
00211 inline Fl_Color labelfgcolor() const { return(_labelfgcolor); }
00213 inline void labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
00215 inline Fl_Color labelbgcolor() const { return(_labelbgcolor); }
00217 inline void labelbgcolor(Fl_Color val) { _labelbgcolor = val; }
00218
00220 // Margins
00223 inline int marginleft() const { return(_marginleft); }
00225 inline void marginleft(int val) { _marginleft = val; }
00227 inline int margintop() const { return(_margintop); }
00229 inline void margintop(int val) { _margintop = val; }
00232 inline int marginbottom() const { return(_marginbottom); }
00234 inline void marginbottom(int val) { _marginbottom = val; }
00236 #: if FLTK_ABI_VERSION >= 10301
00238 inline int openchild_marginbottom() const { return(_openchild_marginbottom); }
00241 inline void openchild_marginbottom(int val) { _openchild_marginbottom = val; }
00244 inline int usericonmarginleft() const { return(_usericonmarginleft); }
00246 inline void usericonmarginleft(int val) { _usericonmarginleft = val; }
00249 #: endif /*FLTK_ABI_VERSION*/
00251 inline int usericonmargintop() const { return(_usericonmargintop); }
00254 inline void usericonmargintop(int val) { _usericonmargintop = val; }
00257 inline int usericonmarginbottom() const { return(_usericonmarginbottom); }
00259 inline void usericonmarginbottom(int val) { _usericonmarginbottom = val; }
00262
inline int labelmarginleft() const {
    return(_labelmarginleft);
}
inline void labelmarginleft(int val) {
    _labelmarginleft = val;
}
#ifdef FLTK_ABI_VERSION >= 10301
inline int widgetmarginleft() const {
    return(_widgetmarginleft);
}
inline void widgetmarginleft(int val) {
    _widgetmarginleft = val;
}
#endif
inline int linespacing() const {
    return(_linespacing);
}
inline void linespacing(int val) {
    _linespacing = val;
}
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;
    if (_connectorstyle == Fl_Tree_Connector::Node) {
        Fl_Tree_Connector::Node *node = static_cast<Fl_Tree_Connector::Node*>(val);
        _connectorwidth = node->width();
    }
}
inline void connectorstyle(int val) {
    _connectorstyle = Fl_Tree_Connector(val);
}
inline int connectorwidth() const {
    return(_connectorwidth);
}
inline void connectorwidth(int val) {
    _connectorwidth = val;
}
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;
    if (_openimage) delete _openimage;
    if (_closeimage) delete _closeimage;
    _userdeimage = _userimage->copy();
    _userdeimage->inactive();
    if (_openimage) _openimage = _openimage->copy();
    if (_closeimage) _closeimage = _closeimage->copy();
    if (_userimage) _userimage->inactive();
    _userimage = 0;
}
inline Fl_Image *opendeicon() const {
    return _opendeimage;
}
inline Fl_Image *closedeicon() const {
    return _closedeimage;
}
inline Fl_Image *userdeicon() const {
    return _userdeimage;
}
#endif
inline Fl_Color usercolor() const {
    return(_usercolor);
}
inline void usercolor(Fl_Color val) {
    _usercolor = val;
}
inline Fl_Font *font() const {
    return(_font);
}
inline void font(Fl_Font *val) {
    _font = val;
}
inline void font(Fl_Font val) {
    _font = val;
}
inline Fl_Fontstyle userfontstyle() const {
    return(_userfontstyle);
}
inline void userfontstyle(Fl_Fontstyle val) {
    _userfontstyle = val;
}
inline void userfontstyle(int val) {
    _userfontstyle = Fl_Fontstyle(val);
}
inline int userfontPointSize() const {
    return(_userfontPointSize);
}
inline void userfontPointSize(int val) {
    _userfontPointSize = val;
}
}
inline Fl_Fontsize userfontsize() const {
    return(_userfontsize);
}
inline void userfontsize(Fl_Fontsize val) {
    _userfontsize = val;
}
inline void userfontsize(int val) {
    _userfontsize = Fl_Fontsize(val);
}
inline int fontsize() const {
    return(_fontsize);
}
inline void fontsize(int val) {
    _fontsize = val;
}
inline char showcollapse() const {
    return(_showcollapse);
}
inline void showcollapse(char val) {
    _showcollapse = val;
}
This file contains simple "C"-style type definitions.

### Typedefs

#### Miscellaneous

- `typedef unsigned int Fl_Char
  24-bit Unicode character - upper 8 bits are unused`
- `typedef const char * Fl_CString
  Flexible length UTF-8 Unicode read-only string.`
- `typedef unsigned int Fl_Shortcut
  24-bit Unicode character + 8-bit indicator for keyboard flags`
• typedef char * Fl_String
  
  Flexible length UTF-8 Unicode text.
• typedef unsigned char uchar
  
  unsigned char
• typedef unsigned long ulong
  
  unsigned long

10.142.1 Detailed Description

This file contains simple "C"-style type definitions.

10.142.2 Typedef Documentation

10.142.2.1 Fl_CString
typedef const char* Fl_CString

Flexible length UTF-8 Unicode read-only string.

See also

  Fl_String

10.142.2.2 Fl_String
typedef char* Fl_String

Flexible length UTF-8 Unicode text.

Todo FIXME: temporary (?) typedef to mark UTF-8 and Unicode conversions

10.143 fl_types.h

Go to the documentation of this file.
10.144 fl_utf8.h File Reference

header for Unicode and UTF-8 character handling

```c
#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <locale.h>
```

Macros

- `#define xchar unsigned short`

Functions

- `FL_EXPORT int fl_access (const char *f, int mode)`
  
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.

- `FL_EXPORT int fl_chmod (const char *f, int mode)`
  
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.

- `FL_EXPORT int fl_execvp (const char *file, char **argv)`

  Cross-platform function to open files with a UTF-8 encoded name.

- `FL_EXPORT char * fl_getcwd (char *b, int l)`

  Cross-platform function to get the current working directory as a UTF-8 encoded value.

- `FL_EXPORT char * fl_getenv (const char *v)`

  Cross-platform function to get environment variables with a UTF-8 encoded name or value.

- `FL_EXPORT char fl_make_path (const char *path)`

  Cross-platform function to recursively create a path in the file system.

- `FL_EXPORT void fl_make_path_for_file (const char *path)`

  Cross-platform function to create a path for the file in the file system.

- `FL_EXPORT int fl_mkdir (const char *f, int mode)`

  Cross-platform function to create a directory with a UTF-8 encoded name.

- `FL_EXPORT unsigned int fl_nonspacing (unsigned int ucs)`

  Returns true if the Unicode character ucs is non-spacing.

- `FL_EXPORT int fl_open (const char *f, int oflags,...)`

  Cross-platform function to open files with a UTF-8 encoded name.

- `FL_EXPORT int fl_rename (const char *f, const char *n)`

  Cross-platform function to rename a filesystem object using UTF-8 encoded names.

- `FL_EXPORT int fl_rmdir (const char *f)`

  Cross-platform function to remove a directory with a UTF-8 encoded name.

- `FL_EXPORT int fl_stat (const char *f, struct stat *b)`

  Cross-platform function to stat() a file using a UTF-8 encoded name or value.

- `FL_EXPORT int fl_system (const char *cmd)`

  Cross-platform function to run a system command with a UTF-8 encoded string.

- `FL_EXPORT int fl_tolower (unsigned int ucs)`

  Returns the Unicode lower case value of ucs.

- `FL_EXPORT int fl_toupper (unsigned int ucs)`

  Returns the Unicode upper case value of ucs.

- `FL_EXPORT unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)`

- `FL_EXPORT int fl_unlink (const char *f)`

Generated by Doxygen
Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

- FL_EXPORT char * fl_utf2mbcs (const char *s)
  Converts UTF-8 string s to a local multi-byte character string.
- FL_EXPORT const char * fl_utf8back (const char *p, const char *start, const char *end)
  Return the number of bytes needed to encode the given UCS4 character in UTF-8.
- FL_EXPORT unsigned fl_utf8decode (const char *p, const char *end, int *len)
- FL_EXPORT int fl_utf8encode (unsigned ucs, char *buf)
- FL_EXPORT unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)
- FL_EXPORT unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen)
- FL_EXPORT unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)
- FL_EXPORT const char * fl_utf8fwd (const char *p, const char *start, const char *end)
- FL_EXPORT 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.
- FL_EXPORT 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.
- FL_EXPORT int fl_utf8locale (void)
- FL_EXPORT int fl_utf8test (const char *src, unsigned len)
- FL_EXPORT unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)
- FL_EXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
- FL_EXPORT unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
- FL_EXPORT unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
- FL_EXPORT int fl_utf_nb_char (const unsigned char *buf, int len)
  Returns the number of Unicode chars in the UTF-8 string.
- FL_EXPORT int fl_utf_strcasecmp (const char *s1, const char *s2)
  UTF-8 aware strcasecmp - converts to Unicode and tests.
- FL_EXPORT int fl_utf_strncasecmp (const char *s1, const char *s2, int n)
  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
- FL_EXPORT int fl_utf_tolower (const unsigned char *str, int len, char *buf)
  Converts the string str to its lower case equivalent into buf.
- FL_EXPORT int fl_utf_toupper (const unsigned char *str, int len, char *buf)
  Converts the string str to its upper case equivalent into buf.
- FL_EXPORT int fl_wcwidth (const char *src)
  extended wrapper around fl_wcwidth(unsigned int ucs) function.
- FL_EXPORT int fl_wcwidth_ (unsigned int ucs)
  wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK

10.144.1 Detailed Description

header for Unicode and UTF-8 character handling

10.145 fl_utf8.h

Go to the documentation of this file.

00001 /*
00002 * "$Id$"
00003 *
00004 * Author: Jean-Marc Lienher ( http://oksid.ch )
00005 * Copyright 2000-2010 by O'ksi'D.
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 * http://www.fltk.org/COPYING.php
00012 */

Generated by Doxygen
/* Please report all bugs and problems on the following page:
 * http://www.fltk.org/str.php
 */

Merged in some functionality from the fltk-2 version. IMM.
The following code is an attempt to merge the functions incorporated in FLTK2
with the functions provided in OksiD’s fltk-1.1.6-utf8 port

#ifndef _HAVE_FL_UTF8_HDR_
#define _HAVE_FL_UTF8_HDR_
#endif

#include "Fl_Export.H"
#include "fl_types.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#ifdef WIN32
#include <sys/types.h>
#include <sys/stat.h>
#include <locale.h>
#include <ctype.h>
define xchar wchar_t
#else
# include <wchar.h>
#include <sys/stat.h>
define xchar unsigned short
#endif

#ifdef __cplusplus
extern "C" {
#endif

/* F2: comes from FLTK2 */
 /* OD: comes from OksiD */

FL_EXPORT int fl_utf8bytes(unsigned ucs);

/* OD: returns the byte length of the first UTF-8 char sequence (returns -1 if not valid) */
FL_EXPORT int fl_utf8len(char c);

/* OD: returns the byte length of the first UTF-8 char sequence (returns +1 if not valid) */
FL_EXPORT int fl_utf8len1(char c);

/* 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);

/* 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: comes from FLTK2 */
/* OD: comes from OksiD */

FL_EXPORT int fl_utf8len(char cl);

FL_EXPORT int fl_utf8len1(char cl);

FL_EXPORT int fl_utf8bytes(unsigned ucs);

FL_EXPORT int fl_utf8len(char c);

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 unsigned fl_utf8encode(const unsigned ucs, char* buf);

/* F2: Move forward to the next valid UTF-8 sequence start between start and end */
FL_EXPORT char* fl_utf8fwd(const char* start, const char* end);

/* F2: Move backward to the previous valid UTF-8 sequence start */
FL_EXPORT char* fl_utf8back(const char* start, const char* end);

/* XX: Convert a single 32-bit Unicode value into UTF16 */
/* 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 *dst, unsigned dstlen, const wchar_t *src, unsigned srclen);

/* 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);

/* 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, 
 depending on original build options */
FL_EXPORT int fl_wcwidth(const char *src);

/* OD: Return true if the character is non-spacing */
FL_EXPORT unsigned int fl_nonspacing(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);

#ifndef WIN32
/* OD: Attempt to convert the UTF-8 string to the current locale */
FL_EXPORT char *fl_utf8_to_locale(const char *s, int len, unsigned int codepage);

/* OD: Attempt to convert a string in the current locale to UTF-8 */
FL_EXPORT char *fl_locale_to_utf8(const char *s, int len, unsigned int codepage);
#endif

>())
/* 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* f, int o, ...);

/* OD: Portable UTF-8 aware unlink wrapper */
FL_EXPORT int fl_unlink(const char *f);

/* OD: Portable UTF-8 aware rmdir wrapper */
FL_EXPORT int fl_rmdir(const char *f);

/* OD: Portable UTF-8 aware getenv wrapper */
FL_EXPORT char* fl_getenv(const char *name);

/* OD: Portable UTF-8 aware mkdir wrapper */
FL_EXPORT int fl_mkdir(const char* f, int mode);

/* 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 */

#ifndef _HAVE_FL_UTF8_HDR_
#endif /* _HAVE_FL_UTF8_HDR_ */

*/

FILE \* Fl_Valuator widget . */

ifndef Fl_Valuator_H

#define Fl_Valuator_H

#else

#endif /* Fl_Valuator_H */

ifndef Fl_Widget_H

#include "Fl_Widget.H"

#endif /* Fl_Widget_H */

// 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 {
    double value_;
double previous_value_; // truncates to this range after rounding
double min, max; // rounds to multiples of A/B, or no rounding if A is zero

protected:
int horizontal() const {return type() & FL_HORIZONTAL;}
Fl_Valuator(int X, int Y, int W, int H, const char * L);

public:
void bounds(double a, double b) {min = a; max = b;}
double minimum() const {return min;}
double minimum(double a) {min = a;}
double maximum() const {return max;}
double 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);
int horizontal() const {return type() & FL_HORIZONTAL;}
double previous_value() const {return previous_value_;
void handle_push() {previous_value_ = value_;
void 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;}

void bounds(double a, double b) {min = a; max = b;}
double minimum() const {return min;}
double minimum(double a) {min = a;}
double maximum() const {return max;}
double 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);
int horizontal() const {return type() & FL_HORIZONTAL;}
double previous_value() const {return previous_value_;
void handle_push() {previous_value_ = value_;
void 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;}

10.147 Fl_Value_Input.H
void draw();

protected:

public:

void resize(int, int, int, int);

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) {textfont_ = s;

Fl_Fontsize textsize() const {return input.textsize();

void textsize(Fl_Fontsize s) {textsize_ = s;

Fl_Color textcolor() const {return input.textcolor();

void textcolor(Fl_Color n) {textcolor_ = n;

Fl_Color cursor_color() const {return input.cursor_color();

void cursor_color(Fl_Color n) {cursor_color_ = n;}}

};

#endf

// End of "$Id$".

*/

/**
 * Fl_Value_Output widget. *
 */

#ifndef Fl_Value_Output_H
#define Fl_Value_Output_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Value_Output : public Fl_Valuator {

Fl_Font textfont_;  

Fl_Fontsize textsize_;  

uchar soft_;  

Fl_Color textcolor_;  

protected:

void draw();

public:

int handle(int);

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 n) {textcolor_ = n;}}

};

#endf

Generated by Doxygen
10.149 Fl_Value_Slider.H

10.150 Fl_Widget.H File Reference
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.
- typedef long fl_intptr_t
- typedef unsigned long fl_uintptr_t

10.150.1 Detailed Description
Fl_Widget, Fl_Label classes.

10.150.2 Macro Definition Documentation

10.150.2.1 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.

10.150.3 Typedef Documentation

10.150.3.1 fl_intptr_t
typedef long fl_intptr_t

Todo: typedef's fl_intptr_t and fl_uintptr_t should be documented.

10.151 Fl_Widget.H

Go to the documentation of this file.
typedef unsigned long fl_uintptr_t;
#endif

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;

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_);
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,
    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,
    COPIED_TOOLTIP = 1 « 17,
    FULLSCREEN = 1 « 18,
    MAC_USE_ACCENTS_MENU = 1 « 19,
    USERFLAG3 = 1 « 29,
    USERFLAG2 = 1 « 30,
    USERFLAG1 = 1 « 31
};

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() {draw_focus(box(), x(), y(), w(), h());}
void draw_focus(Fl_Boxtype t, int x, int y, int w, int h) 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);}]

Fl_Group* parent() const {return parent_;
for hacks only, use Fl_Group::add();

uchar type() const {return type_;}

virtual void resize(int x, int y, int w, int h);

damage_resize(int, int, int, int);

void position(int X, int Y) {resize(X, Y, w_, h_);}

size(int W, int H) {resize(x_, y_, W, H);}

Fl_Align align() const {return label_.align_;

void align(Fl_Align alignment) {label_.align_ = alignment;

void color(Fl_Color bg) {color_ = bg;}

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 color(Fl_Color c) {label_.color = c;

Fl_Fontsize labelsize() const {return label_.size;

void labelcolor(Fl_FontColor c) {label_.color = c;

Fl_Font labelfont() const {return label_.font;

void labelcolor(Fl_FontColor c) {label_.color = c;

Fl_Font labelfont() const {return label_.font;

void labelcolor(Fl_FontColor c) {label_.color = c;

Fl_Fontsize labelsize() const {return label_.size;

void labelcolor(Fl_Fontsize pix) {label_.size = pix;

Fl_Image* image() {return label_.image;

const Fl_Imag* image() {return label_.image;

Generated by Doxygen
void image(Fl_Image* img) {label_.image=img;
}

void image(Fl_Image& img) {label_.image=&img;
}

Fl_Image * deimage() {return label_.deimage;
const Fl_Image * deimage() const {return label_.deimage;

Fl_Callback_p callback() const {return callback_;
void callback(Fl_Callback* cb, void* p) {callback_=cb; user_data_=p;

void callback(Fl_Callback* cb) {callback_=cb;
void callback(Fl_Callback0* cb) {callback_=(Fl_Callback*)cb;
void callback(Fl_Callback1* cb, long p=0) {callback_=(Fl_Callback*)cb;
user_data_=((void*)p);

void* user_data() const {return user_data_;
void* user_data(void* v) {user_data_=v;
long argument() const {return (long)(fl_intptr_t)user_data_;
int argument_r() const;

virtual void show();
virtual void hide();

void set_visible() {flags_&=~INVISIBLE;}
void clear_visible() {flags_|=INVISIBLE;}

unsigned int visible() const {return !(flags_&INVISIBLE);}
int visible_r() const;

virtual void activate();
void activate();

void deactivate();

void set_output() {flags_|=OUTPUT;}
void clear_output() {flags_&=~OUTPUT;}

unsigned int takesevents() const {return !(flags_&(INACTIVE|INVISIBLE|OUTPUT));}

unsigned int output() const {return (flags_&OUTPUT);}

unsigned int changed() const {return flags_&CHANGED;}
unsigned int active() const {return !(flags_&INACTIVE);}
int active_r() const;

void visible_focus(int v) { if (v) set_visible_focus(); else clear_visible_focus();

unsigned int visible_focus() { return flags_&VISIBLE_FOCUS; }
static void default_callback(Fl_Widget *cb, void *d);
void do_callback() {do_callback(this, user_data_);}
void do_callback(Fl_Widget * o, long arg) {do_callback(o, (void*){fl_intptr_t)arg);}
void do_callback(Fl_Widget * o, void * arg=0);

void redraw();
void redraw_label();
uchar damage() const {return damage_;
void clear_damage(uchar c = 0) {damage_ = c;}
void damage(uchar c);
void damage(uchar c, int x, int y, int w, int h);
void draw_label(int, int, int, int, Fl_Align) const;
void measure_label(int& ww, int& hh) const {label_.measure(ww, hh);}
Fl_Window* window() const ;
Fl_Window* top_window() const;
Fl_Window* top_window_offset(int& xoff, int& yoff) const;
virtual Fl_Group* as_group() {return 0;}
virtual Fl_Window* as_window() {return 0;}
virtual class Fl_Gl_Window* as_gl_window() {return 0;}
int use_accents_menu() { return flags() & MAC_USE_ACCENTS_MENU; }
Fl_Color color2() const {return (Fl_Color)color2_;
void color2(unsigned a) {color2_ = a;}
#endif

// Causes a widget to invoke its callback function with arbitrary arguments.
// Documentation and implementation in Fl_Widget.cxx

int contains(const Fl_Widget *w) const ;
int inside(const Fl_Widget * wgt) const {return wgt ? wgt->contains(this) : 0;}

int test_shortcut();
static unsigned int label_shortcut(const char *t);
static int test_shortcut(const char *, const bool require_alt = false);
void _set_fullscreen() {flags_ |= FULLSCREEN;}
void _clear_fullscreen() {flags_ &= ~FULLSCREEN;}

int contains(const Fl_Widget * w) const ;
int inside(const Fl_Widget * wgt) const {return wgt ? wgt->contains(this) : 0;}
redraw();
redraw_label();
damage();

#define FL_RESERVED_TYPE 100

// End of "$Id$.
//
Data supporting a non-rectangular window shape.

Macros

- **#define FL_DOUBLE_WINDOW** 0xF1  
  double window type id
- **#define FL_WINDOW** 0xF0  
  window type id all subclasses have type() \( \geq \) this

10.152.1 Detailed Description

**Fl_Window** widget.

10.153 **Fl_Window.H**

Go to the documentation of this file.

```cpp
#ifndef Fl_Window_H  
define Fl_Window_H

#ifdef WIN32  
#include <windows.h>
#endif

#include "Fl_Group.H"
#include "Fl_Bitmap.H"
#include <stdlib.h>

#define FL_WINDOW 0xF0
#define FL_DOUBLE_WINDOW 0xF1

class Fl_X;
class Fl_RGB_Image;

class FL_EXPORT Fl_Window : public Fl_Group {
  static char *default_xclass_;
  
  // Note: we must use separate statements for each of the following 8 variables,
  // with the static attribute, otherwise MS VC++ 2008/2010 complains :-((
  static // when these members are static, ABI compatibility with 1.3.0 is respected
  int no_fullscreen_x;
  static // when these members are static, ABI compatibility with 1.3.0 is respected
  int no_fullscreen_y;
  static // when these members are static, ABI compatibility with 1.3.0 is respected
  int no_fullscreen_w;
  static // when these members are static, ABI compatibility with 1.3.0 is respected
  int no_fullscreen_h;
  static // when these members are static, ABI compatibility with 1.3.0 is respected
  int fullscreen_screen_top;

  // Generated by Doxygen
```
#include <FL/Fl_Window.H>

int fullscreen_screen_bottom;

int fullscreen_screen_left;

int fullscreen_screen_right;

friend class Fl_X;

Fl_X *i; // points at the system-specific stuff

struct icon_data {
    const void *legacy_icon;
    Fl_RGB_Image **icons;
    int count;
    HICON big_icon;
    HICON small_icon;
};

const char * iconlabel_;  // size_range stuff:

int minw, minh, maxw, maxh;

uchar size_range_set;

// cursor stuff

Fl_Cursor cursor_default;

Fl_Color cursor_fg, cursor_bg;

protected:

struct shape_data_type {
    int lw_, lh_;  // cursor_fg, cursor_bg;
    Fl_Image * shape_;  // cursor
};

#if FLTK_ABI_VERSION < 10303 && !defined(FL_DOXYGEN)

shape_data_type *shape_data_;  // cursor

#endif

private:

void shape_alpha_(Fl_Image * img, int offset);

void shape_pixmap_(Fl_Image * pixmap);

public:

void shape(const Fl_Image * img);

void shape(const Fl_Image& b) { shape(&b); }

protected:

virtual void draw();

virtual void flush();

private:

void size_range_();

void _Fl_Window(); // constructor innards

void fullscreen_x(); // platform-specific part of sending a window to full screen

void fullscreen_off_x(int X, int Y, int W, int H); // platform-specific part of leaving full screen

// unimplemented copy ctor and assignment operator

Fl_Window(const Fl_Window&);

Fl_Window& operator=(const Fl_Window&);

protected:

static Fl_Window *current_;  // constructor

virtual void draw();

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); }

doctoration::

public:

void free_icons();

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();

virtual int handle(int);

virtual void resize(int X, int Y, int W, int H);

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;}

void clear_modal_states() {clear_flag(NON_MODAL | MODAL);}

void set_menu_window() {set_flag(MENU_WINDOW);}

unsigned int menu_window() const {return flags() & MENU_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) {
  this->minw = minw;
  this->minh = minh;
  this->maxw = maxw;
  this->maxh = maxh;
  this->dw = dw;
  this->dh = dh;
  this->aspect = aspect;
  size_range_();
}

const char * label() const {return Fl_Widget::label();}

const char * iconlabel() const {return iconlabel_;

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;

void xclass(const char * c);

static void default_icon(const Fl_RGB_Image *);

static void default_icons(const Fl_RGB_Image *[], int);

void icon(const void * ic);

void icon(const Fl_RGB_Image *);

void icons(const Fl_RGB_Image *[], int);

virtual void show();

virtual void hide();

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);
```cpp
#include <FL/Fl_Group.H>

class FL_EXPORT Fl_Wizard : public Fl_Group {

Fl_Widget *value_;

void draw();

public:

Fl_Wizard(int, int, int, int, const char * = 0);

void next();

void prev();

Fl_Widget *value();

void value(Fl_Widget *);

};
```

---

10.154 Fl_Wizard.H

```cpp
/* File Fl_Wizard widget. */

// Include necessary header files...

class FL_EXPORT Fl_Wizard : public Fl_Group {

Fl_Widget *value_;

void draw();

public:

Fl_Wizard(int, int, int, int, const char * = 0);

void next();

void prev();

Fl_Widget *value();

void value(Fl_Widget *);

};
```
10.155  Fl_XBM_Image.H

```cpp
#ifndef Fl_XBM_Image_H
#define Fl_XBM_Image_H
#include "Fl_Bitmap.H"

class FL_EXPORT Fl_XBM_Image : public Fl_Bitmap {

public:

Fl_XBM_Image(const char * filename);

};
#endif // !Fl_XBM_Image_H
```

10.156  Fl_XPM_Image.H

```cpp
#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
```
Forms emulation header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2011 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

#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;

ifndef __FORMS_H__
define __FORMS_H__
#endif

define NULL 0
#define FALSE 0
#define TRUE 1

#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_LCOL FL_BLACK
#define FL_COL1 FL_GRAY
#define FL_MCOL FL_LIGHT1
#define FL_LEFT_BCOL FL_LIGHT3 // 53 is better match
#define FL_TOP_BCOL FL_LIGHT2 // 51
#define FL_BOTTOM_BCOL FL_DARK2 // 40
#define FL_RIGHT_BCOL FL_DARK3 // 36
#define FL_INACTIVE FL_INACTIVE_COLOR
#define FL_INACTIVE_COL FL_INACTIVE_COLOR
#define FL_FREE_COL1 FL_FREE_COLOR
#define FL_FREE_COL2 ((Fl_Color)(FL_FREE_COLOR+1))
#define FL_FREE_COL3 ((Fl_Color)(FL_FREE_COLOR+2))
#define FL_FREE_COL4 ((Fl_Color)(FL_FREE_COLOR+3))
#define FL_FREE_COL5 ((Fl_Color)(FL_FREE_COLOR+4))
#define FL_FREE_COL6 ((Fl_Color)(FL_FREE_COLOR+5))
#define FL_FREE_COL7 ((Fl_Color)(FL_FREE_COLOR+6))
#define FL_FREE_COL8 ((Fl_Color)(FL_FREE_COLOR+7))
#define FL_FREE_COL9 ((Fl_Color)(FL_FREE_COLOR+8))
#define FL_FREE_COL10 ((Fl_Color)(FL_FREE_COLOR+9))
#define FL_FREE_COL11 ((Fl_Color)(FL_FREE_COLOR+10))
#define FL_FREE_COL12 ((Fl_Color)(FL_FREE_COLOR+11))
#define FL_FREE_COL13 ((Fl_Color)(FL_FREE_COLOR+12))
#define FL_FREE_COL14 ((Fl_Color)(FL_FREE_COLOR+13))
#define FL_FREE_COL15 ((Fl_Color)(FL_FREE_COLOR+14))
#define FL_FREE_COL16 ((Fl_Color)(FL_FREE_COLOR+15))
#define FL_TOMATO ((Fl_Color)(131))
#define FL_INDIANRED ((Fl_Color)(164))
#define FL_SLATEBLUE ((Fl_Color)(195))
#define FL_DARKGOLD ((Fl_Color)(84))
#define FL_PALEGREEN ((Fl_Color)(157))
#define FL_ORCHID ((Fl_Color)(203))
#define FL_DARKCYAN ((Fl_Color)(189))
#define FL_DARKTOMATO ((Fl_Color)(113))
#define FL_WHEAT ((Fl_Color)(174))
#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
#define FL_MBUTTON4 4
#define FL_MBUTTON5 5

#define FL_INVALID_STYLE 255
#define FL_NORMAL_STYLE FL_HELVETICA
#define FL_BOLD_STYLE FL_HELVETICA_BOLD
#define FL_ITALIC_STYLE FL_HELVETICA_ITALIC
#define FL_BOLDITALIC_STYLE FL_HELVETICA_BOLD_ITALIC
#define FL_SHADOW_STYLE (FL_SHADOW_LABEL«8)
#define FL_ENGRAVED_STYLE (FL_ENGRAVED_LABEL«8)
#define FL_EMBOSSED_STYLE (FL_EMBOSSED_LABEL«0)

// size values are different from XForms, match older Forms:
#define FL_TINY_SIZE 8
#define FL_SMALL_SIZE 11 // 10
#define FL_NORMAL_SIZE 14 // 12
#define FL_MEDIUM_SIZE 18 // 14
#define FL_LARGE_SIZE 24 // 18
#define FL_HUGE_SIZE 32 // 24
#define FL_TINY_FONT FL_TINY_SIZE
#define FL_SMALL_FONT FL_SMALL_SIZE
#define FL_NORMAL_FONT FL_NORMAL_SIZE
#define FL_MEDIUM_FONT FL_MEDIUM_SIZE
#define FL_LARGE_FONT FL_LARGE_SIZE
#define FL_HUGE_FONT FL_HUGE_SIZE
#define FL_NORMAL_FONT1 FL_SMALL_FONT
#define FL_NORMAL_FONT2 FL_NORMAL_FONT
#define FL_DEFAULT_FONT FL_NORMAL_FONT
#define FL_DEFAULT_FONT2 FL_NORMAL_FONT

#define FL_RETURN_END_CHANGED FL_WHEN_RELEASE
#define FL_RETURN_CHANGED FL_WHEN_CHANGED
#define FL_RETURN_END FL_WHEN_RELEASE_ALWAYS
#define FL_RETURN_ALWAYS (FL_WHEN_CHANGED|FL_WHEN_NOT_CHANGED)

#define FL_BOUND_WIDTH 3

typedef int FL_Coord;
typedef int FL_COLOR;

// fltk interaction:
#define FL_CMD_OPT void
extern FL_EXPORT void fl_initialize(int*, char**[], const char*, FL_CMD_OPT*, int);
inline void fl_finish() {}

// type of callback is different and no "id" number is returned:
inline void fl_add_timeout(long msec, void (*cb)(void*), void* v) {
    Fl::add_timeout(msec *.001, cb, v);
}
inline void fl_remove_timeout(int) {}

// type of callback is different!
inline void fl_set_idle_callback(void (*cb)(void*), void* v) {
    Fl::set_idle(cb);
}
inline void fl_check_forms();

inline void fl_set_idle_callback(void (*cb)(void*), void* v) {
    Fl::set_idle(cb);
}
inline void fl_check_forms();

// because of new redraw behavior, these are no-ops:
inline void fl_freeze_object(Fl_Widget*) {}
inline void fl_unfreeze_object(Fl_Widget*) {}
inline void fl_freeze_form(Fl_Window*) {}
inline void fl_unfreeze_form(Fl_Window*) {}
inline void fl_freeze_all_forms() {}
inline void fl_unfreeze_all_forms() {}

inline void fl_set_focus_object(Fl_Window*, Fl_Widget* o) {Fl::focus(o);}
inline void fl_reset_focus_object(Fl_Widget* o) {Fl::focus(o);}
define fl_set_object_focus fl_set_focus_object

// void fl_set_form_atclose(Fl_Window*w,int (*cb)(Fl_Window*,void*),void* v)
// 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);}

/* forms lib font indexes must be byte sized - extract correct byte from style word */
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);}
inline void fl_set_object_size(Fl_Widget* o, int w, int h) {o->size(w,h);}
inline void fl_set_object_geometry(Fl_Widget* o, int x, int y, int w, int h) {o->resize(x,y,w,h);}

// Fl_Window:

typedef void (*Forms_CB)(Fl_Widget*, long);

inline void fl_free_form(Fl_Window* x) {delete x;}
inline void fl_redraw_form(Fl_Window* f) {f->redraw();}

inline void fl_bgn_form(Fl_Boxtype b,int w,int h) {
    Fl_Window * g = new Fl_Window(w,h,0);
    g->labelfont((Fl_Font)(b&0xff)); g->labelfont((Fl_Labeltype)(b»8));
    g->show();
}
inline void fl_end_form();
inline void fl_start_form();
inline void fl_end_form();
inline void fl_start_form();
inline void fl_start_form();
inline void fl_add_to_form(Fl_Window* f) {f->begin();}
inline void fl_group(Fl_Bgn_Group) {return new Fl_Group(0,0,0,0,0);}
inline void fl_end_group() {Fl_Group::current()->forms_end();}
inline void fl_add_to_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);
enum { // "p" argument values:
    FL_PLACE_FREE = 0, // make resizable
    FL_PLACE_MOUSE = 1, // mouse centered on form
    FL_PLACE_CENTER = 2, // center of the screen
    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
    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_TRANSIENT
    //FL_MODAL = (1«8) // not implemented yet in Forms
    FL_MODAL = 0
    //FL_MODAL = 1
    //FL_MODAL = (1«8)
}:
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);}
#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_frame(b,x,y,w,h,bgc);
}
inline void fl_draw_text(Fl_Align align, int x, int y, int w, int h, Fl_Font style, const char* s) {
    fl_font(style, size);
    fl_color(fgcolor);
    fl_draw(s, x, y, w, h, align);
}

// this does not work except for CENTER...
inline void fl_draw_text_beside(Fl_Align align, int x, int y, int w, int h, Fl_Font style, const char* s) {
    fl_font(style, size);
    fl_color(fgcolor);
    fl_draw(s, x, y, w, h, align);
}

inline void fl_set_font_name(Fl_Font n, const char* s) {Fl::set_font(n, s);}
inline void fl_mapcolor(Fl_Color c, uchar r, uchar g, uchar b) {Fl::set_color(c, r, g, b);}
#define fl_set_clipping(x, y, w, h) fl_push_clip(x, y, w, h)
#define fl_unset_clipping() fl_pop_clip()

// Forms classes:
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);
}
#include "Fl_Box.H"
forms_constructorb(Fl_Box, fl_add_box)
#include "Fl_Browser.H"
forms_constructor(Fl_Browser, fl_add_browser)
inline void fl_clear_browser(Fl_Widget* o) {
    ((Fl_Browser*) o)->clear();
}
inline void fl_add_browser_line(Fl_Widget* o, const char* s) {
    ((Fl_Browser*) o)->add(s);
}
inline void fl_addto_browser(Fl_Widget* o, const char* s) {
    ((Fl_Browser*) o)->add(s); // should also scroll to bottom */
    //inline void fl_addto_browser_chars(Fl_Widget* o, const char* s) {
    //    ((Fl_Browser*) o)->add(s); // should also scroll to bottom */
}
inline void fl_insert_browser_line(Fl_Widget* o, int n, const char* s) {
    ((Fl_Browser*) o)->insert(n, s);
}
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 ((Fl_Browser*) o)->text(n);
}
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 ((Fl_Browser*) o)->text(n);
}
inline void fl_load_browser(Fl_Widget* o, const char* f) {
    ((Fl_Browser*) o)->load(f);
}
inline void fl_select_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*) o)->select(n, 1);
}
inline void fl_deselect_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*) o)->deselect();
}
inline void fl_deselect_browser(Fl_Widget* o) {
    ((Fl_Browser*) o)->deselect();
}
inline int fl_isselected_browser_line(Fl_Widget* o, int n) {
    return ((Fl_Browser *)o)->selected(n);
}
inline int fl_get_browser_topline(Fl_Widget* o) {
    return ((Fl_Browser *)o)->topline();
}
inline int fl_get_browser(Fl_Widget* o) {
    return ((Fl_Browser *)o)->value();
}
inline int fl_get_browser_maxline(Fl_Widget* o) {
    return ((Fl_Browser *)o)->size();
}
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_INOUT_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)

#define FL_LIGHT_BUTTON 0
#define FL_ROUND_BUTTON 4
#define FL_ROUND3_BUTTON 7
#define FL_TOGGLE_BUTTON FL_PUSH_BUTTON

#include "Fl_Light_Button.H"

forms_constructor(Fl_Light_Button, fl_add_lightbutton)

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* c, const char* const* c) {
    (new Fl_Pixmap(c))->label(o); // does not delete old Fl_Pixmap!
}

#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)->setbounds(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) {
}
00520 #define (Fl_Chart*)o)->autosize(v);
00521 inline void fl_set_chart_lstyle(Fl_Widget* o, Fl_Font v) {
00522 ((Fl_Chart*)o)->textfont(v);
00523 inline void fl_set_chart_lsize(Fl_Widget* o, int v) {
00524 ((Fl_Chart*)o)->textsize(v);
00525 inline void fl_set_chart_lcolor(Fl_Widget* o, Fl_Color v) {
00526 ((Fl_Chart*)o)->textcolor(v);
00527 #define fl_set_chart_lcol fl_set_chart_lcolor
00528 #include "Fl_Choice.H"
00529 inline void fl_clear_choice(Fl_Widget* o) {
00530 ((Fl_Choice *)o)->clear();
00531 inline void fl_addto_choice(Fl_Widget* o, const char* s) {
00532 ((Fl_Choice *)o)->add(s);
00533 inline void fl_replace_choice(Fl_Widget* o, int i, const char* s) {
00534 ((Fl_Choice *)o)->replace(i-1,s);
00535 inline void fl_delete_choice(Fl_Widget* o, int i) {
00536 ((Fl_Choice *)o)->remove(i-1);
00537 inline void fl_set_choice(Fl_Widget* o, int i) {
00538 ((Fl_Choice *)o)->value(i-1);
00539 #define FL_NORMAL_CHOICE 0
00540 #define FL_NORMAL_CHOICE2 0
00541 #define FL_DROPLIST_CHOICE 0
00542 forms_constructor(Fl_Choice, fl_add_choice)
00543 inline void fl_get_choice(Fl_Widget* o, int* i) {
00544 return ((Fl_Choice*)o)->value()+1;
00545 #include "Fl_Clock.H"
00546 forms_constructor(Fl_Clock, fl_add_clock)
00547 inline void fl_set_dial_value(Fl_Widget* o, double v) {
00548 ((Fl_Dial*)o)->value(v);
00549 inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
00550 ((Fl_Dial*)o)->bounds(a, b);
00551 inline void fl_get_dial_bounds(Fl_Widget* o, float* a, float* b) {
00552 *a = float(((Fl_Dial*)o)->minimum());
00553 *b = float(((Fl_Dial*)o)->maximum());
00554 #include "Fl_Counter.H"
00555 forms_constructor(Fl_Counter, fl_add_counter)
00556 inline void fl_set_counter_value(Fl_Widget* o, double v) {
00557 ((Fl_Counter*)o)->value(v);
00558 inline void fl_set_counter_bounds(Fl_Widget* o, double a, double b) {
00559 ((Fl_Counter*)o)->bounds(a, b);
00560 inline void fl_get_counter_bounds(Fl_Widget* o, float* a, float* b) {
00561 *a = float(((Fl_Counter*)o)->minimum());
00562 *b = float(((Fl_Counter*)o)->maximum());
00563 #include "Fl_Dial.H"
00564 #define FL_DIAL_COL1 FL_GRAY
00565 #define FL_DIAL_COL2 37
00566 forms_constructor(Fl_Dial, fl_add_dial)
00567 inline void fl_set_dial_value(Fl_Widget* o, float* a, float* b) {
00568 //inline void fl_set_dial_bounds(Fl_Widget* o, float* a, float* b) {
00569 //inline void fl_get_dial_bounds(Fl_Widget* o, float* a, float* b) {
00570 //inline void fl_set_dial_value(Fl_Widget* o, double v) {
00571 //inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
00572 //inline void fl_get_dial_bounds(Fl_Widget* o, double a, double b) {
00573 //inline void fl_set_dial_value(Fl_Widget* o, double v) {
00574 //inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
00575 //Cursor stuff cannot be emulated because it uses X stuff
00576 //inline void fl_set_cursor(Fl_Window* w, Fl_Cursor c) {w->cursor(c);}
00577 #define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
00578 #define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT
00579 #include "Fl_Dial.H"
00580 #define FL_DIALCOL1 FL_GRAY
00581 #define FL_DIALCOL2 37
00582 //Cursor stuff cannot be emulated because it uses X stuff
00583 #define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
00584 #define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT
00585 #include "Fl_Dial.H"
00586 #define FL_DIALCOL1 FL_GRAY
00587 #define FL_DIALCOL2 37
00588 //Cursor stuff cannot be emulated because it uses X stuff
00589 #define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
00590 #define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT
00591 #define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
00592 #include "Fl_Dial.H"
00593 #define FL_DIALCOL1 FL_GRAY
00594 #define FL_DIALCOL2 37
00595 #define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
00596 #define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT
00597 #include "Fl_Dial.H"
inline void fl_set_dial_return(Fl_Widget* o, int i) {
    ((Fl_Dial*)o)->when((Fl_When)(i | FL_WHEN_RELEASE));
}

inline void fl_set_dial_angles(Fl_Widget* o, int a, int b) {
    ((Fl_Dial*)o)->angles((short)a, (short)b);
}

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);
}

// labelframe nyi
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_TOP_LEFT);
    return o;
}

#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*, int);
FL_EXPORT int fl_show_question(const char*, const char*, const char*);

inline int fl_show_input(const char* l, const char*d=0) { return fl_input("%s", d, l); }

FL_EXPORT /*const*/ char* fl_show_simple_input(const char*, const char* deflt);
FL_EXPORT int fl_show_choice(const char*, const char*, const char*, int);
FL_EXPORT int fl_show_choices(const char*, int, const char*, const char*, int);

inline void fl_set_goodies_font(Fl_Font a, Fl_Fontsize b) { fl_message_font(a, b); }

// Forms-compatible file chooser (implementation in fselect.C):
FL_EXPORT char* fl_show_file_selector(const char*, const char*, const char*, const char*, 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* o, const char* v) {
    ((Fl_Input*)o)->value(v);
}

inline void fl_set_input_return(Fl_Widget* o, int x) {
    ((Fl_Input*)o)->when((Fl_When)(x | FL_WHEN_RELEASE));
}

inline void fl_set_input_color(Fl_Widget* o, Fl_Color a, Fl_Color b) {
    ((Fl_Input*)o)->textcolor(a);
    ((Fl_Input*)o)->cursor_color(b);
}

// inline void fl_set_input_scroll(Fl_Widget*, int);
inline void fl_set_input_cursorpos(Fl_Widget* o, int x, int /*y*/){
    ((Fl_Input*)o)->position(x);
}

// inline void fl_set_input_selected(Fl_Widget*, int);
// inline void fl_set_input_selected_range(Fl_Widget*, int, int);
// inline void fl_set_input_maxchars(Fl_Widget*, int);
// inline void fl_set_input_format(Fl_Widget*, int, int);
// inline void fl_set_input_hscrollbar(Fl_Widget*, int);
// inline void fl_set_input_vscrollbar(Fl_Widget*, int);
// inline void fl_set_input_xoffset(Fl_Widget*, int);
// inline void fl_set_input_topline(Fl_Widget*, int);
// inline int fl_get_input_topline(Fl_Widget*);
// inline int fl_get_input_scrollbarsize(Fl_Widget*, int, int);
inline void fl_set_input_topline(Fl_Widget*, int x, int y) {
    x = (int)-widget->position(); y = 0; return -x;}
// inline int fl_get_input_numberoflines(Fl_Widget*);
// inline int fl_get_input_screenlines(Fl_Widget*);
// inline void fl_set_input_xoffset(Fl_Widget*, int);
// inline int fl_get_input_format(Fl_Widget*, int*, int*);
inline int fl_get_input_topline(Fl_Widget* o) {return ((Fl_Input*)o)->value();}

#include "Fl_Menu_Button.H"

# define FL_TOUCH_MENU 0
# define FL_PUSH_MENU 1
# define FL_PULLDOWN_MENU 2
forms_constructor(Fl_Menu_Button, fl_add_menu)

inline void fl_clear_menu(Fl_Widget* o) {
    ((Fl_Menu_Button *)o)->clear();
}
inline void fl_set_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button *)o)->clear(); ((Fl_Menu_Button*)o)->add(s);
}
inline void fl_addto_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button *)o)->add(s);
}
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,x);
}
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 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_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_xstep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->xstep(v);
}
inline void fl_set_positioner_ystep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->ystep(v);
}
inline void fl_set_positioner_return(Fl_Widget* o, int v) {
    ((Fl_Positioner*)o)->when((Fl_When)(v|FL_WHEN_RELEASE));
}

#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)
# define FL_SLIDER_COL1 FL_GRAY
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_xbounds(Fl_Widget* o, double a, double b) {
    ((Fl_Slider*)o)->xbounds(a,b);
}
inline void fl_get_slider_xbounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Slider*)o)->xminum());
    *b = float(((Fl_Slider*)o)->xmaxum());
}
inline void fl_set_slider_ybounds(Fl_Widget* o, double a, double b) {
    ((Fl_Slider*)o)->ybounds(a,b);
}
inline void fl_get_slider_ybounds(Fl_Widget* o, float* a, float* b) {
    *a = float(((Fl_Slider*)o)->yminum());
    *b = float(((Fl_Slider*)o)->ymaxum());
}
inline void fl_set_slider_xstep(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->xstep(v);
}
inline void fl_set_slider_ystep(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->ystep(v);
}
inline void fl_set_slider_return(Fl_Widget* o, int v) {
    ((Fl_Slider*)o)->when((Fl_When)(v|FL_WHEN_RELEASE));
}

#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_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_xstep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->xstep(v);
}
inline void fl_set_positioner_ystep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->ystep(v);
}
inline void fl_set_positioner_return(Fl_Widget* o, int v) {
    ((Fl_Positioner*)o)->when((Fl_When)(v|FL_WHEN_RELEASE));
}

#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)
#define FL_SLIDER_COL1 FL_GRAY
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();
}

Generated by Doxygen
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_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_precision(Fl_Widget* o, int i) {
  ((Fl_Value_Slider *)o)->precision(i);
}

// The forms text object was the same as an Fl_Box except it inverted the
// meaning of FL_ALIGN_INSIDE. Implementation in forms.cxx
class FL_EXPORT Fl_FormsText : public Fl_Widget {
  protected:
    void draw();
  public:
    Fl_FormsText(Fl_Boxtype b, int X, int Y, int W, int H, const char * l=0) : Fl_Widget(X,Y,W,H,l) {box(b); align(FL_ALIGN_LEFT);}
};

// Stuff from DDForms:

// gl.h File Reference

This file defines wrapper functions for OpenGL in FLTK.
#include "Enumerations.H"
#include <GL/gl.h>

Functions

- FL_EXPORT void gl_color (Fl_Color i)
  
  Sets the current OpenGL color to an FLTK color.

- void gl_color (int c)
  
  back compatibility

- FL_EXPORT int gl_descent ()
  
  Returns the current font's descent.

- FL_EXPORT void gl_draw (const char *)
  
  Draws a null-terminated string in the current font at the current position.

- FL_EXPORT void gl_draw (const char *, float x, float y)

Generated by Doxygen
Draws a nul-terminated string in the current font at the given position.

- FL_EXPORT void gl_draw (const char *, int n)

  Draws an array of n characters of the string in the current font at the current position.

- FL_EXPORT void gl_draw (const char *, int n, float x, float y)

  Draws n characters of the string in the current font at the given position.

- FL_EXPORT 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.

- FL_EXPORT void gl_draw (const char *, int x, int y)

  Draws a nul-terminated string in the current font at the given position.

- FL_EXPORT 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.

- FL_EXPORT void gl_draw_image (const uchar *, int x, int y, int w, int h, int d=3, int ld=0)

- FL_EXPORT void gl_finish ()

  Releases an OpenGL context.

- FL_EXPORT void gl_font (int fontid, int size)

  Sets the current OpenGL font to the same font as calling fl_font().

- FL_EXPORT int gl_height ()

  Returns the current font’s height.

- FL_EXPORT 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.

- FL_EXPORT 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.

- FL_EXPORT void gl_start ()

  Creates an OpenGL context.

- FL_EXPORT double gl_width (const char *)

  Returns the width of the string in the current font.

- FL_EXPORT double gl_width (const char *, int n)

  Returns the width of n characters of the string in the current font.

- FL_EXPORT double gl_width (uchar)

  Returns the width of the character in the current font.

### 10.158.1 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 glxChooseVidual. This does not work with Fl_Double_Window’s! It will try to draw into the front buffer. Depending on the system this will 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).

### 10.158.2 Function Documentation

#### 10.158.2.1 gl_color()

FL_EXPORT 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!
10.158.2.2 gl_draw() [1/7]

FL_EXPORT void gl_draw(
    const char * str
)

Draws a nul-terminated string in the current font at the current position.

See also

On the Mac OS X platform, see gl_texture_pile_height(int)

10.158.2.3 gl_draw() [2/7]

FL_EXPORT 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

On the Mac OS X platform, see gl_texture_pile_height(int)

10.158.2.4 gl_draw() [3/7]

FL_EXPORT 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

On the Mac OS X platform, see gl_texture_pile_height(int)

10.158.2.5 gl_draw() [4/7]

FL_EXPORT 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

On the Mac OS X platform, see gl_texture_pile_height(int)

10.158.2.6 gl_draw() [5/7]

FL_EXPORT 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

On the Mac OS X platform, see gl_texture_pile_height(int)
10.158.2.7  gl_draw() [6/7]

FL_EXPORT void gl_draw (  
    const char * str,  
    int x,  
    int y )

Draws a null-terminated string in the current font at the given position.

See also

On the Mac OS X platform, see gl_texture_pile_height(int)

10.158.2.8  gl_draw() [7/7]

FL_EXPORT 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().

10.158.2.9  gl_rect()

FL_EXPORT 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.

10.158.2.10  gl_rectf()

void gl_rectf (  
    int x,  
    int y,  
    int w,  
    int h ) [inline]

Fills the given rectangle with the current color.

See also

    gl_rect(int x, int y, int w, int h)

10.159  gl.h

Go to the documentation of this file.

00001 //
00002 // "$Id$"
00003 //
00004 // OpenGL header file for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2011 by Bill Spitzak and others.
00007 //
00008 // You must include this instead of GL/gl.h to get the Microsoft
00009 // APIENTRY stuff included (from <windows.h>) prior to the OpenGL
00010 // header files.
00011 //
00012 // This file also provides "missing" OpenGL functions, and
00013 // gl_start() and gl_finish() to allow OpenGL to be used in any window
00014 //

Generated by Doxygen
ifndef FL_gl_H
#define FL_gl_H

#include "Enumerations.H" // for color names

#ifdef WIN32
#include <windows.h>
#endif
#ifndef APIENTRY
#ifndef __CYGWIN__
#define APIENTRY __attribute__ ((__stdcall__))
#else
#define APIENTRY
#endif
#endif

#ifdef __APPLE__
#include <OpenGL/gl.h>
#else
#include <GL/gl.h>
#endif // __APPLE__

FL_EXPORT void gl_start();
FL_EXPORT void gl_finish();

FL_EXPORT void gl_color(Fl_Color i);
inline void gl_color(int c) {gl_color((Fl_Color)c);}

FL_EXPORT void gl_rect(int x,int y,int w,int h);
inline void gl_rectf(int x,int y,int w,int h) {glRecti(x,y,x+w,y+h);}

FL_EXPORT void gl_font(int fontid, int size);
FL_EXPORT int gl_height();
FL_EXPORT int gl_descent();
FL_EXPORT double gl_width(const char *);
FL_EXPORT double gl_width(const char *, int n);
FL_EXPORT double gl_width(uchar);

FL_EXPORT void gl_draw(const char*);
FL_EXPORT void gl_draw(const char*, int n);
FL_EXPORT void gl_draw(const char*, int x, int y);
FL_EXPORT void gl_draw(const char*, float x, float y);
FL_EXPORT void gl_draw(const char*, int n, int x, int y);
FL_EXPORT void gl_draw(const char*, int n, float x, float y);
FL_EXPORT void gl_draw(const char*, int x, int y, int w, int h, Fl_Align);
FL_EXPORT void gl_measure(const char*, int x, int y);

#ifdef __APPLE__
extern FL_EXPORT void gl_texture_pile_height(int max);
#else
extern FL_EXPORT int gl_texture_pile_height();
#endif

FL_EXPORT void gl_draw_image(const uchar *, int x,int y,int w,int h, int d=3, int ld=0);

Fl_EXPORT int gl_texture_pile_height();

#endif // !FL_gl_H

// End of "$Id$".

// 10.160 gl2opengl.h

/* gl.h
If you include this, you might be able to port old GL programs.
There are also much better emulators available on the net.
*/

#include <FL/gl.h>
#include "gl_draw.H"

inline void clear() {glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);}
#define RGBcolor(r,g,b) glColor3ub(r,g,b)
#define bgnline() glBegin(GL_LINE_STRIP)

Generated by Doxygen
// $Id$
// OpenGL 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:
// http://www.fltk.org/COPYING.php
// Please report all bugs and problems on the following page:
// http://www.fltk.org/str.php

#include "gl.h"

extern FL_EXPORT void gl_remove_displaylist_fonts();

// End of "$Id$".

#ifndef FL_glu_H
#define FL_glu_H

#include "Enumerations.H" // for color names

#define bgnpolygon() glBegin(GL_POLYGON)
#define bgnclosedline() glBegin(GL_LINE_LOOP)
#define endline() glEnd()
#define endpolygon() glEnd()
#define endclosedline() glEnd()
#define v2f(v) glVertex2fv(v)
#define v2s(v) glVertex2sv(v)
#define cmov(x,y,z) glRasterPos3f(x,y,z)
#define charstr(s) gl_draw(s)
#define fmprstr(s) gl_draw(s)
typedef float Matrix[4][4];
inline void pushmatrix() {glPushMatrix();}
inline void popmatrix() {glPopMatrix();}
inline void multmatrix(Matrix m) {glMultMatrixf((float *)m);}
inline void color(int n) {glIndexi(n);}
inline void rect(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectf(int x,int y,int r,int t) {glRectf(x,y,r+1,t+1);}
inline void recti(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectfi(int x,int y,int r,int t) {glRecti(x,y,r+1,t+1);}
inline void rects(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectfs(int x,int y,int r,int t) {glRects(x,y,r+1,t+1);}
10.163 glut.H

---

Commented out lines indicate parts of GLUT that are not emulated.

---

#include "gl.h"
#include "Fl.H"
#include "Fl_Gl_Window.H"

class FL_EXPORT Fl_Glut_Window : public Fl_Gl_Window {
  public: // so the inline functions work
    int number;
    int menu[3];
    void make_current();
    void (display)();
    void (overlaydisplay)();
    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);
00065  FL_Glut_Window(int w, int h, const char *);  
00066  FL_Glut_Window(int x, int y, int w, int h, const char *);  
00067  ~FL_Glut_Window();  
00068  
00069  extern FL_EXPORT Fl_Glut_Window *glut_window;  // the current window  
00070  extern FLEXPORT int glut_menu;  // the current menu  
00071  
00072  // function pointers that are not per-window:  
00073  extern FL_EXPORT void (*glut_idle_function)();  
00074  extern FL_EXPORT void (*glut_menustate_function)(int);  
00075  extern FL_EXPORT void (*glut_menustatus_function)(int,int,int);  
00076  
00077  
00079  
00080  // # define GLUT_API_VERSION This does not match any version of GLUT exactly...  
00081  
00082  FL_EXPORT void glutInit(int *argcp, char **argv);  // creates first window  
00083  
00084  FL_EXPORT void glutInitDisplayMode(unsigned int mode);  
00085  // the FL_ symbols have the same value as the GLUT ones:  
00086  # define GLUT_RGB FL_RGB  
00087  # define GLUT_RGBA FL_RGB  
00088  # define GLUT_INDEX FL_INDEX  
00089  # define GLUT_SINGLE FL_SINGLE  
00090  # define GLUT_DOUBLE FL_DOUBLE  
00091  # define GLUT_ACCUM FL_ACCUM  
00092  # define GLUT_ALPHA FL_ALPHA  
00093  # define GLUT_DEPTH FL_DEPTH  
00094  # define GLUT_STENCIL FL_STENCIL  
00095  # define GLUT_MULTISAMPLE FL_MULTISAMPLE  
00096  # define GLUT_STEREO FL_STEREO  
00097  // # define GLUT_LUMINANCE 512  
00098  
00099  FL_EXPORT void glutInitWindowPosition(int x, int y);  
00100  
00101  FL_EXPORT void glutInitWindowSize(int w, int h);  
00102  
00103  FL_EXPORT void glutMainLoop();  
00104  
00105  FL_EXPORT int glutCreateWindow(char *title);  
00106  FL_EXPORT int glutCreateWindow(const char *title);  
00107  
00108  FL_EXPORT int glutCreateSubWindow(int win, int x, int y, int width, int height);  
00109  
00110  FL_EXPORT void glutDestroyWindow(int win);  
00111  
00112  inline void glutPostRedisplay() {glut_window->redraw();}  
00113  
00114  FL_EXPORT void glutPostWindowRedisplay(int win);  
00115  
00116  FL_EXPORT void glutSwapBuffers();  
00117  
00118  inline int glutGetWindow() {return glut_window->number;}  
00119  
00120  FL_EXPORT void glutSetWindow(int win);  
00121  
00122  inline void glutSetWindowTitle(char *t) {glut_window->label(t);}  
00123  
00124  inline void glutSetIconTitle(char *t) {glut_window->iconlabel(t);}  
00125  
00126  inline void glutPositionWindow(int x, int y) {glut_window->position(x,y);}  
00127  
00128  inline void glutReshapeWindow(int w, int h) {glut_window->size(w,h);}  
00129  
00130  inline void glutPopWindow() {glut_window->show();}  
00131  
00132  inline void glutPushWindow() { /* do nothing */}  
00133  
00134  inline void glutIconifyWindow() {glut_window->iconize();}  
00135  
00136  inline void glutShowWindow() {glut_window->show();}  
00137  
00138  inline void glutHideWindow() {glut_window->hide();}  
00139  
00140  inline void glutFullScreen() {glut_window->fullscreen();}  
00141  
00142  inline void glutSetCursor(Fl_Cursor cursor) {glut_window->cursor(cursor);}  
00143  // notice that the numeric values are different than glut:  
00144  # define GLUT_CURSOR_RIGHT_ARROW ((Fl_Cursor)2)  
00145  # define GLUT_CURSOR_LEFT_ARROW ((Fl_Cursor)67)  
00146  # define GLUT_CURSOR_INFO ((Fl_Cursor)45)  
00147  # define GLUT_CURSOR_DESTROY ((Fl_Cursor)45)  
00148  # define GLUT_CURSOR_HELP (Fl_Cursor)45  
00149  # define GLUT_CURSOR_CYCLE (Fl_Cursor)63  
00150  # define GLUT_CURSOR_SPRAY (Fl_Cursor)63  
00151  # define GLUT_CURSOR_WAIT (Fl_Cursor)63  
00152  # define GLUT_CURSOR_TEXT (Fl_Cursor)63
# 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() {glut_window->make_overlay_current();}
inline void glutRemoveOverlay() {glut_window->hide_overlay();}
inline void glutUseLayer(GLenum layer) {
  layer ? glut_window->make_overlay_current() : glut_window->make_current();
enum {GLUT_NORMAL, GLUT_OVERLAY};
inline void glutPostOverlayRedisplay() {glut_window->redraw_overlay();}
inline void glutShowOverlay() {glut_window->redraw_overlay();}
inline void glutHideOverlay() {glut_window->hide_overlay();}
FL_EXPORT int glutCreateMenu(void (*)(int));
FL_EXPORT void glutDestroyMenu(int menu);
inline int glutGetMenu() {return glut_menu;}
inline void glutSetMenu(int m) {glut_menu = m;}
FL_EXPORT void glutAddMenuEntry(char *label, int value);
FL_EXPORT void glutAddSubMenu(char *label, int submenu);
FL_EXPORT void glutChangeToMenuEntry(int item, char *labela, int value);
FL_EXPORT void glutChangeToSubMenu(int item, char *label, int submenu);
FL_EXPORT void glutRemoveMenuItem(int item);
inline void glutAttachMenu(int b) {glut_window->menu[b] = glut_menu;}
inline void glutDetachMenu(int b) {glut_window->menu[b] = 0;}
inline void glutDisplayFunc(void (*f)()) {glut_window->display = f;}
inline void glutReshapeFunc(void (*f)(int w, int h)) {glut_window->reshape = f;}
inline void glutKeyboardFunc(void (*f)(uchar key, int x, int y)) {
  glut_window->keyboard = f;}
inline void glutMouseFunc(void (*f)(int b, int state, int x, int y)) {
  glut_window->mouse = 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)) {glut_window->motion = f;}
inline void glutPassiveMotionFunc(void (*f)(int x, int y)) {
  glut_window->passivemotion = f;}
inline void glutEntryFunc(void (*f)(int s)) {glut_window->entry = f;}
enum {GLUT_LEFT, GLUT_ENTERED};
inline void glutVisibilityFunc(void (*f)(int state)) {
  glut_menustate_function = f;}
FL_EXPORT void glu

Generated by Doxygen
inline void glutMenuStatusFunc(void (*)(int status, int x, int y)) {
    glut_menu_status_function = f;
}

enum {GLUT_MENU_NOT_IN_USE, GLUT_MENU_IN_USE};

inline void glutSpecialFunc(void (*)(int key, int x, int y)) {
    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 (*)(f)()) {
    glut_window->overlaydisplay = f;
}

//inline void glutWindowStatusFunc(void (*)(int state));

//enum {GLUT_HIDDEN, GLUT_FULLY_RETAINED, GLUT_PARTIALLY_RETAINED,
//      GLUT_FULLY_COVERED};

//inline void glutSetColor(int ndx, float red, float green, float blue);

//inline float 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 {
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
    GL_FALSE = 0,
    GL_TRUE = 1,
# define GLUT_WINDOW_STENCIL_SIZE GL_STENCIL_BITS
# define GLUT_WINDOW_DEPTH_SIZE GL_DEPTH_BITS
# define GLUT_WINDOW_RED_SIZE GL_RED_BITS
# define GLUT_WINDOW_GREEN_SIZE GL_GREEN_BITS
# define GLUT_WINDOW_BLUE_SIZE GL_BLUE_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

FL_EXPORT int glutDeviceGet(GLenum type);

// WARNING: these values are different than GLUT uses:
#define GLUT_ACTIVE_SHIFT FL_SHIFT
#define GLUT_ACTIVE_CTRL FL_CTRL
#define GLUT_ACTIVE_ALT FL_ALT
inline int glutGetModifiers() {return Fl::event_state() & (GLUT_ACTIVE_SHIFT | GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT);}

FL_EXPORT int glutLayerGet(GLenum);
#define GLUT_OVERLAY_POSSIBLE 800
#define GLUT_LAYER_IN_USE 801
#define GLUT_HAS_OVERLAY 802
#define GLUT_TRANSPARENT_INDEX 803
#define GLUT_NORMAL_DAMAGED 804
#define GLUT_OVERLAY_DAMAGED 805

//inline int glutVideoResizeGet(GLenum param);
#define GLUT_VIDEO_RESIZE_POSSIBLE 900
#define GLUT_VIDEO_RESIZE_IN_USE 901
#define GLUT_VIDEO_RESIZE_X_DELTA 902
#define GLUT_VIDEO_RESIZE_Y_DELTA 903
#define GLUT_VIDEO_RESIZE_WIDTH_DELTA 904
#define GLUT_VIDEO_RESIZE_HEIGHT_DELTA 905
#define GLUT_VIDEO_RESIZE_X 906
#define GLUT_VIDEO_RESIZE_Y 907
#define GLUT_VIDEO_RESIZE_WIDTH 908
#define GLUT_VIDEO_RESIZE_HEIGHT 909

//inline void glutSetupVideoResizing();

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)
FL_EXPORT void glutBitmapCharacter(void *, int character);
FL_EXPORT int glutBitmapHeight(void *, int character);
FL_EXPORT void glutBitmapString(void *, const unsigned char *, string);
FL_EXPORT int glutBitmapWidth(void *, int character);
FL_EXPORT int glutExtensionSupported(char *, name);
FL_EXPORT void glutStrokeCharacter(void *, int character);
FL_EXPORT GLfloat glutStrokeHeight(void *, int character);
FL_EXPORT int glutStrokeLength(void *, const unsigned char *, string);
FL_EXPORT void glutStrokeString(void *, const unsigned char *, string);
FL_EXPORT int glutStrokeWidth(void *, int character);

/* GLUT stroke 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);

/* 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();

#ifndef // !Fl_glut_H
#endif // End of "$Id$".

Mac OS X-specific symbols.

Classes

- class Fl_Mac_App_Menu
  Mac OS-specific class allowing to customize and localize the application menu.
Functions

- 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.

- void fl_open_callback (void (∗ cb)(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)

- int fl_mac_quit_early
  
  Determines whether cmd-Q or the "Quit xxx" item of application menu terminates the app or only the event loop.

- class Fl_Sys_Menu_Bar ∗ fl_sys_menu_bar
  
  The system menu bar.

10.164.1 Detailed Description

Mac OS X-specific symbols.

10.165 mac.H

Go to the documentation of this file.
#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_X_VERSION_MAX_ALLOWED < MAC_OS_X_VERSION_10_4
typedef CGImageAlphaInfo CGBitmapInfo;
#endif

typedef struct flCocoaRegion {
    int count;
    CGRect *rects;
} *Fl_Region; // a region is the union of a series of rectangles

#include "Fl_Window.H"
#include "Fl_Font.H"
struct XPoint { int x, y; };  
struct XRectangle { int x, y, width, height; };  

#ifndef CGFLOAT_DEFINED // appears with 10.5 in CGBase.h
#if defined(__LP64__) && __LP64__
typedef double CGFloat;  
#else  
typedef float CGFloat;  
#endif  
#endif // CGFLOAT_DEFINED  

CGRect fl_cgrectmake_cocoa(int x, int y, int w, int h);  
inline CGRect fl_cgrectmake_cocoa(int x, int y, int w, int h) {  
CGRect R = (CGRect)malloc(sizeof(R));  
R->count = 1;  
R-> rects = (CGRect *)malloc(sizeof(CGRect));  
*(R->rects) = fl_cgrectmake_cocoa(x, y, w, h);  
return R;  
}  

extern CGRect fl_cgrectmake_cocoa(int x, int y, int w, int h);  
inline Fl_Region XRectangleRegion(int x, int y, int w, int h) {  
Fl_Region R = (Fl_Region)malloc(sizeof( *R));  
R->count = 1;  
R->rects = (CGRect *)malloc(sizeof(CGRect));  
*(R->rects) = fl_cgrectmake_cocoa(x, y, w, h);  
return R;  
}  

Fl_Region XRectangleRegion(int x, int y, int w, int h) {  
Fl_Region R = (Fl_Region)malloc(sizeof( *R));  
R->count = 1;  
R->rects = (CGRect *)malloc(sizeof(CGRect));  
*(R->rects) = fl_cgrectmake_cocoa(x, y, w, h);  
return R;  
}  

extern NSCursor *fl_default_cursor;  

This object contains all mac-specific stuff about a window:  
WARNING: this object is highly subject to change!  
class Fl_X {  
    Window xid; // pointer to the Cocoa window object (FLWindow *)  
    Fl_Offscreen other_xid; // pointer for offscreen bitmaps (overlay window)  
    Fl_Window *w; // FLTK window for  
    Fl_Region region;  
    #if FLTK_ABI_VERSION < 10304  
    Fl_Region subRegion; // for ABI compatibility, recycled to replace subRect_  
    #endif  
    Fl_X *next; // chain of mapped windows  
    #if FLTK_ABI_VERSION < 10304  
    Fl_X *xidChildren; // useless with true subwindows, recycled to replace mapped_to_retina_  
    Fl_X *xidNext; // useless with true subwindows  
    #endif  
    int wait_for_expose;  
    NSCursor *cursor;  
    static Fl_X * first;  
    static Fl_X * i(const Fl_Window* w) {return w->i;}  
    static int fake_X_wm(const Fl_Window *,int&,int&,int&,int&,int&,int,int,int,int,int);  
    static void make(Fl_Window *);  
    void flush();  
    static void set_high_resolution(bool);  
    #if FLTK_ABI_VERSION >= 10304  
    CGRect * subRect() { return subRect_; } // getter  
    void subRect(CGRect *r) { subRect_ = r; } // setter  
    #else  
    CGRect * subRect() { return (CGRect*)subRegion; } // getter  
    void subRect(CGRect *r) { subRegion = (Fl_Region)r; } // setter  
    #endif  
    bool mapped_to_retina(); // is window mapped to retina display?  
    void mapped_to_retina(bool); // sets whether window is mapped to retina display  
    bool changed_resolution(); // did window just moved to display with another resolution?  
    void changed_resolution(bool); // sets whether window just moved to display with another resolution  
    bool in_windowDidResize(); // is window performing windowDidResize?  
    void in_windowDidResize(bool); // sets whether window is performing windowDidResize  
    // Quartz additions:  
    CGContextRef gc; // graphics context (NULL when using QD)  
    static void q_fill_context(); // fill a Quartz context with current FLTK state  
    static void q_clear_clipping(); // remove all clipping from a Quartz context  
    static void q_release_context(Fl_X *x=0); // free all resources associated with fl_gc  
    static void q_begin_image(CGRect*, int x, int y, int w, int h);  
    static void q_end_image();  
    // Cocoa additions  
    static NSOpenGLPixelFormat *mode_to_NSOpenGLPixelFormat(int mode, const int*); // computes NSOpenGLPixelFormat from G1 window's mode  
    static NSOpenGLContext* create_GLcontext_for_window(NSOpenGLPixelFormat *pixelformat,  
NSOpenGLContext *shared_ctx, Fl_X *window);  
    static void q_context_update(NSOpenGLContext*);  
    static void q_context_flushbuffer(NSOpenGLContext*);  
    static void q_context_release(NSOpenGLContext*);  
    static void q_context_makecurrent(NSOpenGLContext*);  
    static void GL_cleardrawable(void);  
    static void gl_start(NSOpenGLContext *);  
    void destroy(void);  
    void map(void);  
    void unmap(void);  
    void collapse(void);
static unsigned char *bitmap_from_window_rect(Fl_Window *win, int x, int y, int w, int h, int *bytesPerPixel);
static Fl_Region intersect_region_and_rect(Fl_Region current, int x, int y, int w, int h);
static void draw_layer_to_context(void *layer, CGContextRef ctxt, int w, int h);
private:
  CGRect * subRect_; // makes sure subwindow remains inside its parent window
  unsigned mapped_to_retina_; // stores 3 binary flags: whether window is mapped to retina display; whether resolution just changed; whether window is OpenGL and is currently being resized.
#if FLTK_ABI_VERSION >= 10304
#else
  bool subwindow; // for ABI compatibility, useless with true subwindows
#endif // FLTK_ABI_VERSION >= 10304
#endif // FL_LIBRARY || FL_INTERNALS
extern Window fl_window;
#endif // FL_DOXYGEN
extern void fl_open_callback(void (*cb)(const char *));
extern void fl_mac_set_about( Fl_Callback *cb, void *user_data, int shortcut = 0);
extern int fl_mac_os_version;
extern int fl_mac_quit_early;
extern class Fl_Sys_Menu_Bar *fl_sys_menu_bar;
struct Fl_Menu_Item;
class Fl_Mac_App_Menu {
public:
  static const char * about;
  static const char * print;
  static const char * services;
  static const char * hide;
  static const char * hide_others;
  static const char * show;
  static const char * quit;
  static void custom_application_menu_items(const Fl_Menu_Item *m);
};
Generated by Doxygen
10.166  math.h

#include <math.h>

#ifndef M_PI
#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
#endif // !M_PI

#ifndef M_SQRT2
#define M_SQRT2 1.41421356237309504880
#define M_SQRT1_2 0.70710678118654752440
#endif // !M_SQRT2

if (defined(WIN32) || defined(CRAY)) && !defined(__MINGW32__) && !defined(__MWERKS__)

inline double rint(double v) {return floor(v+.5);}
inline double copysign(double a, double b) {return b<0 ? -a : a;}
#endif // fl_math_h

10.167  names.h

#ifndef M_SQRT2
#define M_SQRT2 1.41421356237309504880
#define M_SQRT1_2 0.70710678118654752440
#endif // !M_SQRT2

if (defined(WIN32) || defined(CRAY))

if defined(__MINGW32__) && !defined(__MWERKS__)

inline double rint(double v) {return floor(v+.5);}
inline double copysign(double a, double b) {return b<0 ? -a : a;}
#endif // fl_math_h
#ifndef FL_NAMES_H
#define FL_NAMES_H

const char * const fl_eventnames[] =
{
  "FL_NO_EVENT",
  "FL_PUSH",
  "FL_RELEASE",
  "FL_ENTER",
  "FL_LEAVE",
  "FL_DRAG",
  "FL_FOCUS",
  "FL_UNFOCUS",
  "FL_KEYDOWN",
  "FL_KEYUP",
  "FL_CLOSE",
  "FL_MOVE",
  "FL_SHORTCUT",
  "FL_DEACTIVATE",
  "FL_ACTIVATE",
  "FL_HIDE",
  "FL_SHOW",
  "FL_PASTE",
  "FL_SELECTIONCLEAR",
  "FL_MOUSEWHEEL",
  "FL_DND_ENTER",
  "FL_DND_DRAG",
  "FL_DND_LEAVE",
  "FL_DND_RELEASE",
  "FL_DND_STACK",
  "FL_SCREEN_CONFIGURATION_CHANGED",
  "FL_FULLSCREEN",
  "FL_ZOOM_GESTURE",
  "FL_EVENT_27", // not yet defined, just in case they /will/ be defined ...
  "FL_EVENT_28",
  "FL_EVENT_29",
  "FL_EVENT_30"
};

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",
};
#endif /* FL_NAMES_H */

// End of "$Id$".
This file is present for compatibility with FLTK 1.4 and later. In FLTK 1.4 FL/platform.H replaces FL/x.H. FLTK 1.4 code that includes FL/platform.H instead of FL/x.H can now be compiled with FLTK 1.3.5 and later versions.

```c
#ifndef Fl_X_H
#include <FL/x.H>
#endif
```

The file is present for compatibility with FLTK 1.4 and later. In FLTK 1.4 FL/platform.H replaces FL/x.H. FLTK 1.4 code that includes FL/platform.H instead of FL/x.H can now be compiled with FLTK 1.3.5 and later versions.

```c
#ifndef FL_DOXYGEN
#ifndef Fl_X_H
#error "Never use <FL/win32.H> directly; include <FL/x.H> instead."
#endif // !Fl_X_H
```

The file is present for compatibility with FLTK 1.4 and later. In FLTK 1.4 FL/platform.H replaces FL/x.H. FLTK 1.4 code that includes FL/platform.H instead of FL/x.H can now be compiled with FLTK 1.3.5 and later versions.

```c
#define LPMINMAXINFO MINMAXINFO*
#define VK_LWIN 0x5B
#define VK_RWIN 0x5C
#define VK_APPS 0x5D
```

```c
struct XRectangle {int x, y, width, height;};
extern Fl_Region XRectangleRegion(int x, int y, int w, int h);
```

```c
inline void XDestroyRegion(Fl_Region r) {DeleteObject(r);}
inline void XClipBox(Fl_Region r, XRectangle* rect) {
  RECT win_rect; GetRgnBox(r, &win_rect);
  rect->x=win_rect.left;
  rect->y=win_rect.top;
  rect->width=win_rect.right-win_rect.left;
  rect->height=win_rect.bottom-win_rect.top;
}
```

```c
#define XDestroyWindow(a,b) DestroyWindow(b)
#define XMapWindow(a,b) ShowWindow(b, SW_RESTORE)
```
```c
// this object contains all win32-specific stuff about a window:
// Warning: this object is highly subject to change!

class Fl_EXPORT Fl_X {
public:
    // member variables - add new variables only at the end of this block
    Window xid;
    HBITMAP other_xid; // for double-buffered windows
    Fl_Window * w;
    Fl_Region region;
    Fl_X *next;
    int wait_for_expose;
    HDC private_dc; // used for OpenGL
    HCURSOR cursor;
    int custom_cursor;
    HDC saved_hdc; // saves the handle of the DC currently loaded
    // static variables, static functions and member functions
    static Fl_X * first;
    static Fl_X * i(const Fl_Window* w) {return w->i;}
    static int fake_X_wm(const Fl_Window * w, int &X, int &Y, int &bt, int &bx, int &by);
    void make_fullscreen(int X, int Y, int W, int H);
    void setwindow(Fl_Window * wi) {w=wi; wi->i=this;}
    void flush() {w->flush();}
    void set_minmax(LPMINMAXINFO minmax);
    void mapraise();
    static void set_default_icons(const Fl_RGB_Image *, int);
    static void set_default_icons(HICON, HICON);
    void set_icons();
    int set_cursor(Fl_Cursor);
    int set_cursor(const Fl_RGB_Image *, int, int);
    static Fl_X * make(Fl_Window*);
};

extern FL_EXPORT UINT fl_wake_msg;
extern FL_EXPORT char fl_override_redirect; // hack into Fl_Window::make_xid()
extern FL_EXPORT int fl_background_pixel; // hack into Fl_Window::make_xid()
extern FL_EXPORT HPALETTE fl_palette; // non-zero only on 8-bit displays!
extern FL_EXPORT void fl_release_dc(HWND w, HDC dc);
extern FL_EXPORT void fl_save_dc(HWND w, HDC dc);

inline Window fl_xid(const Fl_Window* w) { Fl_X *temp = Fl_X::i(w); return temp ? temp->xid : 0; }

extern FL_EXPORT Fl_Display_Device *fl_display;
extern FL_EXPORT Window fl_window;
extern FL_EXPORT HDC fl_gc;
extern FL_EXPORT MSG fl_msg;
extern FL_EXPORT HDC fl_GetDC(Window);
extern FL_EXPORT HDC fl_makeDC(HBITMAP);

// off-screen pixmaps: create, destroy, draw into, copy to window
typedef HBITMAP Fl_Offscreen;

#define fl_create_offscreen(w, h) CreateCompatibleBitmap((fl_gc ? fl_gc : fl_GetDC(0)), w, h)
define fl_begin_offscreen(b) HDC _sgc=fl_gc; Window _sw=fl_window; hdc = fl_makeDC(b); int _savedc = SaveDC(hdc); fl_window=(HWND)b; fl_push_no_clip()
define fl_end_offscreen(); fl_pop_clip(); RestoreDC(hdc, _savedc); DeleteDC(hdc); _ss->set_current(); fl_window=_sw; hdc = _sgc
```
00146 FL_EXPORT void fl_copy_offscreen(int x, int y, int w, int h, HBITMAP pixmap, int srcx, int srcy);
00147 #define fl_delete_offscreen(bitmap) DeleteObject(bitmap)
00148
00149 // Bitmap masks
00150 typedef HBITMAP Fl_Bitmask;
00151
00152 extern FL_EXPORT Fl_Bitmask fl_create_bitmask(int w, int h, const uchar *data);
00153 extern FL_EXPORT Fl_Bitmask fl_create_alphamask(int w, int h, int d, int ld, const uchar *data);
00154 extern FL_EXPORT void fl_delete_bitmask(Fl_Bitmask bm);
00155
00156 // Dummy function to register a function for opening files via the window manager...
00157 inline void fl_open_callback(void (*)(const char *)) {};
00158
00159 extern FL_EXPORT int fl_parse_color(const char* p, uchar& r, uchar& g, uchar& b);
00160 #endif // FL_DOXYGEN
00161 //
00162 // End of "$Id$".
00163 //

10.170 x.H

0001 //
0002 // X11 header file for the Fast Light Tool Kit (FLTK).
0003 //
0004 // Copyright 1998-2023 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 // http://www.fltk.org/COPYING.php
0011 //
0012 // Please report all bugs and problems on the following page:
0013 //
0014 // http://www.fltk.org/str.php
0015 //
0016 //
0017 // These are internal fltk symbols that are necessary or useful for
0018 // calling Xlib. You should include this file if (and ONLY if) you
0019 // need to call Xlib directly. These symbols may not exist on non-X
0020 // systems.
0021 #if !defined(Fl_X_H) && !defined(FL_DOXYGEN)
0022 # define Fl_X_H
0023 # include "Enumerations.H"
0024 # define FL_X_H
0025 # include "Enumerations.H"
0026 #endif
0027 #ifdef WIN32
0028 # include "win32.H"
0029 #elif defined(__APPLE__)
0030 # include "mac.H"
0031 #else
0032 # if defined(_ABIN32) || defined(_ABIN64) // fix for broken SGI Irix X .h files
0033 # pragma set woff 3322
0034 # endif
0035 # include "<X11/Xlib.h>"
0036 # include "<X11/Xutil.h>"
0037 # if defined(_ABIN32) || defined(_ABIN64)
0038 # pragma reset woff 3322
0039 # endif
0040 # include "<X11/Xatom.h>"
0041 # include "Fl_Window.H"
0042 // Mirror X definition of Region to Fl_Region, for portability...
0043 typedef Region Fl_Region;
0044
0045 FL_EXPORT void fl_open_display();
0046 FL_EXPORT void fl_open_display(Display*);
0047 FL_EXPORT void fl_close_display();
0048
0049 // constant info about the X server connection:
0050 extern FL_EXPORT Display* fl_display;
0051 extern FL_EXPORT int fl_screen;
0052 extern FL_EXPORT XVisualInfo fl_visual;
0053 extern FL_EXPORT Colormap fl_colormap;
0054
0055
0056 // drawing functions:
0057 extern FLEXPORT GC fl_gc;
0058 extern FLEXPORT Window fl_window;
0059 FLEXPORT ulong fl_xpixel(Fl_Color i);
0060 FLEXPORT ulong fl_xpixel(uchar r, uchar g, uchar b);
0061 FLEXPORT void fl_clip_region(Fl_Region);
0062 FLEXPORT Fl_Region fl_clip_region();
0063
0064 // feed events into fltk:

Generated by Doxygen
FL_EXPORT int fl_handle(const XEvent&);

You can use these in Fl::add_handler() to look at events:

extern FL_EXPORT const XEvent* fl_xevent;
extern FL_EXPORT ulong fl_event_time;

// off-screen pixmaps: create, destroy, draw into, copy to window:
typedef ulong Fl_Offscreen;
#define fl_create_offscreen(w,h) XCreatePixmap(fl_display, RootWindow(fl_display, fl_screen), w, h, fl_visual->depth)
#define fl_create_offscreen_with_alpha(w,h) XCreatePixmap(fl_display, RootWindow(fl_display, fl_screen), w, h, 32)

// begin/end are macros that save the old state in local variables:
#define fl_begin_offscreen(pixmap)
  Window _sw=fl_window; fl_window=pixmap;
  GC _sgc = fl_gc; if (!_sgc) fl_gc = XCreateGC(fl_display, pixmap, 0, 0);
  Fl_Surface_Device::_ss = Fl_Surface_Device::surface();
  Fl_Display_Device::display_device()->set_current();
  fl_push_no_clip();
#define fl_end_offscreen()
  fl_pop_clip(); fl_window = _sw; _ss->set_current();
  if (!_sgc) XFreeGC(fl_display, fl_gc);
  fl_gc = _sgc

extern FL_EXPORT void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
#define fl_delete_offscreen(pixmap) XFreePixmap(fl_display, pixmap)

// Bitmap masks

typedef ulong Fl_Bitmask;

extern FL_EXPORT Fl_Bitmask fl_create_bitmask(int w, int h, const uchar *data);
extern FL_EXPORT Fl_Bitmask fl_create_alphamask(int w, int h, int d, int ld, const uchar *data);
extern FL_EXPORT void fl_delete_bitmask(Fl_Bitmask bm);

# if defined(FL_LIBRARY) || defined(FL_INTERNALS)
  extern FL_EXPORT Window fl_message_window;
  extern FL_EXPORT void *fl_xftfont;
  FL_EXPORT Fl_Region XRectangleRegion(int x, int y, int w, int h); // in fl_rect.cxx
# endif

// access to core fonts:

// This class provides a "smart pointer" that returns a pointer to an XFontStruct.
// The global variable fl_xfont can be called wherever a bitmap "core" font is
// needed, e.g. when rendering to a GL context under X11.
// With XFT / X11 fonts, fl_xfont will attempt to return the current selected font.
// With XFT / X11 fonts, fl_xfont will attempt to return the bitmap "core" font most
// similar to (usually the same as) the current XFT font.

class Fl_XFont_On_Demand
{
  public:
    Fl_XFont_On_Demand(XFontStruct * p = NULL) : ptr(p) { }
    Fl_XFont_On_Demand& operator=(const Fl_XFont_On_Demand& x)
    { ptr = x.ptr; return *this; }
    Fl_XFont_On_Demand& operator=(XFontStruct * p)
    { ptr = p; return *this; }
    XFontStruct* value();
    operator XFontStruct *() { return value(); }
    XFontStruct& operator *() { return *value(); }
    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;
    }

// this object contains all X-specific stuff about a window:
// Warning: this object is highly subject to change!
// FL_LIBRARY or FL_INTERNALS must be defined to access this class.

class FL_EXPORT Fl_X
{
  public:
    Window xid;
    Window other_xid;
    Fl_Window* w;
    Fl_Display_Device* x;
    Fl_Region Region;
    Fl_X* next;
    char wait_for_expose;
    char backbuffer_bad; // used for XDBE
    static Fl_X* first;
    static Fl_X* i(const Fl_Window* wi) {return wi->i;}
    void setwindow(Fl_Window* wi) {wi->i=this;}
    void sendxjunk();
    static void set_default_icons(const Fl_RGB_Images*, int);
    void set_icons();
    int set_cursor(Fl_Cursor);
    int set_cursor(const Fl_RGB_Images*, int, int);
    int set_cursor(Fl_Cursor);
    int set_cursor(const Fl_RGB_Images*, int, int);
00149  static void make_xid(Fl_Window *, XVisualInfo* = fl_visual, Colormap=fl_colormap);
00149  static Fl_X* set_xid(Fl_Window*, Window);
00150  // kludges to get around protection:
00151  void flush() {w->flush();}
00152  static void x(Fl_Window* wi, int X) {wi->x(X);}
00153  static void y(Fl_Window* wi, int Y) {wi->y(Y);}
00154  static int ewmh_supported();
00155  static int xrender_supported();
00156  static void activate_window(Window w);
00157  };
00158
00159 extern FL_EXPORT char fl_override_redirect; // hack into Fl_X::make_xid()
00160 extern FL_EXPORT int fl_background_pixel; // hack into Fl_X::make_xid()
00161
00162 inline Window fl_xid(const Fl_Window* w) { Fl_X *xTemp = Fl_X::i(w); return xTemp ? xTemp->xid : 0; }
00163
00164 #else
00165
00166 extern FL_EXPORT Window fl_xid_(const Fl_Window* w);
00167 #define fl_xid(w) fl_xid_(w)
00168
00169 #endif // FL_LIBRARY || FL_INTERNALS
00170
00171 FL_EXPORT Fl_Window* fl_find(Window xid);
00172
00173 // Dummy function to register a function for opening files via the window manager...
00174 inline void fl_open_callback(void (*)(const char *)) {}
00175
00176 extern FL_EXPORT int fl_parse_color(const char* p, uchar& r, uchar& g, uchar& b);
00177
00178 #endif
00179 #endif
00180

10.171  cgdebug.h

00001 //
00002 // "$Id$"
00003 //
00004 // OS X Core Graphics debugging help for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2010 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 // This file has proven to be tremendously useful in debugging
00020 // code. This file is normally not included into any FLTK builds,
00021 // but since it has proven to be tremendously useful in debugging
00022 // the FLTK port to "Quartz", I decided to add this file in case
00023 // more bugs show up.
00024 //
00025 // This header is activated by adding the following
00026 // line to "config.h"
00027 //
00028 // #include "src/cgdebug.h"
00029 //
00030 // When used erroneously, Core Graphics prints warnings to
00031 // stderr. This is helpful, however it is not possible to
00032 // associate a line number or source file with the warning message.
00033 // This header file outputs a trace of CG calls, interweaving
00034 // them with CG warnings.
00035 //
00036 //
00037 // Matthias
00038
00039 #ifndef CGDEBUG
00040 #define CGDEBUG
00041
00042 //+#include <stdio.h>
00043 #include <Carbon/Carbon.h>
00044
00045 +//BitmapContextCreate
00046 +//BitmapContextGetData
00047 +//ClipCGContextToRegion
00048 +//QBeginCGContext
00049 +//QEndCGContext

Generated by Doxygen
10.171 cgdebug.h

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
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136

//+AddArc
//+AddLineToPoint
// ClipToRect
// ClosePath
//+ConcatCTM
//+DrawImage
// FillPath
// FillRect
// Flush
//+GetCTM
// MoveToPoint
//+Release
// RestoreGState
// SaveGState
//+ScaleCTM
//+SetLineCap
//+SetLineDash
//+SetLineJoin
//+SetLineWidth
//+SetRGBFillColor
//+SetRGBStrokeColor
//+SetShouldAntialias
//+SetTextMatrix
//+StrokePath
//+TranslateCTM
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__); \

Generated by Doxygen

1833


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) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
    dbgCGContextFillPath(a); \
    fprintf(stderr, "n"); }

inline void dbgCGContextClosePath(CGContextRef context)
{
    CGContextClosePath(context);
}

#define CGContextClosePath(a) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
    dbgCGContextClosePath(a); \
    fprintf(stderr, "n"); }

inline void dbgCGContextFlush(CGContextRef context)
{
    CGContextFlush(context);
}

#define CGContextFlush(a) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
    dbgCGContextFlush(a); \
    fprintf(stderr, "n"); }

inline void dbgCGContextSaveGState(CGContextRef context)
{
    CGContextSaveGState(context);
}

#define CGContextSaveGState(a) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
    dbgCGContextSaveGState(a); \
    fprintf(stderr, "n"); }

inline void dbgCGContextRestoreGState(CGContextRef context)
{
    CGContextRestoreGState(context);
}

#define CGContextRestoreGState(a) {
    fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
    dbgCGContextRestoreGState(a); \
    fprintf(stderr, "n"); }

#define fastarrow_width 16
#define fastarrow_height 16
static const unsigned char fastarrow_bits[] = {
    0x00, 0x00, 0x00, 0x07, 0xe0, 0x07, 0xfc, 0x03, 0xff, 0xff, 0xfc, 0x03,
    0xe0, 0x07, 0x00, 0x07, 0xe0, 0x00, 0xe0, 0x07, 0xc0, 0x3f, 0xff, 0xff,
    0xc0, 0x3f, 0xe0, 0x07, 0xe0, 0x00, 0x00, 0x00};
10.173  fl_arc.cxx File Reference
Utility functions for drawing arcs and circles.
#include <FL/fl_draw.H>
#include <FL/math.h>

10.173.1 Detailed Description
Utility functions for drawing arcs and circles.

10.174  fl_arci.cxx File Reference
Utility functions for drawing circles using integers.
#include <FL/fl_draw.H>
#include <FL/x.H>
#include <config.h>

10.174.1 Detailed Description
Utility functions for drawing circles using integers.

10.175  fl_ask.cxx File Reference
Utility Functions for Common Dialogs.
#include <stdio.h>
#include <stdarg.h>
#include "flstring.h"
#include <FL/Fl.H>
#include <FL/fl_ask.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Secret_Input.H>
#include <FL/x.H>
#include <FL/fl_draw.H>

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 message.

• int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style fmt message, this dialog features up to 3 customizable choice buttons.

• int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Like fl_choice() but with extended (negative) return values.

• const char * fl_input (const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message.
• void fl_message (const char ∗fmt,...)
  Shows an information message dialog box.
• void fl_message_hotspot (int enable)
  Sets whether or not to move the common 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 common 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_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.

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

10.175.1 Detailed Description
Utility Functions for Common Dialogs.

10.176 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 ()
  Sets the drawing function for a given box type.
- void fl_internal_boxtype (Fl_Boxtype t, Fl_Box_Draw_F ∗f)
- 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_THIN_UP_FRAME.
- 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.

10.176.1 Detailed Description

drawing code for common box types.

Generated by Doxygen
10.176.2 Function Documentation

10.176.2.1 fl_internal_boxtype()

void fl_internal_boxtype (  
    Fl_Boxtype t,  
    Fl_Box_Draw_F * f )

Sets the drawing function for a given box type.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>f</td>
<td>box drawing function</td>
</tr>
</tbody>
</table>

10.176.2.2  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.

10.177  fl_cmap.h

```
00001  0x00000000,
00002  0xff000000,
00003  0x00ff0000,
00004  0xffff0000,
00005  0x0000ff00,
00006  0xff00ff00,
00007  0x00ffff00,
00008  0xffffff00,
00009  0x55555500,
00010  0x71717100,
00011  0x71c67100,
00012  0x8e8e8e00,
00013  0x8e388e00,
00014  0x388e8e00,
00015  0x00008000,
00016  0x8e8e8e00,
00017  0x8e8e8e00,
00018  0x8e8e8e00,
00019  0x8e8e8e00,
00020  0x8e8e8e00,
00021  0x8e8e8e00,
00022  0x8e8e8e00,
00023  0x8e8e8e00,
00024  0x8e8e8e00,
00025  0x8e8e8e00,
00026  0x8e8e8e00,
00027  0x8e8e8e00,
00028  0x8e8e8e00,
00029  0x8e8e8e00,
00030  0x8e8e8e00,
00031  0x8e8e8e00,
00032  0x8e8e8e00,
00033  0x8e8e8e00,
00034  0x8e8e8e00,
00035  0x8e8e8e00,
00036  0x8e8e8e00,
00037  0x8e8e8e00,
00038  0x8e8e8e00,
00039  0x8e8e8e00,
00040  0x8e8e8e00,
00041  0x8e8e8e00,
00042  0x8e8e8e00,
00043  0x8e8e8e00,
00044  0x8e8e8e00,
00045  0x8e8e8e00,
00046  0x8e8e8e00,
00047  0x8e8e8e00,
00048  0x8e8e8e00,
00049  0x8e8e8e00,
00050  0x8e8e8e00,
00051  0x8e8e8e00,
00052  0x8e8e8e00,
00053  0x8e8e8e00,
00054  0x8e8e8e00,
00055  0x8e8e8e00,
00056  0x8e8e8e00,
00057  0x8e8e8e00,
00058  0x8e8e8e00,
```
10.178  fl_color.cxx File Reference

Color handling.
#include "Fl_XColor.H"
#include <FL/Fl.H>
#include <FL/x.H>
#include <FL/fl_draw.H>
#include "fl_cmap.h"

Macros

• define fl_overlay 0
  
  HAVE_OVERLAY determines whether fl_overlay is variable or defined as 0.

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_contrast (Fl_Color fg, Fl_Color bg)
  
  Returns a color that contrasts with the background color.

• Fl_Color fl_inactive (Fl_Color c)
  
  Returns the inactive, dimmed version of the given color.

• ulong fl_xpixel (Fl_Color i)
  
  Returns the X pixel number used to draw the given FLTK color index.

• ulong fl_xpixel (uchar r, uchar g, uchar b)
  
  Returns the X pixel number used to draw the given rgb color.

Variables

• uchar fl_bluemask
  
  color mask used in current color map handling

• int fl_blueshift
  
  color shift used in current color map handling

• int fl_extrashift
  
  color shift used in current color map handling

• uchar fl_greenmask
  
  color mask used in current color map handling
10.179 Fl_compose.cxx File Reference

Utility functions to support text input.

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

Variables

- XIC fl_xim_ic

10.179.1 Detailed Description

Utility functions to support text input.

10.180 fl_curve.cxx File Reference

Utility for drawing Bezier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.

```c
#include <FL/fl_draw.H>
#include <math.h>
```

10.180.1 Detailed Description

Utility for drawing Bezier 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!

10.181 fl_dnd_x.cxx

```c
// Drag & Drop code 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:
// https://www.fltk.org/bugs.php

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

Generated by Doxygen
#include <FL/x.H>
#include "flstring.h"

extern Atom fl_XdndAware;
extern Atom fl_XdndSelection;
extern Atom fl_XdndEnter;
extern Atom fl_XdndTypeList;
extern Atom fl_XdndPosition;
extern Atom fl_XdndLeave;
extern Atom fl_XdndDrop;
extern Atom fl_XdndStatus;
extern Atom fl_XdndActionCopy;
extern Atom fl_XdndFinished;
extern Atom fl_XdndURIList;
extern Atom fl_XaUtf8String;

extern char fl_i_own_selection[2];
extern char *fl_selection_buffer[2];

extern void fl_sendClientMessage(Window window, Atom message,
    unsigned long d0,
    unsigned long d1=0,
    unsigned long d2=0,
    unsigned long d3=0,
    unsigned long d4=0);

// return version # of Xdnd this window supports. Also change the
// window to the proxy if it uses a proxy:
static int dnd_aware(Window& window) {
    Atom actual; int format; unsigned long count, remaining;
    unsigned char *data = 0;
    XGetWindowProperty(fl_display, window, fl_XdndAware,
        0, 4, False, XA_ATOM,
        &actual, &format, &count, &remaining, &data);
    int ret = 0;
    if (actual == XA_ATOM && format==32 && count && data)
        ret = int( *(Atom*)data);
    if (data) { XFree(data); data = 0; }
    return ret;
}

static int grabfunc(int event) {
    if (event == FL_RELEASE) Fl::pushed(0);
    return 0;
}

// send an event to an fltk window belonging to this program:
static int local_handle(int event, Fl_Window* window) {
    fl_local_grab = grabfunc;
    return Fl::handle(event,window);
}

int Fl::dnd() {
    Fl_Window *source_fl_win = Fl::first_window();
    Fl::first_window()->cursor(FL_CURSOR_MOVE);
    Window source_window = fl_xid(Fl::first_window());
    fl_local_grab = grabfunc;
    Window target_window = 0;
    Fl_Window *local_window = 0;
    int dndversion = 4; int dest_x, dest_y;
    XSetSelectionOwner(fl_display, fl_XdndSelection, fl_message_window, fl_event_time);

    while (Fl::pushed()) {
        // figure out what window we are pointing at:
        Window new_window = 0; int new_version = 0;
        Fl_Window* new_local_window = 0;
        for (Window child = RootWindow(fl_display, fl_screen));) {
            Window root; unsigned int junk3;
            XQueryPointer(fl_display, child, &root, &child, &e_x_root, &e_y_root, &dest_x, &dest_y, &junk3);
            if (child) {
                if ((new_window && (new_version = dnd_aware(root)))
                    new_window = root;
                break;
            }
            new_window = child;
            if ((new_local_window = fl_find(child))) break;
            if ((new_version = dnd_aware(new_window))) break;
        }
    }
}
if (new_window != target_window) {
  if (local_window) {
    local_handle(FL_DND_LEAVE, local_window);
  } else if (dndversion) {
    fl_sendClientMessage(target_window, fl_XdndLeave, source_window);
  }
  dndversion = new_version;
  target_window = new_window;
  local_window = new_local_window;
  if (local_window) {
    local_handle(FL_DND_ENTER, local_window);
  } else if (dndversion) {
    // Send an X-DND message to the target window. In order to
    // support dragging of files/URLs as well as arbitrary text,
    // we look at the selection buffer - if the buffer starts
    // with a common URI scheme, does not contain spaces, and
    // contains at least one CR LF, then we flag the data as
    // both a URI list (MIME media type "text/uri-list") and
    // plain text. Otherwise, we just say it is plain text.
    if (!strncmp(fl_selection_buffer[0], "file://", 8) ||
        !strncmp(fl_selection_buffer[0], "http://", 7) ||
        !strncmp(fl_selection_buffer[0], "https://", 8) ||
        !strncmp(fl_selection_buffer[0], "ipp://", 6) ||
        !strncmp(fl_selection_buffer[0], "mailto:", 7) ||
        !strncmp(fl_selection_buffer[0], "news://", 6) ||
        !strncmp(fl_selection_buffer[0], "smb://", 6)) &&
        !strchr(fl_selection_buffer[0], ' ') &&
        strstr(fl_selection_buffer[0], "\r\n") {
      // Send file/URI list...
      fl_sendClientMessage(target_window, fl_XdndEnter, source_window, dndversion<<24,
                           fl_XdndURIList, fl_XaUtf8String, XA_STRING);
    } else {
      // Send plain text...
      fl_sendClientMessage(target_window, fl_XdndEnter, source_window, dndversion<<24,
                           fl_XaUtf8String, XA_STRING, 0);
    }
  }
  Fl::wait();
}

if (local_window) {
  fl_i_own_selection[0] = 1;
  if (local_handle(FL_DND_DRAG, local_window)) paste(*belowmouse(), 0);
} else if (dndversion) {
  fl_sendClientMessage(target_window, fl_XdndDrag, source_window,
                       0, (e_x_root<<16)|e_y_root, fl_event_time,
                       fl_XdndActionCopy);
}

if (local_window) {
  if (!strncmp(fl_selection_buffer[0], "file://", 8) ||
      !strncmp(fl_selection_buffer[0], "ftp://", 6) ||
      !strncmp(fl_selection_buffer[0], "http://", 7) ||
      !strncmp(fl_selection_buffer[0], "https://", 8) ||
      !strncmp(fl_selection_buffer[0], "ipp://", 6) ||
      !strncmp(fl_selection_buffer[0], "ldap:", 5) ||
      !strncmp(fl_selection_buffer[0], "mailto:", 7) ||
      !strncmp(fl_selection_buffer[0], "news://", 6)) &&
      !strchr(fl_selection_buffer[0], ' ') &&
      strstr(fl_selection_buffer[0], "\r\n") {
    // fake a drop by clicking the middle mouse button:
    XButtonEvent msg;
    msg.type = ButtonPress;
    msg.window = target_window;
    msg.root = RootWindow(fl_display, fl_screen);
    msg.x = dest_x;
    msg.y = dest_y;
    msg.x_root = Fl::e_x_root;
    msg.y_root = Fl::e_y_root;
    msg.state = 0x0;
    msg.button = Button2;
    XSendEvent(fl_display, target_window, False, 0L, (XEvent *)&msg);
    msg.time++;
    msg.state = 0x200;
    msg.type = ButtonRelease;
    XSendEvent(fl_display, target_window, False, 0L, (XEvent *)&msg);
  }
  if (local_grab) {
    if (local_window) {
      if (local_handle(FL_DND_RELEASE, local_window)) paste(*belowmouse(), 0);
    } else if (dndversion) {
      fl_sendClientMessage(target_window, fl_XdndDrop, source_window,
                            0, fl_event_time);
    } else if (target_window) {
      // fake a drop by clicking the middle mouse button:
      XButtonEvent msg;
      msg.type = ButtonPress;
      msg.window = target_window;
      msg.root = RootWindow(fl_display, fl_screen);
      msg.x = dest_x;
      msg.y = dest_y;
      msg.x_root = Fl::e_x_root;
      msg.y_root = Fl::e_y_root;
      msg.state = 0x0;
      msg.button = Button2;
      XSendEvent(fl_display, target_window, False, 0L, (XEvent *)&msg);
      msg.time++;
      msg.state = 0x200;
      msg.type = ButtonRelease;
      XSendEvent(fl_display, target_window, False, 0L, (XEvent *)&msg);
      return 1;
    }
  }
  if (local_grab) { return 1; }

Generated by Doxygen
10.182 Fl_Double_Window.cxx File Reference

Fl_Double_Window implementation.

```c++
#include <config.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Overlay_Window.H>
#include <FL/Fl_Printer.H>
#include <FL/x.H>
#include <FL/fl_draw.H>
```

### Functions

- **void fl_begin_offscreen (Fl_Offscreen ctx)**
  Send all subsequent drawing commands to this offscreen buffer.

- **char fl_can_do_alpha_blending ()**
  Checks whether platform supports true alpha blending for RGBA images.

- **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_delete_offscreen (Fl_Offscreen ctx)**
  Deletion of an offscreen graphics buffer.

- **void fl_end_offscreen ()**
  Quit sending drawing commands to the current offscreen buffer.

### Variables

- **const int stack_max = 16**

10.182.1 Detailed Description

Fl_Double_Window implementation.

10.183 Fl_Font.H

```c++
// "SIDS"
// Font definitions for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2011 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
// Two internal fltk data structures:
// Fl_Fontdesc: an entry into the fl_font() table. There is one of these
// for each fltk font number.
```

```c++
 ifndef FL_FONT_
 define FL_FONT_
 #include <config.h>
 #include <config.h>
 if USE_XFT
```
typedef struct _XftFont XftFont;

# if defined(WIN32) && !defined(__APPLE__)
XftFont f;
# endif

int width[64];

TEXTMETRIC metr;

int angle;

XftFont * font;

int angle;

XftFont *xfontname, Fl_Fontsize size, int angle);

# else

CTFontRef fontref;

float width[512]; // array of arrays of character widths

# endif

# elif USE_XFT

XftFont * font;

//const char * encoding;

int angle;

FL_EXPORT Fl_Font_Descriptor(const char * xfontname, Fl_Fontsize size, int angle);

# else

XUtf8FontStruct * font; // X UTF-8 font information

int angle;

FL_EXPORT Fl_Font_Descriptor(const char * xfontname);

# endif

# if HAVE_GL

unsigned int listbase;// base of display list, 0 = none

#ifndef __APPLE_QUARTZ__

char glok[64];

#endif // __APPLE_QUARTZ__

# endif // HAVE_GL

// the unicode span is divided in 512 blocks of 128 characters

# endif // FL_DOXYGEN

};

FL_EXPORT ~Fl_Font_Descriptor();

#endif // FL_DOXYGEN

外接 FL_EXPORT Fl_Font_Descriptor *fl_fontsize; // the currently selected one

struct Fl_Fontdesc {

const char *name;

descriptor *name;

char fontname[128]; // "Pretty" font name

char xfontname[128]; // matched X font names

int n; // size of xlist, negative = don't free xlist!

# endif

};

FL_EXPORT Fl_Fontdesc *fl_fonts; // the table

# ifndef WIN32

// functions for parsing X font names:

FL_EXPORT const char *fl_font_word(const char *p, int n);

FL_EXPORT char *fl_find_fontsize(char *name);

# endif

# endif

// End of "$Id$".
#ifndef FL_DOXYGEN

Fl_Font_Descriptor::Fl_Font_Descriptor(const char* name) {
  font = XCreateUtf8FontStruct(fl_display, name);
  if (!font) {
    Fl::warning("bad font: ": name);
    font = XCreateUtf8FontStruct(fl_display, "fixed");
  }
}

Fl_XFont_On_Demand fl_xfont;

Fl_Font_Descriptor::~Fl_Font_Descriptor() {
  if (this == fl_graphics_driver->font_descriptor()) {
    fl_graphics_driver->font_descriptor(NULL);
    fl_xfont = 0;
  }
  XFreeUtf8FontStruct(fl_display, font);
}

// WARNING: if you add to this table, you must redefine FL_FREE_FONT
// in Enumerations.H & recompile!!
static Fl_Fontdesc built_in_table[] = {
  "-*-helvetica-medium-r-normal-**",
  "-*-helvetica-bold-r-normal-**",
  "-*-helvetica-medium-o-normal-**",
  "-*-helvetica-bold-o-normal-**",
  "-*-courier-medium-r-normal-**",
  "-*-courier-bold-r-normal-**",
  "*-times-medium-r-normal-**",
  "*-times-bold-r-normal-**",
  "*-times-medium-i-normal-**",
  "*-times-bold-i-normal-**",
  "*-symbol-**",
  "*-lucidatypewriter-medium-r-normal-sans-**",
  "*-lucidatypewriter-bold-r-normal-sans-**",
  "*-zapf dingbats-**",
};

Fl_Fontdesc* fl_fonts = built_in_table;

#define MAXSIZE 32767

const char* fl_font_word(const char* p, int n) {
  while ( *p) {if (*p=='-') {if (!--n) break;} p++}
  return p;
}

char* fl_find_fontsize(char* name) {
  char * c = name;
  if ( *c== '-') {
    c = (char *)fl_font_word(c,7);
    if ( *c++ && isdigit(*c)) return c;
  }
  return 0;
}

while (+p) {if (p=='-') {if (!--n) break} p++;
  return p;
}

const char* fl_find_fontsize(char* name) {
  char c = name;
  // for standard x font names, try after 7th dash:
  if (*c == '-') {
    c = (char*)fl_font_word(c,7);
    if (*c++ && isdigit(*c)) return c;
  }
  return 0;
  // malformed x font name?
}

char r = 0;

// find last set of digits:
for (c++; c++)
  if (isdigit(*c) && !isdigit(*(c-1))) r = c;

Generated by Doxygen
return r;
}

//const char* fl_encoding = "iso8859-1";
const char* fl_encoding = "iso10646-1";

// return true if this matches fl_encoding:
int fl_correct_encoding(const char* name) {
    if (*name != '-') return 0;
    const char * c = fl_font_word(name, 13);
    return (*c++ && !strcmp(c, fl_encoding));
}

static const char *find_best_font(const char *fname, int size) {
    int cnt;
    static char **list = NULL;
    // locate or create an Fl_Font_Descriptor for a given Fl_Fontdesc and size:
    if (list) XFreeFontNames(list);
    list = XListFonts(fl_display, fname, 100, &cnt);
    if (!list) return "fixed";

    // search for largest <= font size:
    char name = list[0]; int ptsize = 0; // best one found so far
    int matchedlength = 32767; // holds scalable font name
    int found_encoding = 0;
    for (int n=0; n < cnt; n++) {
        char * thisname = list[n];
        if (fl_correct_encoding(thisname)) {
            if (!found_encoding) ptsize = 0; // force it to choose this
            found_encoding = 1;
        } else {
            if (found_encoding) continue;
        }
        char * c = (char*)fl_find_fontsize(thisname);
        int thissize = c ? atoi(c) : MAXSIZE;
        int thislength = strlen(thisname);
        if (thissize == size) {
            name = thisname;
            ptsize = size;
            matchedlength = thislength;
        } else if (thissize > size) {
            if (found_encoding) continue;
        } else if (thissize < size) {
            name = thisname;
            ptsize = thissize;
            matchedlength = thislength;
        }
    }

    // if (ptsize != size) { // see if we already found this unscalable font:
    // for (f = s->first; f; f = f->next) {
    // if (f->minsize <= ptsize && f->maxsize >= ptsize) {
    // f->minsize = ptsize; f->maxsize = ptsize;
    // return f;
    // } else if (ptsize == size) {
    // f->minsize = ptsize; f->maxsize = ptsize;
    // return f;
    // } else if (ptsize < size) {
    // f->minsize = size; f->maxsize = size;
    // return f;
    // } else if (ptsize > size) {
    // f->minsize = size; f->maxsize = size;
    // return f;
    // } else {
    // f->minsize = size; f->maxsize = size;
    // return f;
    // } else {
    // okay, we definately have some name, make the font:
    // s = new Fl_Font_Descriptor(name);
    // if (ptsize < size) {f->minsize = ptsize; f->maxsize = size} else {f->minsize = size; f->maxsize = ptsize} ;
    // } else {
    // okay, we definately have some name, make the font:
    // s = new Fl_Font_Descriptor(name);
    // if (ptsize < size) {f->minsize = ptsize; f->maxsize = size} else {f->minsize = size; f->maxsize = ptsize} ;
    // return name;
    // }

    static const char *find_best_font(const char *fname, int size) {
        int i = 0;
        char *buf;
        return r;
    }

    static char *put_font_size(const char *n, int size)
    {
        return r;
    }
const char *ptr;
const char *f;
char *name;
int nbf = 1;
name = strdup(n);
while (name[i]) {
    if (name[i] == ',') {nbf++; name[i] = '\0';}
i++;
}
buf = (char *) malloc(nbf * 256);
buf[0] = '\0';
ptr = name;
i = 0;
while (ptr && nbf > 0) {
    f = find_best_font(ptr, size);
    while (*f) {
        buf[i] = *f;
        f++; i++;
    }
    nbf--;
    while (*ptr) ptr++;
    if (nbf) {
        ptr++;
        buf[i] = ',';
        i++;
        while(isspace(*ptr)) ptr++;
    }
    buf[i] = '\0';
    free(name);
    return buf;
}
}char *fl_get_font_xfld(int fnum, int size) {
    Fl_Fontdesc * s = fl_fonts+fnum;
    if (!s->name) s = fl_fonts; // use font 0 if still undefined
    fl_open_display();
    return put_font_size(s->name, size);
}
static Fl_Font_Descriptor* find(int fnum, int size) {
    char *name;
    Fl_Fontdesc * s = fl_fonts+fnum;
    if (!s->name) s = fl_fonts; // use font 0 if still undefined
    Fl_Font_Descriptor * f;
    for (f = s->first; f; f = f->next)
        if (f->size == size) return f;
    fl_open_display();
    name = put_font_size(s->name, size);
    f = new Fl_Font_Descriptor(name);
    f->size = size;
    f->next = s->first;
    s->first = f;
    free(name);
    return f;
}void Fl_Xlib_Graphics_Driver::font(Fl_Font fnum, Fl_Fontsize size) {
    if (fnum==-1) {
        Fl_Graphics_Driver::font(0, 0);
        return;
    }
    if (fnum == Fl_Graphics_Driver::font() && size == Fl_Graphics_Driver::size()) return;
    Fl_Graphics_Driver::font(fnum, size);
    Fl_Font_Descriptor * f = find(fnum, size);
    if (f != this->font_descriptor()) {
        this->font_descriptor(f);
        fl_xfont = f->fonts[0];
        font_gc = 0;
    }
    return;
}int Fl_Xlib_Graphics_Driver::height() { GENERATED BY DOXYGEN
}
if (font_descriptor()) return font_descriptor()->font->ascent + font_descriptor()->font->descent;
else return -1;
}

int Fl_Xlib_Graphics_Driver::descent() {
if (font_descriptor()) return font_descriptor()->font->descent;
else return -1;
}

int Fl_Xlib_Graphics_Driver::width(const char* c, int n) {
if (font_descriptor()) return (double) XUtf8TextWidth(font_descriptor()->font, c, n);
else return -1;
}

double Fl_Xlib_Graphics_Driver::width(unsigned int c) {
if (font_descriptor()) return (double) XUtf8UcsWidth(font_descriptor()->font, c);
else return -1;
}

void Fl_Xlib_Graphics_Driver::text_extents(const char *c, int n, int &dx, int &dy, int &W, int &H) {
if (font_gc != fl_gc) {
if (!font_descriptor()) font(FL_HELVETICA, FL_NORMAL_SIZE);
font_gc = fl_gc;
XSetFont(fl_display, fl_gc, font_descriptor()->font->fid);
}
int xx, yy, ww, hh;
xx = yy = ww = hh = 0;
if (fl_gc) XUtf8_measure_extents(fl_display, fl_window, font_descriptor()->font, fl_gc, &xx, &yy,
4ww, 4hh, c, n);
W = ww; H = hh; dx = xx; dy = yy;
}

void Fl_Xlib_Graphics_Driver::draw(const char* c, int n, int x, int y) {
if (font_gc != fl_gc) {
if (!font_descriptor()) this->font(FL_HELVETICA, FL_NORMAL_SIZE);
font_gc = fl_gc;
XSetFont(fl_display, fl_gc, font_descriptor()->font->fid);
}
if (fl_gc) XUtf8DrawString(fl_display, fl_window, font_descriptor()->font, fl_gc, x, y, c, n);
}

void Fl_Xlib_Graphics_Driver::rtl_draw(const char* c, int n, int x, int y) {
if (font_gc != fl_gc) {
if (!font_descriptor()) this->font(FL_HELVETICA, FL_NORMAL_SIZE);
font_gc = fl_gc;
XSetFont(fl_display, fl_gc, font_descriptor()->font->fid);
}
if (fl_gc) XUtf8DrawRtlString(fl_display, fl_window, font_descriptor()->font, fl_gc, x, y, c, n);
}

#endif // FL_DOXYGEN

// End of "$Id$".
// Generated by Doxygen

// OpenGL definitions 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:
// http://www.fltk.org/COPYING.php
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).

The function 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 fl_create_gl_context is provided to create it for any
X visual.

fl_set_gl_context makes the given OpenGL context current and makes
it draw into the passed window. It tracks the current one context
and avoids calling the context switching code when the same context
is used, though it is a mystery to me why the GLX/WGL libraries
don't do this themselves...

fl_no_gl_context clears that cache so the next fl_set_gl_context is
guaranteed to work.

fl_delete_gl_context destroys the context.

This code is used by Fl_Gl_Window, gl_start(), and gl_visual()

#ifndef Fl_Gl_Choice_H
#define Fl_Gl_Choice_H

// Warning: whatever GLContext is defined to must take exactly the same
// space in a structure as a void*!!!

#ifdef WIN32
#include <FL/gl.h>
define GLContext HGLRC
#elif defined(__APPLE_QUARTZ__)
#include <OpenGL/gl.h>
define GLContext NSOpenGLContext
#else
class NSOpenGLPixelFormat;
class NSOpenGLContext;
define GLContext NSOpenGLContext*
define NSOpenGLPixelFormat*
define NSOpenGLContext
#endif

class Fl_Gl_Choice {
int mode;
const int *alist;
Fl_Gl_Choice *next;

public:

#ifndef WIN32
#endif

int pixelformat; // the visual to use
PIXELFORMATDESCRIPTION pf;
#endif define(__APPLE_QUARTZ__)

NSOpenGLPixelFormat* pixelformat;

#define GLXCONTEXT

XVisualInfo *vis; // the visual to use
Colormap colormap; // a colormap for that visual
GLXFBConfig best_fb;
#endif

// Return one of these structures for a given gl mode.
// The second argument is a glX attribute list, and is used if mode is
// zero. This is not supported on Win32;
static Fl_Gl_Choice *find(int mode, const int *);

#endif define(__APPLE_QUARTZ__)

#ifdef WIN32
GLContext fl_create_gl_context(Fl_Window*, const Fl_Gl_Choice*, int layer=0);
#else

#endif

Generated by Doxygen
GLContext fl_create_gl_context(Fl_Window*, const Fl_Gl_Choice*, int layer=0);
#else
GLContext fl_create_gl_context(XVisualInfo* vis);
#endif

//static inline
GLContext fl_create_gl_context(Fl_Window*, const Fl_Gl_Choice* g);//
return fl_create_glcontext(g->vis);
#endif

void fl_set_gl_context(Fl_Window*, GLContext);
void fl_no_gl_context();
void fl_delete_gl_context(GLContext);
#endif

// End of "$Id$".

#include <FL/Fl.H>
#include <FL/fl_draw.H>
#include <FL/x.H>
#include <FL/Fl_Printer.H>
#include "flstring.h"
#include <stdio.h>

Variables
- int fl_line_width = 0

10.186.1 Detailed Description
Line style drawing utility hiding different platforms.

10.187 FL_Native_File_Chooser_common.cxx

#include <string.h>
#include <FL/Enumerations.H>

static char *strnew(const char *val) {
  if ( val == NULL ) return(NULL);
  char *s = new char[strlen(val)+1];
  strcpy(s, val);
  return(s);
}
static char *strfree(char *val) {
    if ( val ) delete [] val;
    return(NULL);
}

static char *strapp(char *s, const char *val) {
    if ( ! val ) {
        return(s); // Nothing to append? return s
    }
    if ( ! s ) {
        return(strnew(val)); // New string? return copy of val
    }
    if ( ! s ) {
        return(strnew(val)); // New string? return copy of val
    }
    char *news = new char[strlen(s)+strlen(val)+1];
    strcpy(news, s);
    strcat(news, val);
    delete [] s; // delete old string
    return(news); // return new copy
}

void chrcat(char *s, char c) {
    char tmp[2] = { c, '\0' };
    strcat(s, tmp);
}

int Fl_Native_File_Chooser::have_looked_for_GTK_libs = 0;

10.188 Fl_Native_File_Chooser_FLTK.cxx

#include <config.h>
#include <FL/Fl_Native_File_Chooser.H>
#include <FL/Fl_Native_File_Chooser_common.cxx>
#include <FL/Fl_Native_File_Chooser_GTK.cxx>
#include <sys/stat.h>
#include <string.h>

int Fl_Native_File_Chooser::have_looked_for_GTK_libs = 0;

// FREE STRING CREATED WITH strnew(), NULLS OUT STRING
// Value can be NULL

// 'DYNAMICALLY' APPEND ONE STRING TO ANOTHER
// Returns newly allocated string, or NULL
// 's' can be NULL; returns a strnew(val).
// 'val' can be NULL; s is returned unmodified.
// Usage:
//  char *s = strnew("foo"); // s = "foo"
//  s = strapp(s, "bar"); // s = "foobar"

// APPEND A CHARACTER TO A STRING
// This does NOT allocate space for the new character.

// End of "$Id$".

// FLTK native file chooser widget wrapper for GTK's GtkFileChooserDialog
// Copyright 1998-2014 by Bill Spitzak and others.
// Copyright 2012 IMM
// 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 to:
// http://www.fltk.org/str.php

// Generated by Doxygen
```cpp
#include <Fl_Native_File_Chooser.h>

Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
#if FLTK_ABI_VERSION <= 10302
    _btype = val;
    _options = NO_OPTIONS;
    _filter = NULL;
    _filtvalue = 0;
    _parsedfilter = NULL;
    _preset_file = NULL;
    _prevvalue = NULL;
    _directory = NULL;
    _errmsg = NULL;
#endif // FLTK_ABI_VERSION
    if (have_looked_for_GTK_libs == 0) {
        // First time here, try to find the GTK libs if they are installed
        if (HAVE_DLSYM && HAVE_DLFCN_H)
            if (Fl::option(Fl::OPTION_FNFC_USES_GTK)) {
                Fl_GTK_File_Chooser::probe_for_GTK_libs();
            }
        have_looked_for_GTK_libs = -1;
    }
    // if we found all the GTK functions we need, we will use the GtkFileChooserDialog
    if ((Fl_GTK_File_Chooser::did_find_GTK_libs)
        _gtk_file_chooser = new Fl_GTK_File_Chooser(val);
    else
        _x11_file_chooser = new Fl_FLTK_File_Chooser(val);
}
Fl_Native_File_Chooser::~Fl_Native_File_Chooser() {
    delete _x11_file_chooser;
}

void Fl_Native_File_Chooser::type(int t) { return _x11_file_chooser->type(t); }
int Fl_Native_File_Chooser::type() const { return _x11_file_chooser->type(); }
void Fl_Native_File_Chooser::options(int o) { _x11_file_chooser->options(o); }
int Fl_Native_File_Chooser::options() const { return _x11_file_chooser->options(); }
int Fl_Native_File_Chooser::count() const { return _x11_file_chooser->count(); }
const char *Fl_Native_File_Chooser::filename() const { return _x11_file_chooser->filename(); }
const char *Fl_Native_File_Chooser::filename(int i) const { return _x11_file_chooser->filename(i); }
void Fl_Native_File_Chooser::directory(const char *val) { _x11_file_chooser->directory(val); }
const char *Fl_Native_File_Chooser::directory() const { return _x11_file_chooser->directory(); }
void Fl_Native_File_Chooser::title(const char *t) { _x11_file_chooser->title(t); }
const char* Fl_Native_File_Chooser::title() const { return _x11_file_chooser->title(); }
void Fl_Native_File_Chooser::filter(const char *f) { _x11_file_chooser->filter(f); }
int Fl_Native_File_Chooser::filters() const { return _x11_file_chooser->filters(); }
void Fl_Native_File_Chooser::filter_value(int i) { _x11_file_chooser->filter_value(i); }
int Fl_Native_File_Chooser::filter_value() const { return _x11_file_chooser->filter_value(); }
void Fl_Native_File_Chooser::preset_file(const char* f) { _x11_file_chooser->preset_file(f); }
const char* Fl_Native_File_Chooser::preset_file() const { return _x11_file_chooser->preset_file(); }
int Fl_Native_File_Chooser::show() { return _x11_file_chooser->show(); }
}
```
```cpp
Fl_FLTK_File_Chooser::~Fl_FLTK_File_Chooser() {
    delete _file_chooser;
    _file_chooser = NULL;
    _filter = strfree(_filter);
    _parsedfilt = strfree(_parsedfilt);
    _preset_file = strfree(_preset_file);
    _prevvalue = strfree(_prevvalue);
    _directory = strfree(_directory);
    _errmsg = strfree(_errmsg);
}

void Fl_FLTK_File_Chooser::errmsg(const char *msg) {
    _errmsg = strfree(_errmsg);
    _errmsg = strnew(msg);
}

int Fl_FLTK_File_Chooser::type_fl_file(int val) {
    switch (val) {
    case Fl_Native_File_Chooser::BROWSE_FILE:
        return(Fl_File_Chooser::SINGLE);
    case Fl_Native_File_Chooser::BROWSE_DIRECTORY:
        return(Fl_File_Chooser::SINGLE | Fl_File_Chooser::DIRECTORY);
    case Fl_Native_File_Chooser::BROWSE_MULTI_FILE:
        return(Fl_File_Chooser::MULTI);
    case Fl_Native_File_Chooser::BROWSE_MULTI_DIRECTORY:
        return(Fl_File_Chooser::DIRECTOY | Fl_File_Chooser::MULTI);
    case Fl_Native_File_Chooser::BROWSE_SAVE_FILE:
        return(Fl_File_Chooser::SINGLE | Fl_File_Chooser::CREATE);
    case Fl_Native_File_Chooser::BROWSE_SAVE_DIRECTORY:
        return(Fl_File_Chooser::DIRECTOY | Fl_File_Chooser::MULTI | Fl_File_Chooser::CREATE);
    default:
        return(Fl_File_Chooser::SINGLE);
    }
}

void Fl_FLTK_File_Chooser::type(int val) {
    _btype = val;
    _file_chooser->type(type_fl_file(val));
}

int Fl_FLTK_File_Chooser::type() const {
    return(_btype);
}

void Fl_FLTK_File_Chooser::options(int val) {
    _options = val;
}

int Fl_FLTK_File_Chooser::options() const {
    return(_options);
}

int Fl_FLTK_File_Chooser::show() {
    // FILTER
    if ( _parsedfilt ) {
        _file_chooser->filter(_parsedfilt);
    }
    // FILTER VALUE
    // Set this /after/ setting the filter
    // _file_chooser->filter_value(_filvalue);
    // DIRECTORY
    if ( _directory && _directory[0] ) {
        _file_chooser->directory(_directory);
    } else {
        _file_chooser->directory(_prevvalue);
    }
    // PRESET FILE
    if ( _preset_file ) {
        _file_chooser->value(_preset_file);
    }
    // OPTIONS: PREVIEW
    _file_chooser->preview( (options() & Fl_Native_File_Chooser::PREVIEW) ? 1 : 0);
    // OPTIONS: NEW FOLDER
    if ( options() & Fl_Native_File_Chooser::NEW_FOLDER )
        _file_chooser->type(_file_chooser->type() | Fl_File_Chooser::CREATE); // on
```

Generated by Doxygen
// SHOW
_file_chooser->show();

// BLOCK WHILE BROWSER SHOWN
while ( _file_chooser->shown() ) {
 Fl::wait();
}

if ( _file_chooser->value() && _file_chooser->value()[0] ) {
 _prevvalue = strfree(_prevvalue);
 _prevvalue = strnew(_file_chooser->value());
 _filtvalue = _file_chooser->filter_value(); // update filter value
}

// HANDLE SHOWING 'SaveAs' CONFIRM
if ( options() & Fl_Native_File_Chooser::SAVEAS_CONFIRM && type() == Fl_Native_File_Chooser::BROWSE_SAVE_FILE ) {
 struct stat buf;
 if ( stat(_file_chooser->value(), &buf) != -1 ) {
 if ( buf.st_mode & S_IFREG ) { // Regular file + exists?
 if ( exist_dialog() == 0 ) {
 return(1);
 }
 }
 }
}

if ( _file_chooser->count() ) return(0);
else return(1);

const char *Fl_FLTK_File_Chooser::errmsg() const {
 return(_errmsg ? _errmsg : "No error");
}

const char* Fl_FLTK_File_Chooser::filename() const {
 if ( _file_chooser->count() > 0 ) {
 return(_file_chooser->value());
 }
 return("");
}

const char* Fl_FLTK_File_Chooser::filename(int i) const {
 if ( i < _file_chooser->count() ) {
 return(_file_chooser->value(i+1)); // convert fltk 1 based to our 0 based
 }
 return("");
}

void Fl_FLTK_File_Chooser::title(const char *val)
 _file_chooser->label(val);

void Fl_FLTK_File_Chooser::directory(const char *val)
 _directory = strfree(_directory);
 _directory = strnew(val);
}

const char *Fl_FLTK_File_Chooser::title() const {
 return(_file_chooser->label());
}

const char *Fl_FLTK_File_Chooser::filename(int i) const {
 if ( i < _file_chooser->count() ) {
 return(_file_chooser->value(i+1)); // convert fltk 1 based to our 0 based
 }
 return("");
}

void Fl_FLTK_File_Chooser::filter(const char *val)
 _filter = strfree(_filter);
 _filter = strnew(val);
 parse_filter();
}

int Fl_FLTK_File_Chooser::filters() const {
 return(_nfilters);
}

void Fl_FLTK_File_Chooser::filter_value(int val)
 _filtvalue = val;

int Fl_FLTK_File_Chooser::count() const {
 return _file_chooser->count();
}

int Fl_FLTK_File_Chooser::filenames(int i) const {
 return(_file_chooser->count());
}

int Fl_FLTK_File_Chooser::filtervalues(int i) const {
 return(_filtvalue);
const char *Fl_FLTK_File_Chooser::directory() const {
    return _directory;
}

// PRIVATE: Convert our filter format to fltk's chooser format
// FROM TO (FLTK)
// ------------------------- --------------------------
// "*.cxx" " *.cxx Files(*.cxx)"
// "C Files*t.(cxx,h)" "C Files(*.cxx,h)"
// "C Files	.*(cxx,h)\nText Files\.txt" "C Files*.cxx,h)\nText Files\.txt"

Returns a modified version of the filter that the caller is responsible
for freeing with strfree().

void Fl_FLTK_File_Chooser::parse_filter() {
    _parsedfilt = strfree(_parsedfilt); // clear previous parsed filter (if any)
    _nfilters = 0;
    char *in = _filter;
    if ( !in ) return;

    int has_name = strchr(in, '\t') ? 1 : 0;
    char mode = has_name ? 'n' : 'w'; // parse mode: n=title, w=wildcard
    char wildcard[1024] = ""; // parsed wildcard
    char name[1024] = "";

    // Parse filter user specified
    for ( ; in; in++ ) {
        switch ( *in) {
            case '\t':
                if ( mode != 'n' ) goto regchar;
                mode = 'w';
                break;
            case '\n':
                goto regex;
            case '\0':
                // FINISHED PARSING NAME?
                case '\0';
                    // APPEND NEW FILTER TO LIST
                    if ( wildcard[0] ) {
                        // OUT: "name(wild)\nname(wild)"
                        char comp[2048];
                        sprintf(comp, "%s%.511s(%.511s)", ((_parsedfilt)?"":"%), name, wildcard);
                        _parsedfilt = strapp(_parsedfilt, comp);
                        _nfilters++;
                    } //DEBUG printf("DEBUG: PARSED FILT NOW <%s>
                    //RESET wildcard[0] = name[0] = '\0';
            case '\0':
                // APPEND NEW FILTER TO LIST
            default: // handle all non-special chars
                // handle regular char
        }
    }

    // Parse all other chars
    switch ( mode ) {
        case 'n': chrcat(name, *in); continue;
        case 'w': chrcat(wildcard, *in); continue;
    }
    break;
}

int Fl_FLTK_File_Chooser::exist_dialog() {

}

const char *Fl_FLTK_File_Chooser::preset_file() const {
    return _preset_file;
}

void Fl_FLTK_File_Chooser::preset_file(const char* val) {
    _preset_file = strfree(_preset_file);
    _preset_file = strnew(val);
}

const char *Fl_FLTK_File_Chooser::preset_file() const {
    return _preset_file;
}

int Fl_FLTK_File_Chooser::exist_dialog() {
return fl_choice("%s", fl_cancel, fl_ok, NULL, Fl_Native_File_Chooser::file_exists_message);
}

// End of "$Id$".

// "$Id$"

// FLTK native file chooser widget wrapper for GTK's GtkFileChooserDialog

// Copyright 1998-2014 by Bill Spitzak and others.

// Copyright 2012 IMM

// 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 to:

// http://www.fltk.org/str.php

#include <FL/x.H>

#if HAVE_DLSYM && HAVE_DLFCN_H
#include <dlfcn.h> // for dlopen et al
#endif

#include <locale.h> // for setlocale

/* --------------------- Type definitions from GLIB and GTK --------------------- */

#ifndef FALSE
#define FALSE (0)
#endif

#ifndef TRUE
#define TRUE (!FALSE)
#endif

typedef void* gpointer;

typedef int gint;

typedef unsigned int guint;

typedef unsigned long gulong;

typedef gint gboolean;

typedef char gchar;

typedef struct _GSList GSList;

struct _GSList
{
    gpointer data;
    GSList *next;
};

define g_slist_next(slist) ((slist) ? (((GSList *)(slist))->next) : NULL)

typedef gpointer *gpointer;

typedef int gint;

typedef unsigned int guint;

typedef unsigned long gulong;

typedef gboolean gboolean;

typedef struct _GtkWidget GtkWidget;

typedef struct _GtkFileChooser GtkFileChooser;

typedef struct _GtkDialog GtkDialog;

typedef struct _GtkFileFilter GtkFileFilter;

typedef struct _GtkToggleButton GtkToggleButton;

typedef enum {
    GTK_FILE_FILTER_FILENAME = 1 << 0,
    GTK_FILE_FILTER_URI = 1 << 1,
    GTK_FILE_FILTER_DISPLAY_NAME = 1 << 2,
    GTK_FILE_FILTER_MIME_TYPE = 1 << 3
} GtkFileFilterFlags;

typedef struct _GtkFileFilterInfo GtkFileFilterInfo;

define _GtkFileChooserGtkFileChooserDialog;

define _GtkDialog(GtkDialog);

define _GtkFileChooser(GtkFileChooser);

define _GtkWindow(GtkWindow);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooserGtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser GtkFileChooserDialog;

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);

define _GtkFileChooserFilter(GtkFileFilter);

define _GtkToggleButton(GtkToggleButton);

define _GtkFileFilterInfo(GtkFileFilterInfo);

define _GtkFileChooser(GtkFileChooserDialog);

define _GtkWindow(GtkWindow);

define _GtkFileChooser(GtkFileChooser);
/* --------------------- End of Type definitions from GLIB and GTK --------------------- */

int Fl_GTK_File_Chooser::did_find_GTK_libs = 0;

/* These are the GTK/GLib methods we want to load, but not call by name...! */

fl_g_free = NULL;
fl_g_slist_nth_data = NULL;
fl_g_slist_length = NULL;
fl_g_slist_free = NULL;
fl_gtk_init_check = NULL;
fl_gtk_widget_destroy = NULL;
fl_gtk_file_chooser_set_select_multiple = NULL;
fl_gtk_file_chooser_set_do_overwrite_confirmation = NULL;
fl_gtk_file_chooser_set_current_name = NULL;
fl_gtk_file_chooser_set_current_folder = NULL;
fl_gtk_file_chooser_get_select_multiple = NULL;

Generated by Doxygen
typedef gboolean (*XX_gtk_events_pending)(void); static XX_gtk_events_pending fl_gtk_events_pending = NULL;

GtkWidget * gtk_file_chooser_dialog_new(const gchar *title, GtkWidget *parent, GtkWidget* fl_gtk_file_chooser_dialog_new = NULL;

void gtk_file_chooser_add_filter(GtkFileChooser*, GtkWidget* fl_gtk_file_chooser_add_filter = NULL;

GtkFileFilter* gtk_file_chooser_get_filter(GtkFileChooser*); GtkWidget* fl_gtk_file_chooser_get_filter = NULL;

void gtk_file_chooser_set_filter(GtkFileChooser*, GtkWidget* fl_gtk_file_chooser_set_filter = NULL;

GtkFileFilter* gtk_file_filter_new(); GtkWidget* fl_gtk_file_filter_new = NULL;

void gtk_file_filter_add_pattern(GtkFileFilter*, GtkWidget* fl_gtk_file_filter_add_pattern = NULL;

void gtk_file_filter_add_custom(GtkFileFilter *filter, GtkWidget* fl_gtk_file_filter_add_custom = NULL;

void gtk_file_filter_set_name(GtkFileFilter*, GtkWidget* fl_gtk_file_filter_set_name = NULL;

const gchar* gtk_file_filter_get_name(GtkFileFilter*); GtkWidget* fl_gtk_file_filter_get_name = NULL;

gulong g_signal_connect_data(gpointer, GtkWidget* fl_gtk_file_filter_set_extra_widget = NULL;

gboolean gtk_toggle_button_get_active(GtkToggleButton*); GtkWidget* fl_gtk_toggle_button_get_active = NULL;

void gtk_file_chooser_set_show_hidden(GtkFileChooser*, GtkWidget* fl_gtk_file_chooser_set_show_hidden = NULL;

gboolean gtk_file_chooser_get_show_hidden(GtkFileChooser*); GtkWidget* fl_gtk_file_chooser_get_show_hidden = NULL;

void gtk_toggle_button_set_active(GtkToggleButton*, GtkWidget* fl_gtk_toggle_button_set_active = NULL;

Generated by Doxygen
`Fl_GTK_File_Chooser`: Fl_GTK_File_Chooser(int val) : Fl_FLTK_File_Chooser(-1)

* gtkw_ptr = NULL; // used to hold a GtkWidget*
* gtkw_slist = NULL; // will hold the returned file names in a multi-selection...
* gtkw_count = 0; // How many items were selected?
* gtkw_filename = NULL; // holds the last name we read back in a single file selection...
* gtkw_title = NULL; // dialog title
* _btype = val;
* previous_filter = NULL;

`Fl_GTK_File_Chooser::~Fl_GTK_File_Chooser()`

* if(gtkw_ptr) {
  * fl_gtk_widget_destroy (gtkw_ptr);
  * gtkw_ptr = NULL;
* }
* if(gtkw_filename) {
  * fl_g_free(gtkw_filename);
  * gtkw_filename = NULL;
* }
* if(gtkw_slist) {
  * GSList *iter = (GSList *)gtkw_slist;
  * while(iter) {
    * if(iter->data) fl_g_free(iter->data);
    * iter = g_slist_next(iter);
  } // remove all the items...
  * gtkw_slist = NULL;
* }
* gtkw_count = 0; // assume we have no files selected now
* gtkw_title = strfree(gtkw_title);
* }

`void Fl_GTK_File_Chooser::type(int val)`

* _btype = val;

`int Fl_GTK_File_Chooser::count()`

* return gtkw_count;

`const char *Fl_GTK_File_Chooser::filename()`

* if(gtkw_ptr) {
  * if(fl_gtk_file_chooser_get_select_multiple((GtkFileChooser *)gtkw_ptr) == FALSE) {
    * return gtkw_filename;
  } else {
    * GSList *iter = (GSList *)gtkw_slist;
    * char *nm = (char *)iter->data;
    * return nm;
  }
* } else {
  * return(**);
* }

`const char *Fl_GTK_File_Chooser::filename(int i)`

* if ((unsigned)i < gtkw_count) {
  * GSList *iter = (GSList *)gtkw_slist;
  * char *nm = (char *)fl_g_slist_nth_data(iter, i);
  * return nm;
* }

`void Fl_GTK_File_Chooser::title(const char *val)`

* strfree(gtkw_title);
* gtkw_title = strnew(val);

`const char *Fl_GTK_File_Chooser::title()`

* return gtkw_title;

/* changes the extension of the outfile in the chooser according to newly selected filter */
`void Fl_GTK_File_Chooser::changed_output_type(const char *filter)`
if (!options().&Fl_Native_File_Chooser::USE_FILTER_EXT) return;
if (!strcmp(filter, '\') || strcmp(filter, '\') || strcmp(filter+1, '\') || strncmp(filter, "\.*", 2)) return;
const char *p = fl_gtk_file_chooser_get_filename((GtkFileChooser*)_gtkw_ptr);
if (!p) return;
p = fl_filename_name(p);
const char *q = strchr(p, '.');
if (!q) q = p + strlen(p);
char *r = new char[strlen(p) + strlen(filter)];
strcpy(r, p);
strcpy(r + (q - p), filter + 1);
fl_gtk_file_chooser_set_current_name((GtkFileChooser *)_gtkw_ptr, r);
delete[] r;
}

/* Filters files before display in chooser. Also used to detect when the filter just changed */
 gboolean Fl_GTKFileChooser::custom_gtk_filter_function(const GtkFileFilterInfo *info, Fl_GTKFileChooser::pair* p)
{
if (p->running->previous_filter != p->filter) {
    p->running->changed_output_type(p->filter);
p->running->previous_filter = p->filter;
}
return (gboolean)fl_filename_match(fl_filename_name(info->filename), p->filter);
}

void Fl_GTKFileChooser::free_pair(Fl_GTKFileChooser::pair *p)
{
delete p;
}

static void hidden_files_cb(GtkToggleButton *togglebutton, gpointer user_data)
{
    gboolean state = fl_gtk_toggle_button_get_active(togglebutton);
    fl_gtk_file_chooser_set_show_hidden((GtkFileChooser *)user_data, state);
}

int Fl_GTKFileChooser::show()
{
    // The point here is that after running a GTK dialog, the calling program's current locale is modified.
    // To avoid that, we memorize the calling program's current locale, and the locale as modified
    // by GTK after the first dialog use. We restore the calling program's current locale
    // before returning, and we set the locale as modified by GTK before subsequent GTK dialog uses.
    static bool first = true;
    char *p;
    char *before = NULL;
    static char *gtk_wants = NULL;
    fl_open_display();
    // record in before the calling program's current locale
    p = setlocale(LC_ALL, NULL);
    if (p) before = strdup(p);
    if (gtk_wants) // set the locale as GTK 'wants it'
        setlocale(LC_ALL, gtk_wants);
    int retval = fl_gtk_chooser_wrapper(); // may change the locale
    if (first) {
        first = false;
        // record in gtk_wants the locale as modified by the GTK dialog
        p = setlocale(LC_ALL, NULL);
        if (p) gtk_wants = strdup(p);
    }
    if (before) {
        setlocale(LC_ALL, before); // restore calling program's current locale
        free(before);
    }
    return retval;
}

static char *extract_dir_from_path(const char *path)
{
static char *dir = NULL;
if (fl_filename_isdir(path)) {
    return (char *)path;
}
if ( *path != '/') return NULL;
if (dir) free(dir);
dir = strdup(path);
do {
    char *p = strrchr(dir, '/');
    if (p == dir) p++;
    *p = 0;
} while (!fl_filename_isdir(dir));
return dir;

Generated by Doxygen
static void run_response_handler(GtkDialog *dialog, gint response_id, gpointer data)
{
    gint *ri = (gint *)data;
    *ri = response_id;
}

int Fl_GTK_File_Chooser::fl_gtk_chooser_wrapper()
{
    int result = 1;
    static int have_gtk_init = 0;
    char *p;
    if(!have_gtk_init) {
        have_gtk_init = -1;
        int ac = 0;
        fl_gtk_init_check(&ac, NULL);
    }
    if(gtkw_ptr) { // discard the previous dialog widget
        fl_gtk_widget_destroy (gtkw_ptr);
        gtkw_ptr = NULL;
    }
    // set the dialog action type
    GtkFileChooserAction gtw_action_type;
    switch (_btype) {
    case Fl_Native_File_Chooser::BROWSE_DIRECTORY:
        case Fl_Native_File_Chooser::BROWSE_MULTI_DIRECTORY:
        gtw_action_type = GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER;
        break;
    case Fl_Native_File_Chooser::BROWSE_SAVE_FILE:
        gtw_action_type = GTK_FILE_CHOOSER_ACTION_SAVE;
        break;
    case Fl_Native_File_Chooser::BROWSE_SAVE_DIRECTORY:
        gtw_action_type = GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER;
        break;
    case Fl_Native_File_Chooser::BROWSE_MULTI_FILE:
    case Fl_Native_File_Chooser::BROWSE_FILE:
        default:
        gtw_action_type = GTK_FILE_CHOOSER_ACTION_OPEN;
        break;
    }
    // create a new dialog
    gtkw_ptr = fl_gtk_file_chooser_dialog_new (gtkw_title,
        NULL, /* parent_window */
        gtw_action_type, GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
        gtw_action_type == GTK_FILE_CHOOSER_ACTION_SAVE ||
        gtw_action_type == GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER
            ? GTK_STOCK_SAVE : GTK_STOCK_OPEN,
        GTK_RESPONSE_ACCEPT,
        NULL);
    // did we create a valid dialog widget?
    if(!gtkw_ptr) {
        // fail
        return -1;
    }
    // set the dialog properties
    switch (_btype) {
    case Fl_Native_File_Chooser::BROWSE_MULTI_DIRECTORY:
        case Fl_Native_File_Chooser::BROWSE_MULTI_FILE:
        fl_gtk_file_chooser_set_select_multiple((GtkFileChooser *)gtkw_ptr, TRUE);
        break;
    case Fl_Native_File_Chooser::BROWSE_SAVE_FILE:
        if (_preset_file) fl_gtk_file_chooser_set_current_name ((GtkFileChooser *)gtkw_ptr,
            fl_filename_name(_preset_file));
        break;
    case Fl_Native_File_Chooser::BROWSE_SAVE_DIRECTORY:
        fl_gtk_file_chooser_set_create_folders((GtkFileChooser *)gtkw_ptr, TRUE);
        fl_gtk_file_chooser_set_do_overwrite_confirmation ((GtkFileChooser *)gtkw_ptr,
            (_options & Fl_Native_File_Chooser::SAVEAS_CONFIRM)?TRUE:FALSE);
        break;
    case Fl_Native_File_Chooser::BROWSE_DIRECTORY:
    case Fl_Native_File_Chooser::BROWSE_FILE:
        default:
        break;
    }
    if (_directory && _directory[0]) {
        Generated by Doxygen
    }
p = extract_dir_from_path(_directory);
else if (_preset_file) {
    p = extract_dir_from_path(_preset_file);
    if (p) fl_gtk_file_chooser_set_current_folder((GtkFileChooser *)gtkw_ptr, p);
}

GtkFileFilter **filter_tab = NULL;
if (_parsedfilt) {
    filter_tab = new GtkFileFilter *[_nfilters];
    char *filter = strdup(_parsedfilt);
    p = strtok(filter, " ");
    int count = 0;
    while (p) {
        filter_tab[count] = fl_gtk_file_filter_new();
        fl_gtk_file_filter_set_name(filter_tab[count], p);
        p = strchr(p, '+') + 1;
        char *q = strchr(p, ')'); *q = 0;
        fl_gtk_file_filter_add_custom(filter_tab[count],
            GTK_FILE_FILTER_FILENAME,
            (GtkFileFilterFunc)Fl_GTK_File_Chooser::custom_gtk_filter_function,
            new Fl_GTK_File_Chooser::pair(this, p),
            (GDestroyNotify)Fl_GTK_File_Chooser::free_pair);
        fl_gtk_file_chooser_add_filter((GtkFileChooser *)gtkw_ptr, filter_tab[count]);
        p = strtok(NULL, " ");
        count++;
    }
    free(filter);
    fl_gtk_file_chooser_set_filter((GtkFileChooser *)gtkw_ptr, filter_tab[_filtvalue < _nfilters?_filtvalue:0]);
    previous_filter = NULL;
}

if (_parsedfilt) {
    GtkFileFilter * gfilter = fl_gtk_file_chooser_get_filter((GtkFileChooser *)gtkw_ptr);
    for (_filtvalue = 0; _filtvalue < _nfilters; _filtvalue++)
        if (filter_tab[_filtvalue] == gfilter) break;
}

if (response_id == GTK_RESPONSE_ACCEPT) {
    if (_parsedfilt) {
        GtkFileFilter *gfilter = fl_gtk_file_chooser_get_filter((GtkFileChooser *)gtkw_ptr);
        for (_filtvalue = 0; _filtvalue < _nfilters; _filtvalue++)
            if (filter_tab[_filtvalue] == gfilter) break;
    }
}
```c
00582  GSList *iter = (GSList *)gtkw_slist;
00583  while(iter) {
00584      if(iter->data) fl_g_free(iter->data);
00585      iter = g_slist_next(iter);
00586  }
00587  fl_g_slist_free((GSList *)gtkw_slist);
00588  gtkw_slist = NULL;
00589 }
00590  gtkw_count = 0; // assume we have no files selected now
00591  
00592  if(fl_gtk_file_chooser_get_select_multiple((GtkFileChooser *)gtkw_ptr) == FALSE) {
00593      gtkw_filename = fl_gtk_file_chooser_get_filename ((GtkFileChooser *)gtkw_ptr);
00594      if (gtkw_filename) {
00595          gtkw_count = 1;
00596          result = 0;
00597          //printf("single: %s\n", gtkw_filename);
00598      }
00599  }
00600  else {
00601      gtkw_slist = fl_gtk_file_chooser_get_filenames((GtkFileChooser *)gtkw_ptr);
00602      gtkw_count = fl_g_slist_length((GSList *)gtkw_slist);
00603      if(gtkw_count) result = 0;
00604      // puts("multiple");
00605      // GSList *iter = (GSList *)gtkw_slist;
00606      // printf (*Selected %d files\n", gtkw_count);
00607      // while(iter) {
00608      //     char *nm = (char *)iter->data;
00609      //     printf("%s\n", nm);
00610      //     iter = g_slist_next(iter);
00611      // }
00612      // }
00614  }
00615  delete[] filter_tab;
00616  if ( response_id == GTK_RESPONSE_DELETE_EVENT) gtkw_ptr = NULL;
00617  else fl_gtk_widget_hide (gtkw_ptr);
00618  
00619  // I think this is analogous to doing a Fl::check() - we need this here to make sure
00620  // the GtkFileChooserDialog is removed from the display correctly
00621  while (fl_gtk_events_pending ()) fl_gtk_main_iteration ();
00622  return result;
00623 } // fl_gtk_chooser_wrapper
00624
00625 #if HAVE_DLSYM && HAVE_DLFCN_H
00626 # define GET_SYM(SSS, LLL)
00627     dlerror(); /* Clear any existing error */
00628     fl_##SSS = (XX_##SSS)dlsym(LLL, #SSS);
00629     if ((pc_dl_error = dlerror()) != NULL) {
00630         fprintf(stderr, "%s\n", pc_dl_error);
00631         did_find_GTK_libs = 0;
00632         return; }
00633     static void* fl_dlopen(const char *filename1, const char *filename2)
00634     { void *ptr = dlopen(filename1, RTLD_LAZY | RTLD_GLOBAL);
00635         if (!ptr) ptr = dlopen(filename2, RTLD_LAZY | RTLD_GLOBAL);
00636         return ptr; }
00637 #endif
00638 /*
00639  * Use dlopen to see if we can load the gtk dynamic libraries that
00640  * will allow us to create a GtkFileChooserDialog() on the fly,
00641  * without linking to the GTK libs at compile time.
00642 */
00643 #ifdef FL_GTK_FILE_CHOOSER_PROBE_FOR_GTK_LIBS
00644  #if HAVE_DLSYM & HAVE_DLFCN_H
00645  #define GET_SYM(SSS, LLL)
00646  
00647  #ifdef __APPLE_CC__ // allows testing on Darwin + X11
00648  #elif defined (__APPLE_CC__)
00649  
00650  #ifdef __APPLE_CC__ // allows testing on Darwin + X11
00651  #elif defined __APPLE_CC__
00652  
00653  #endif
00654  
00655  #endif
00656  
00657  #endif
00658  
00659  
00660  
00661  
00662  
00663  
00664  
00665  
00666  
00667  
00668  
```

00569 } 
00570 else // Try then with GTK3  
00571 ptr_gtk = fl_dlopen("libgtk-3.so", "libgtk-3.so.0");  
00572 #ifdef DEBUG  
00573 if (ptr_gtk && ptr_glib) {  
00574 puts("selected GTK-3\n");  
00575 }  
00576 #endif  
00577 }  
00578 else if(!ptr_glib) || (!ptr_gtk)) {  
00579 #ifdef DEBUG  
00580 puts("Failure to load libglib or libgtk");  
00581 #endif  
00582 did_find_GTK_libs = 0;  
00583 return;  
00584  
00585 }  
00586  
00587 char *pc_dl_error; // used to report errors by the GET_SYM macro...  
00588 // items we need from GLib  
00589 GET_SYM(g_free, ptr_glib);  
00590 GET_SYM(g_slist_nth_data, ptr_glib);  
00591 GET_SYM(g_slist_length, ptr_glib);  
00592 GET_SYM(g_slist_free, ptr_glib);  
00593 // items we need from GTK  
00594 GET_SYM(gtk_init_check, ptr_gtk);  
00595 GET_SYM(gtk_widget_destroy, ptr_gtk);  
00596 GET_SYM(gtk_file_chooser_set_select_multiple, ptr_gtk);  
00597 GET_SYM(gtk_file_chooser_set_do_overwrite_confirmation, ptr_gtk);  
00598 GET_SYM(gtk_file_chooser_set_current_name, ptr_gtk);  
00599 GET_SYM(gtk_file_chooser_set_current_folder, ptr_gtk);  
00600 GET_SYM(gtk_file_chooser_set_create_folders, ptr_gtk);  
00601 GET_SYM(gtk_file_chooser_set_get_select_multiple, ptr_gtk);  
00602 GET_SYM(gtk_widget_hide, ptr_gtk);  
00603 GET_SYM(gtk_file_chooser_get_filename, ptr_gtk);  
00604 GET_SYM(gtk_file_chooser_get_filenames, ptr_gtk);  
00605 GET_SYM(gtk_main_iteration, ptr_gtk);  
00606 GET_SYM(gtk_events_pending, ptr_gtk);  
00607 GET_SYM(gtk_file_chooser_dialog_new, ptr_gtk);  
00608 GET_SYM(gtk_file_chooser_add_filter, ptr_gtk);  
00609 GET_SYM(gtk_file_chooser_set_filter, ptr_gtk);  
00610 GET_SYM(gtk_file_filter_new, ptr_gtk);  
00611 GET_SYM(gtk_file_filter_add_pattern, ptr_gtk);  
00612 GET_SYM(gtk_file_filter_add_custom, ptr_gtk);  
00613 GET_SYM(gtk_file_filter_set_name, ptr_gtk);  
00614 GET_SYM(gtk_file_filter_get_name, ptr_gtk);  
00615 GET_SYM(gtk_file_chooser_set_extra_widget, ptr_gtk);  
00616 GET_SYM(gtk_widget_show_now, ptr_gtk);  
00617 GET_SYM(gtk_file_chooser_set_extra_widget, ptr_gtk);  
00618 GET_SYM(gtk_check_button_new_with_label, ptr_gtk);  
00619 GET_SYM(gtk_signal_connect_data, ptr_gtk);  
00620 GET_SYM(gtk_toggle_button_get_active, ptr_gtk);  
00621 GET_SYM(gtk_file_chooser_set_show_hidden, ptr_gtk);  
00622 GET_SYM(gtk_file_chooser_get_show_hidden, ptr_gtk);  
00623 GET_SYM(gtk_file_chooser_set_extra_widget, ptr_gtk);  
00624 did_find_GTK_libs = 1;  
00625 #endif  
00626 }  
00627 // probe_for_GTK_libs  
00628  
00629 // End of "$Id$".  
00630  

10.190  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>  

10.190.1  Detailed Description  

implementation of class Fl_Paged_Device.
10.191  fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.

```cpp
#include <config.h>
#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>
#include <FL/x.H>
```

Functions

- **Fl_Region XRectangleRegion** (int x, int y, int w, int h)

Variables

- int fl_line_width_

10.191.1  Detailed Description

Drawing and clipping routines for rectangles.

10.192  fl_set_fonts_x.cxx

0001 //
0002 // "$Id$"
0003 //
0004 // X11 font utilities for the Fast Light Tool Kit (FLTK).
0005 //
0006 // Copyright 1998-2010 by Bill Spitzak and others.
0007 //
0008 // This library is free software. Distribution and use rights are outlined in
0009 // the file "COPYING" which should have been included with this file. If this
0010 // file is missing or damaged, see the license at:
0011 //
0012 // http://www.fltk.org/COPYING.php
0013 //
0014 // Please report all bugs and problems on the following page:
0015 //
0016 // http://www.fltk.org/str.php
0017 //
0018 // This function fills in the fltk font table with all the fonts that
0019 // are found on the X server. It tries to place the fonts into families
0020 // and to sort them so the first 4 in a family are normal, bold, italic,
0021 // and bold italic.
0022 //
0023 // Standard X fonts are matched by a pattern that is always of
0024 // this form, and this pattern is put in the table:
0025 // "*-family-weight-slant-width1-style-*-registry-encoding"
0026 //
0027 // Non-standard font names (those not starting with '-') are matched
0028 // by a pattern of the form "prefix*suffix", where the '*' is where
0029 // fltk thinks the point size is, or by the actual font name if no
0030 // point size is found.
0031 //
0032 // Fltk knows how to pull an "attribute" out of a font name, such as
0033 // bold or italic, by matching known x font field values. All words
0034 // that don't match a known attribute are combined into the "name"
0035 // of the font. Names are compared before attributes for sorting, this
0036 // makes the bold and plain version of a font come out next to each
0037 // other despite the poor X font naming scheme.
0038 //
0039 // By default fl_set_fonts() only does iso8859-1 encoded fonts. You can
0040 // do all normal X fonts by passing "-*" or every possible font with "*".
0041 //
0042 // Fl::set_font will take strings other than the ones this stores
0043 // and can identify any font on X that way. You may want to write your
0044 // own system of font management and not use this code.
0045 //
0046 // turn word N of a X font name into either some attribute bits
0047 // (right now 0, FL_BOLD, or FL_ITALIC), or into -1 indicating that
0048 // the word should be put into the name:
static int attribute(int n, const char *p) {
// don't put blank things into name:
if (! *p || *p=='-' || *p=='*') return 0;
if (n == 3) // weight
if (!strncmp(p,"normal",6) ||
!strncmp(p,"light",5) ||
!strncmp(p,"medium",6) ||
!strncmp(p,"book",4)) return 0;
if (!strncmp(p,"bold",4)) return FL_BOLD;
} else if (n == 4) // slant
if (*p == 'r') return 0;
if (*p == 'i' || *p == 'o') return FL_ITALIC;
} else if (n == 5) // sWidth
if (!strncmp(p,"normal",6)) return 0;
return -1;
}

// return non-zero if the registry-encoding should be used:
extern const char* fl_encoding;
static int use_registry(const char *p) {
return *p && *p!='*' && strcmp(p,fl_encoding);
}

// Bug: older versions calculated the value for *ap as a side effect of
// making the name, and then forgot about it. To avoid having to change
// the header files I decided to store this value in the last character
#define ENDOFBUFFER 127 // sizeof(Fl_Font.fontname)-1

// turn a stored (with *'s) X font name into a pretty name:
const char* Fl::get_font_name(Fl_Font fnum, int* ap) {
Fl_Fontdesc *f = fl_fonts + fnum;
if (!f->fontname[0]) {
int type = 0;
const char * p = f->name;
if (!p) {
if (ap) *ap = 0;
return "";
}
char *o = f->fontname;
if ( *p != '-') { // non-standard font, just replace * with spaces:
if (strstr(p,"bold")) type = FL_BOLD;
if (strstr(p,"ital")) type |= FL_ITALIC;
for (; *p; p++) {
if ( *p == ' ' || *p == '-' || *p == 'o')
if (! *p) break;
if (o < (f->fontname + ENDOFBUFFER - 1)) *o++ = ' '; 
else {
strncpy(o,p,ENDOFBUFFER);
}
if (o++) *o = 0;
}
} else { // standard dash-separated font:
// get the family:
const char *x = fl_font_word(p,2); if (*x) x++; if (*x=='*') x++;
if (*x) {
if (ap) *ap = 0;
return p;
}
// collect all the attribute words:
for (int n = 3; n <= 6; n++) {
// get the next word:
int t = attribute(n,x);
if (t < 0) {
if (o < (f->fontname + ENDOFBUFFER - 1)) *o++ = ' '; 
else {
strncpy(o,x,ENDOFBUFFER);
}
if (o = f->fontname+ENDOFBUFFER-1;)
o = f->fontname+ENDOFBUFFER-1;
} else {
strncpy(f->fontname, x, ENDOFBUFFER);
}
if ((o - x) < (int)(ENDOFBUFFER - 1)) {
MRS: we want strncpy here, not strlcpy...
strncpy(o,x,o-x);
}
else 
strncpy(f->fontname, x, ENDOFBUFFER);
}
} else {
// standard dash-separated font:
// get the family:
const char *x = fl_font_word(p,2); if (*x) x++; if (*x=='*') x++;
if (*x) {
if (ap) *ap = 0;
return p;
}
// collect all the attribute words:
for (int n = 3; n <= 6; n++) {
// get the next word:
int t = attribute(n,x);
if (t < 0) {
if (o < (f->fontname + ENDOFBUFFER - 1)) *o++ = ' '; 
else {
strncpy(o,x,o-x);
}
} else 
}
strncpy(o, x, ENDOFBUFFER - (o - f->fontname) - 1);
00139 o = f->fontname + ENDOFBUFFER - 1;
00140 }

00141 else type |= t;
00142 }
00143 // skip over the '*' for the size and get the registry-encoding:
00144 x = fl_font_word(e, 2);
00145 if (*x) {x++; o++ = '('; while (*x) *o++ = *x++; *o++ = ')';}
00147 *o = 0;
00148 if (type & FL_BOLD) strlcat(f->fontname, " bold", ENDOFBUFFER);
00149 if (type & FL_ITALIC) strlcat(f->fontname, " italic", ENDOFBUFFER);

00151 f->fontname[ENDOFBUFFER] = (char)type;
00152 }
00153 if (ap) *ap = f->fontname[ENDOFBUFFER];
00154 return f->fontname;
00155 }
00156 }
00157 extern "C" {
00158 static int ultrasort(const void *aa, const void *bb) {
00159 const char *a = *(char **)aa;
00160 const char *b = *(char **)bb;

00161 // sort all non x-fonts at the end:
00162 if (*a != '-') {
00163 if (b == '-') return 1;
00164 if (*a == '-') return 0;
00165 if (isdigit(*a) && isdigit(*b)) {
00166 int na = strtol(a, (char **)&a, 10);
00167 int nb = strtol(b, (char **)&b, 10);
00168 if (strcmp(a, b)) return *a - *b;
00169 } else if (!a || !b) {
00170 return (a ? -1 : b ? 1 : 0);
00171 }
00172 for (a++, b++; *a && *a != '-');
00173 for (b++, *b && *b != '-');

00174 // compare the family and all the attribute words:
00175 int atype = 0;
00176 int btype = 0;
00177 for (int n = 2; n <= 6; n++) {
00178 if (at = attribute(n, a))
00179 if (bt = attribute(n, b))
00180 if (at < 0) if (bt >= 0) return 1;
00181 b = fl_font_word(b, 1);
00182 if (bt < 0) return -1;
00183 a = fl_font_word(a, 1);
00184 if (at < 0) return -1;
00185 if (bt < 0) return -1;
00186 a = at; btype = bt;
00187 }
00188 // remember the pixel size:
00189 int asize = atoi(a);
00190 int bsize = atoi(b);
00191 // compare the registry/encoding:
00192 a = fl_font_word(a, 6);
00193 b = fl_font_word(b, 6);
00194 if (strcmp(a, b)) return (*a != *b) ? *a == *b+1 : 0;
00195 return (asize < bsize) - (bsize <= asize);
00196 if (use_registry(a))
00197 if (!use_registry(b)) return 1;
00198 else if (use_registry(b)) return -1;
00199 else if (*a == *b+1) {
00200 a++; b++;
00201 if (atype != btype) return atype - btype;
00202 if (asize != bsize) return asize - bsize;
// something wrong, just do a string compare...
return strcmp(*(char**)aa, *(char**)bb);
}

// converts a X font name to a standard starname, returns point size:
static int to_canonical(char *to, const char *from, size_t tolen) {
  char * c = fl_find_fontsize((char*)from);
  if (!c) return -1; // no point size found...

  const char * endptr;
  int size = strtol(c,(char **)&endptr,10);
  if (from[0] == '-') {
    // replace the "foundry" with -*-:
    *to++ = '-'; *to++ = '*';
    for (from++; *from && *from != '-'; from++);
  // skip to the registry-encoding:
  endptr = (char*)fl_font_word(endptr,6);
  if (*endptr && !use_registry(endptr+1)) endptr = "";

  int n = c-from;
  // MBS: we want strncpy here, not strlcpy...
  if (n > (int)(tolen - 1)) return -1;
  strncpy(to,from,n);
  to[n++] = ' *';
  strlcpy(to+n,endptr, tolen - n);
  return size;
}

static unsigned int fl_free_font = FL_FREE_FONT;

Fl_Font Fl::set_fonts(const char* xstarname) {
  if (fl_free_font > (unsigned)FL_FREE_FONT) // already been here
    return (Fl_Font)fl_free_font;
  fl_open_display();
  int xlistsize;
  char buf[20];
  if (!xstarname) {
    strcpy(buf,"- *-"); strcpy(buf+3,fl_encoding);
    xstarname = buf;
  }
  char **xlist = XListFonts(fl_display, xstarname, 10000, &xlistsize);
  if (!xlist) return (Fl_Font)fl_free_font;
  qsort(xlist, xlistsize, sizeof( *xlist), ultrasort);
  int used_xlist = 0;
  for (int i=0; i<xlistsize;)
    if ((xstarname) [i][0] == '-') strlcpy(buf,2,fl_encoding);
  xstarname = buf;

  char **xlist = XListFonts(fl_display, xstarname, 10000, &xlistsize);
  if (!xlist) return (Fl_Font)fl_free_font;
  qsort(xlist, xlistsize, sizeof(*xlist), ultrasort);
  int used_xlist = 0;
  for (int i=0; i<xlistsize;)
    if (i_first_xlist = i;)
      if (xlist[i][0] == '-';
      strlcpy(buf+2,fl_encoding);
  char xstarname[1024];
  int size = to_canonical(xstarname, p, sizeof(xstarname));
  if (size >= 0) {
    for (;j;) { // find all matching fonts:
      if (j >= xlistsize) break;
      const char *q = xlist[i];
      char this_canon[1024];
      if (to_canonical(this_canon, q, sizeof(this_canon)) < 0) break;
      if (strcmp(canon, this_canon)) break;
      i++;
    }
    unsigned int j;
    for (j = 0; j++) {
      /if (p == cannon) p = cannon;
      if (!p) p = strdup(p);
      p = cannon;
      if (j < FL_FREE_FONT) {
        /if (j) if (fl.Fonts[j].name && !strcmp(fl.Fonts[j].name, p)) break;
        /} else {/}
        j = fl_free_font++;
        /if (j) XFreeFonts(xlist);
      }
      return (Fl_Font)fl_free_font;
  }

int Fl::get_font_sizes(Fl_Font fnum, int*& sizep) {
  static Fl_Fontdesc * s = fl.Fonts+fnum;
  int sizep = 0;
  if (!s) return 0; // empty slot in table, use entry 0
  }

Generated by Doxygen
00312  fl_open_display();
00313  s->xlist = XListFonts(fl_display, s->name, 100, &(s->n));
00314  if (!s->xlist) return 0;
00315  }
00316  int listsizesize = s->n; if (listsizesize<0) listsizesize = -listsizesize;
00317  static int sizes[128];
00318  int numsizes = 0;
00319  for (int i = 0; i < listsizesize; i++) {
00320    char *q = s->xlist[i];
00321    char *d = fl_find_fontsize(q);
00322    if (!d) continue;
00323    int s = strtol(d,0,10);
00324    if (!numsizes || sizes[numsizes-1] < s) {
00325      sizes[numsizes++] = s;
00326    } else {
00327      // insert-sort the new size into list:
00328      int n;
00329      for (n = numsizes-1; n > 0; n--) if (sizes[n-1] < s) break;
00330      if (sizes[n] != s) {
00331        for (int m = numsizes; m > n; m--) sizes[m] = sizes[m-1];
00332        sizes[n] = s;
00333        numsizes++;
00334      }
00335    }
00336  }
00337  sizep = sizes;
00338  return numsizes;
00339  }
00340
00341  //
00342  // End of "$Id$".
00343  //

10.193  fl_vertex.cxx File Reference

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.
#include <config.h>
#include <FL/fl_draw.H>
#include <FL/X.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>

10.193.1  Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

10.194  Fl_XColor.H

00001  //
00002  // "$Id$"
00003  //
00004  // X-specific color definitions for the Fast Light Tool Kit (FLTK).
00005  //
00006  // Copyright 1998-2010 by Bill Spitzak and others.
00007  //
00008  // This library is free software. Distribution and use rights are outlined in
00009  // the file "COPYING" which should have been included with this file. If this
00010  // file is missing or damaged, see the license at:
00011  //
00012  // http://www.fltk.org/COPYING.php
00013  //
00014  // Please report all bugs and problems on the following page:
00015  //
00016  // http://www.fltk.org/str.php
00017  //
00018  //
00019  #include <config.h>
00020  #include <FL/Enumerations.H>
00021  //
00022  // one of these for each color in fltk's "colormap":
00023  // if overlays are enabled, another one for the overlay
00024  struct Fl_XColor {
00025    unsigned char r,g,b; // actual color used by X
00026    unsigned char mapped; // true when XAllocColor done

Generated by Doxygen
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;

// End of "$Id$".

#ifndef flstring_h
#define flstring_h

#include <FL/Fl_Export.H>
#include <config.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.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... */

#ifndef __cplusplus
extern "C" {
#endif

FL_EXPORT extern int fl_snprintf(char *, size_t, const char *, ...);
#ifndef HAVE_SNPRINTF
#define snprintf fl_snprintf
#endif

FL_EXPORT extern int fl_vsnprintf(char *, size_t, const char *, va_list ap);
#ifndef HAVE_VSNPRINTF
#define vsnprintf fl_vsnprintf
#endif

#ifndef __cplusplus
}
#endif

Generated by Doxygen
strlcpy() and strlcat() are some really useful BSD string functions that work the way strncpy() and strncat() *should* have worked.

```c
#include <string.h>

FL_EXPORT extern size_t fl_strlcat(char *, const char *, size_t);
#endif /* !HAVE_STRLCAT */

FL_EXPORT extern size_t fl_strlcpy(char *, const char *, size_t);
#endif /* !HAVE_STRLCPY */

locale independent ascii compare, does not introduce locale
```
# freeglut_teapot_data.h

```
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 }, /* handle */
{ 48, 49, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73 }, /* lid */
{ 0, 1, 2, 3, 106, 107, 108, 109, 110, 111, 111, 112, 113, 114, 115, 116, 117 }, /* bottom */
{ 118, 118, 118, 118, 124, 122, 119, 121, 123, 126, 125, 120, 40, 39, 38, 37 }, /* spout */
{ 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }, /* handle */
{ 55, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67 }, /* spout */
{ 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 };
};

static double tex[][2][2] =

{ {0.0, 0.0}, {1.0, 0.0} };
```

---

Generated by Doxygen

```
10.196 freeglut_teapot_data.h
1875
00053  * SAVINGS OR REVENUE, OR THE CLAIMS OF THIRD PARTIES, WHETHER OR
00054  * NOT SILICON GRAPHICS, INC. HAS BEEN ADVISED OF THE POSSIBILITY
00055  * OF SUCH LOSS, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
00056  * ARISING OUT OF OR IN CONNECTION WITH THE POSSESSION, USE OR
00057  * PERFORMANCE OF THIS SOFTWARE.
00058  *
00059  * US Government Users Restricted Rights
00060  *
00061  * Use, duplication, or disclosure by the Government is subject to
00062  * restrictions set forth in FAR 52.227.19(c)(2) or subparagraph
00063  *(c)(1)(ii) of the Rights in Technical Data and Computer
00064  * Software clause at DFAR 252.227-7013 and/or in similar or
00065  * successor clauses in the FAR or the DOD or NASA FAR
00066  * Supplement. Unpublished-- rights reserved under the copyright
00067  * laws of the United States. Contractor/manufacturer is Silicon
00068  * Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA
00069  * 94039-7311.
00070  *
00071  * OpenGL(TM) is a trademark of Silicon Graphics, Inc.
00072  */
00073 */
00074 */
00075  */
00076  */
00077 */
00078 static int patchdata[][16] =
00079  { 102, 103, 104, 105, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 }, /* rim */
00080  { 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 }, /* body */
00081  { 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40 }, /* handle */
00082  { 48, 49, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73 }, /* lid */
00083  { 0, 1, 2, 3, 106, 107, 108, 109, 110, 111, 111, 112, 113, 114, 115, 116, 117 }, /* bottom */
00084  { 118, 118, 118, 118, 124, 122, 119, 121, 123, 126, 125, 120, 40, 39, 38, 37 }, /* spout */
00085  { 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }, /* handle */
00086  { 55, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67 }, /* spout */
00087  { 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83 }, /* spout */
00088  { 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95 };
00089  }
00090  
00091  static double tex[][2][2] =
00092  { {0.0, 0.0}, {1.0, 0.0} };
```
00140  { {0.0, 1.0}, {1.0, 1.0} }
00141 }
00142
00143 #endif /* FREEGLUT_TEAPOT_DATA_H */
00144
10.197 mediumarrow.h

00001 #define mediumarrow_width 16
00002 #define mediumarrow_height 16
00003 static const unsigned char mediumarrow_bits[] = {
00004 0x40, 0x00, 0x60, 0x00, 0x70, 0x00, 0x78, 0x00, 0x80, 0xfc, 0x3f, 0x78, 0x00,
00005 0x70, 0x00, 0x60, 0x02, 0x40, 0x06, 0x00, 0x0e, 0x00, 0x1e, 0xfc, 0x3f,
00006 0x00, 0x1e, 0x00, 0x0e, 0x00, 0x06, 0x00, 0x06, 0x00, 0x02};

10.198 print_panel.h

00001 //
00002 // "$Id$"
00003 //
00004 // Print panel for the Fast Light Tool Kit (FLTK).
00005 //
00006 // Copyright 1998-2010 by Bill Spitzak and others.
00007 //
00008 // This library is free software. Distribution and use rights are outlined in
00009 // the file "COPYING" which should have been included with this file. If this
00010 // file is missing or damaged, see the license at:
00011 //
00012 // http://www.fltk.org/COPYING.php
00013 //
00014 // Please report all bugs and problems on the following page:
00015 //
00016 // http://www.fltk.org/str.php
00017 //
00018 //
00019 // This is a temporary file. It is only for development and will
00020 // probably be removed later.
00021 //
00022 //
00023 ifndef print_panel_h
00024 #define print_panel_h
00025 #include <FL/Fl.H>
00026 #include <FL/Fl_Double_Window.H>
00027 #include <FL/Fl_Group.H>
00028 #include <FL/Fl_Choice.H>
00029 #include <FL/Fl_Button.H>
00030 #include <FL/Fl_Box.H>
00031 #include <FL/Fl_Round_Button.H>
00032 #include <FL/Fl_Input.H>
00033 #include <FL/Fl_Spinner.H>
00034 #include <FL/Fl_Check_Button.H>
00035 #include <FL/Fl_Return_Button.H>
00036 #include <FL/Fl_Progress.H>
00037 enum printing_style {SystemV, BSD};
00038 static Fl_Double_Window* make_print_panel();
00039 static void print_cb(Fl_Return_Button *, void *);
00040 static void print_update_status();
00041 static void print_update_status();
00042 static void print_update_status();
00043 #endif
00044
10.199 scandir_posix.c

00001 /*
00002 * "$Id$"
00003 */
00004 * This implementation of 'scandir()' is intended to be POSIX.1-2008 compliant.
00005 * A POSIX.1-1990 compliant system is required as minimum base.
00006 * Note:
00007 * The 'const' declarations were removed to match FLTK 1.3 wrapper (STR #2931)
00008 *
00009 * Copyright (c) 2013 by Michael Bauerle
00010 *
00011 * This library is free software. Distribution and use rights are outlined in

Generated by Doxygen
#scandir_posix.c

```
/* 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 */

#include <sys/types.h> /* XPG2 require this for '*dir()' functions */
#include <dirent.h>
#include <errno.h>
#include <stdlib.h> /* For 'malloc()', 'realloc()' and 'qsort()' */
#include <stddef.h> /* For 'offsetof()', 'NULL' and 'size_t' */
#include <limits.h> /* For 'INT_MAX' */
#include <string.h> /* For 'memcpy()' */

#define ENTRIES_MIN (size_t) 32

static pthread_mutex_t scandir_mutex = PTHREAD_MUTEX_INITIALIZER;

/* This function reads the next entry from the directory referenced by 'dirp',
allocate a buffer for the entry and copy it into this buffer.
On sucess zero is returned and the caller is responsible for 'free()'ing the
buffer after use.
On error the return value is nonzero, 'entryp' and 'len' are invalid.
Should be declared as 'static inline' if the compiler support that.
*/
static int
readentry(DIR *dirp, struct dirent **entryp, size_t *len)
{
  int result = -1;
  struct dirent *e;

  if (!pthread_mutex_lock(&scandir_mutex))
  {
    /* Ensure that there is no code path that bypass the '_unlock()' call! */
    errno = 0;
    e = readdir(dirp);
    if (NULL == e)
      if (!errno)
        *entryp = NULL;
      else
        *entryp = NULL;
    else
      {
        *entryp = e;
        *len = (size_t) strlen(e->d_name) + 1;
        result = 0;
      }

  /* No more entries in directory */
  return result;
}
```

Generated by Doxygen
```c
/* Entry found, allocate local buffer */
*len = offsetof(struct dirent, d_name) + strlen(e->d_name) + (size_t) 1;
*entrp = (struct dirent *) malloc(*len);
if (NULL != *entrp) {
    memcpy((void *) *entrp, (void *) e, *len);
    /* Force NUL termination at end of buffer */
    *(char *) *entrp + *len - (size_t) 1] = 0;
    result = 0;
}
}
#else /* HAVE_PTHREAD */
*
/* In a multithreading environment the systems dirent buffer may be shared */
/* between all threads. Therefore the mutex must stay locked until we have */
/* copied the data to our thread local buffer. */
pthread_mutex_unlock(&scandir_mutex);
#endif /* HAVE_PTHREAD */

return result;

/* ========================================================================== */
int
fl_scandir(const char *dir, struct dirent ***namelist,
int ( *sel)(struct dirent *),
int ( *compar)(struct dirent **, struct dirent **))
{
    int result = -1;
    DIR *dirp;
    size_t len, num = 0, max = ENTRIES_MIN;
    struct dirent *entryp, **entries, **p;

    entries = (struct dirent **) malloc(sizeof(*entries) * max);
    if (NULL != entries)
    {
        /* Open directory 'dir' (and verify that it really is a directory) */
        dirp = opendir(dir);
        if (NULL != dirp)
        {
            /* Read next directory entry */
            while (!readentry(dirp, &entryp, &len))
            {
                if (NULL == entryp)
                {
                    /* EOD => Return number of directory entries */
                    result = (int) num;
                    break;
                }
                /* Apply select function if there is one provided */
                if (NULL != sel) { if (!sel(entryp)) continue; }
                entries[num++] = entryp;
                if (num >= max)
                {
                    /* Allocate exponentially increasing sized memory chunks */
                    if (INT_MAX / 2 >= (int) max) { max *= (size_t) 2; }
                    else
                    {
                        errno = ENOMEM;
                        break;
                    }
                }
                /* Sort entries in array if there is a compare function provided */
                if (NULL != compar) { if (!compar(entryp)) continue; }
                entries[num++] = entryp;
                if (num >= max)
                {
                    /* Allocate exponentially increasing sized memory chunks */
                    if (INT_MAX / 2 >= (int) max) { max *= (size_t) 2; }
                    else
                    {
                        errno = ENOMEM;
                        break;
                    }
                }
                p = (struct dirent **) realloc((void *) entries,
                    sizeof(*entries) * max);
                if (NULL != p) { entries = p; }
                else break;
            }
            closedir(dirp);
        }
    }
    /* A standard compliant ‘closedir()’ is allowed to fail with ‘EINVAL’, but */
    /* the state of the directory structure is undefined in this case. */
    /* Therefore we ignore the return value because we can’t call ‘closedir()’ */
    /* again and must hope that the system has released all resources. */
    /* */
    /* Sort entries in array if there is a compare function provided */
    if (NULL != compar) { if (!compar(entryp)) continue; }
```

Generated by Doxygen

1878 File Documentation
```c
qsort(void * entries, num, sizeof(*entries),
      (int (*)(const void *, const void *)) compar);

*namelist = entries;

/* Check for error */
if (-1 == result) {
    /* Free all memory we have allocated */
    while (num--) { free(entries[num]); }
    free(entries);
}
return result;
```

---

```c
#define slowarrow_width 16
#define slowarrow_height 16
static const unsigned char slowarrow_bits[] = {
    0x40, 0x00, 0x40, 0x00, 0x60, 0x00, 0x60, 0x00, 0xf0, 0x0f, 0x60, 0x00,
    0x60, 0x00, 0x40, 0x02, 0x40, 0x02, 0x00, 0x06, 0x00, 0x06, 0xf0, 0x0f,
    0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x00, 0x02, 0x00, 0x02};
```

---

```c
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,
```

---

*End of "$Id$".

---
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,
    GC gc,
    int x,
    int y,
    const char *string,
    int num_bytes);

void XUtf8DrawImageString(
    Display *display,
    Drawable d,
    XUtf8FontStruct *font_set,
    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);

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 XUtf8LookupString Xutf8LookupString

else
int
XUtf8LookupString(
XIC ic,
XKeyPressEvent* event,
char* buffer_return,
int bytes_buffer,
KeySym* keysym,
Status* status_return);

unsigned short
XUtf8IsNonSpacing(
unsigned int ucs);

unsigned short
XUtf8IsRightToLeft(
unsigned int ucs);

int
XUtf8Tolower(
int ucs);

int
XUtf8Toupper(
int ucs);

#endif

# ifdef __cplusplus
}
#endif

End of "$Id$".

--

*/

00001 /* spacing */
00002
00003 static const unsigned short ucs_table_0041[] = {
00004 /* U+0041 */ 0x0061,
00005 /* U+0042 */ 0x0062,
00006 /* U+0043 */ 0x0063,
00007 /* U+0044 */ 0x0064,
00008 /* U+0045 */ 0x0065,
00009 /* U+0046 */ 0x0066,
00010 /* U+0047 */ 0x0067,
00011 /* U+0048 */ 0x0068,
00012 /* U+0049 */ 0x0069,
00013 /* U+004A */ 0x006A,
00014 /* U+004B */ 0x006B,
00015 /* U+004C */ 0x006C,
00016 /* U+004D */ 0x006D,
00017 /* U+004E */ 0x006E,
00018 /* U+004F */ 0x006F,
00122 0x00,
00123 0x00,
00124 0x00,
00125 0x00,
00126 0x00,
00127 0x00,
00128 0x00,
00129 0x00,
00130 0x00,
00131 /* U+00C0 */ 0x00E0,
00132 /* U+00C1 */ 0x00E1,
00133 /* U+00C2 */ 0x00E2,
00134 /* U+00C3 */ 0x00E3,
00135 /* U+00C4 */ 0x00E4,
00136 /* U+00C5 */ 0x00E5,
00137 /* U+00C6 */ 0x00E6,
00138 /* U+00C7 */ 0x00E7,
00139 /* U+00C8 */ 0x00E8,
00140 /* U+00C9 */ 0x00E9,
00141 /* U+00CA */ 0x00EA,
00142 /* U+00CB */ 0x00EB,
00143 /* U+00CC */ 0x00EC,
00144 /* U+00CD */ 0x00ED,
00145 /* U+00CE */ 0x00EE,
00146 /* U+00CF */ 0x00EF,
00147 /* U+00D0 */ 0x00F0,
00148 /* U+00D1 */ 0x00F1,
00149 /* U+00D2 */ 0x00F2,
00150 /* U+00D3 */ 0x00F3,
00151 /* U+00D4 */ 0x00F4,
00152 /* U+00D5 */ 0x00F5,
00153 /* U+00D6 */ 0x00F6,
00154 0x00,
00155 /* U+00D7 */ 0x00F7,
00156 /* U+00D8 */ 0x00F8,
00157 /* U+00D9 */ 0x00F9,
00158 /* U+00DA */ 0x00FA,
00159 /* U+00DB */ 0x00FB,
00160 /* U+00DC */ 0x00FC,
00161 /* U+00DD */ 0x00FD,
00162 0x00,
00163 0x00,
00164 0x00,
00165 0x00,
00166 0x00,
00167 0x00,
00168 0x00,
00169 0x00,
00170 0x00,
00171 0x00,
00172 0x00,
00173 0x00,
00174 0x00,
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 /* U+0100 */ 0x0101,
00209 /* U+010E */ 0x010F,
00210 0x00,
00211 /* U+0110 */ 0x0111,
00212 0x00,
00213 /* U+0112 */ 0x0113,
00214 0x00,
00215 /* U+0114 */ 0x0115,
00216 0x00,
00217 /* U+0116 */ 0x0117,
00218 0x00,
00219 /* U+0118 */ 0x0119,
00220 0x00,
00221 /* U+011A */ 0x011B,
00222 0x00,
00223 /* U+011C */ 0x011D,
00224 0x00,
00225 /* U+011E */ 0x011F,
00226 0x00,
00227 /* U+0120 */ 0x0121,
00228 0x00,
00229 /* U+0122 */ 0x0123,
00230 0x00,
00231 /* U+0124 */ 0x0125,
00232 0x00,
00233 /* U+0126 */ 0x0127,
00234 0x00,
00235 /* U+0128 */ 0x0129,
00236 0x00,
00237 /* U+012A */ 0x012B,
00238 0x00,
00239 /* U+012C */ 0x012D,
00240 0x00,
00241 /* U+012E */ 0x012F,
00242 0x00,
00243 /* U+0130 */ 0x0,
00244 0x00,
00245 /* U+0132 */ 0x0133,
00246 0x00,
00247 /* U+0134 */ 0x0135,
00248 0x00,
00249 /* U+0136 */ 0x0137,
00250 0x00,
00251 0x00,
00252 /* U+0139 */ 0x013A,
00253 0x00,
00254 /* U+013B */ 0x013C,
00255 0x00,
00256 /* U+013D */ 0x013E,
00257 0x00,
00258 /* U+013F */ 0x0140,
00259 0x00,
00260 /* U+0141 */ 0x0142,
00261 0x00,
00262 /* U+0143 */ 0x0144,
00263 0x00,
00264 /* U+0145 */ 0x0146,
00265 0x00,
00266 /* U+0147 */ 0x0148,
00267 0x00,
00268 0x00,
00269 /* U+014A */ 0x014B,
00270 0x00,
00271 /* U+014C */ 0x014D,
00272 0x00,
00273 /* U+014E */ 0x014F,
00274 0x00,
00275 /* U+0150 */ 0x0151,
00276 0x00,
00277 /* U+0152 */ 0x0153,
00278 0x00,
00279 /* U+0154 */ 0x0155,
00280 0x00,
00281 /* U+0156 */ 0x0157,
00282 0x00,
00283 /* U+0158 */ 0x0159,
00284 0x00,
00285 /* U+015A */ 0x015B,
00286 0x00,
00287 /* U+015C */ 0x015D,
00288 0x00,
00289 /* U+015E */ 0x015F,
00290 0x00,
00291 /* U+0160 */ 0x0161,
00292 0x00,
00293 /* U+0162 */ 0x0163,
00294 0x00,
00295 /* U+0164 */ 0x0165,
0x00,
0x00 */ "U+0216" */ 0x0217,
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,
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,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
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 /* U+0280 */ 0x0, 00579 /* U+0281 */ 0x0, 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 /* U+028F */ 0x0, 00595 0x00, 00596 /* U+0299 */ 0x0, 00597 0x00, 00598 0x00, 00599 0x00, 00600 0x00, 00601 0x00, 00602 0x00, 00603 0x00, 00604 /* U+0282 */ 0x0, 00605 0x00, 00606 /* U+0283 */ 0x0, 00607 /* U+0284 */ 0x0, 00608 0x00, 00609 0x00, 00610 /* U+0285 */ 0x0, 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 /* U+0286 */ 0x0, 00634 });
00635 00636 static const unsigned short ucs_table_0386[] = {
00637 /* U+0386 */ 0x03AC, 00638 0x00,
00639 /* U+0388 */ 0x03AD, 00640 /* U+0389 */ 0x03AE,
00641 0x00,
00642 0x00,
00643 0x00,
00905 /* U+0492 */ 0x0493,
00906 0x00,
00907 /* U+0494 */ 0x0495,
00908 0x00,
00909 /* U+0496 */ 0x0497,
00910 0x00,
00911 /* U+0498 */ 0x0499,
00912 0x00,
00913 /* U+049A */ 0x049B,
00914 0x00,
00915 /* U+049C */ 0x049D,
00916 0x00,
00917 /* U+049E */ 0x049F,
00918 0x00,
00919 /* U+04A0 */ 0x04A1,
00920 0x00,
00921 /* U+04A2 */ 0x04A3,
00922 0x00,
00923 /* U+04A4 */ 0x04A5,
00924 0x00,
00925 /* U+04A6 */ 0x04A7,
00926 0x00,
00927 /* U+04A8 */ 0x04A9,
00928 0x00,
00929 /* U+04AA */ 0x04AB,
00930 0x00,
00931 /* U+04AC */ 0x04AD,
00932 0x00,
00933 /* U+04AE */ 0x04AF,
00934 0x00,
00935 /* U+04B0 */ 0x04B1,
00936 0x00,
00937 /* U+04B2 */ 0x04B3,
00938 0x00,
00939 /* U+04B4 */ 0x04B5,
00940 0x00,
00941 /* U+04B6 */ 0x04B7,
00942 0x00,
00943 /* U+04B8 */ 0x04B9,
00944 0x00,
00945 /* U+04BA */ 0x04BB,
00946 0x00,
00947 /* U+04BC */ 0x04BD,
00948 0x00,
00949 /* U+04BE */ 0x04BF,
00950 0x00,
00951 0x00,
00952 /* U+04C1 */ 0x04C2,
00953 0x00,
00954 /* U+04C3 */ 0x04C4,
00955 0x00,
00956 0x00,
00957 0x00,
00958 /* U+04C7 */ 0x04CB,
00959 0x00,
00960 0x00,
00961 0x00,
00962 /* U+04CB */ 0x04CC,
00963 0x00,
00964 0x00,
00965 0x00,
00966 0x00,
00967 /* U+04D0 */ 0x04D1,
00968 0x00,
00969 /* U+04D2 */ 0x04D3,
00970 0x00,
00971 /* U+04D4 */ 0x04D5,
00972 0x00,
00973 /* U+04D6 */ 0x04D7,
00974 0x00,
00975 /* U+04D8 */ 0x04D9,
00976 0x00,
00977 /* U+04DA */ 0x04DB,
00978 0x00,
00979 /* U+04DC */ 0x04DD,
00980 0x00,
00981 /* U+04DE */ 0x04DF,
00982 0x00,
00983 /* U+04ED */ 0x04E1,
00984 0x00,
00985 /* U+04E2 */ 0x04E3,
00986 0x00,
00987 /* U+04E4 */ 0x04E5,
00988 0x00,
00989 /* U+04E6 */ 0x04E7,
00990 0x00,
00991 /* U+04E8 */ 0x04E9,
01079 /* U+0540 */ 0x0570,
01080 /* U+0541 */ 0x0571,
01081 /* U+0542 */ 0x0572,
01082 /* U+0543 */ 0x0573,
01083 /* U+0544 */ 0x0574,
01084 /* U+0545 */ 0x0575,
01085 /* U+0546 */ 0x0576,
01086 /* U+0547 */ 0x0577,
01087 /* U+0548 */ 0x0578,
01088 /* U+0549 */ 0x0579,
01089 /* U+054A */ 0x057A,
01090 /* U+054B */ 0x057B,
01091 /* U+054C */ 0x057C,
01092 /* U+054D */ 0x057D,
01093 /* U+054E */ 0x057E,
01094 /* U+054F */ 0x057F,
01095 /* U+0550 */ 0x0580,
01096 /* U+0551 */ 0x0581,
01097 /* U+0552 */ 0x0582,
01098 /* U+0553 */ 0x0583,
01099 /* U+0554 */ 0x0584,
01100 /* U+0555 */ 0x0585,
01101 /* U+0556 */ 0x0586,
01102
01103 static const unsigned short ucs_table_10A0[] = {
01104 /* U+10A0 */ 0x10D0,
01105 /* U+10A1 */ 0x10D1,
01106 /* U+10A2 */ 0x10D2,
01107 /* U+10A3 */ 0x10D3,
01108 /* U+10A4 */ 0x10D4,
01109 /* U+10A5 */ 0x10D5,
01110 /* U+10A6 */ 0x10D6,
01111 /* U+10A7 */ 0x10D7,
01112 /* U+10A8 */ 0x10D8,
01113 /* U+10A9 */ 0x10D9,
01114 /* U+10AA */ 0x10DA,
01115 /* U+10AB */ 0x10DB,
01116 /* U+10AC */ 0x10DC,
01117 /* U+10AD */ 0x10DD,
01118 /* U+10AE */ 0x10DE,
01119 /* U+10AF */ 0x10DF,
01120 /* U+10B0 */ 0x10E0,
01121 /* U+10B1 */ 0x10E1,
01122 /* U+10B2 */ 0x10E2,
01123 /* U+10B3 */ 0x10E3,
01124 /* U+10B4 */ 0x10E4,
01125 /* U+10B5 */ 0x10E5,
01126 /* U+10B6 */ 0x10E6,
01127 /* U+10B7 */ 0x10E7,
01128 /* U+10B8 */ 0x10E8,
01129 /* U+10B9 */ 0x10E9,
01130 /* U+10BA */ 0x10EA,
01131 /* U+10BB */ 0x10EB,
01132 /* U+10BC */ 0x10EC,
01133 /* U+10BD */ 0x10ED,
01134 /* U+10BE */ 0x10EE,
01135 /* U+10BF */ 0x10EF,
01136 /* U+10C0 */ 0x10F0,
01137 /* U+10C1 */ 0x10F1,
01138 /* U+10C2 */ 0x10F2,
01139 /* U+10C3 */ 0x10F3,
01140 /* U+10C4 */ 0x10F4,
01141 /* U+10C5 */ 0x10F5,
01142
01143 static const unsigned short ucs_table_1E00[] = {
01144 /* U+1E00 */ 0x1E01,
01145 /* U+1E01 */ 0x1E02,
01146 /* U+1E02 */ 0x1E03,
01147 /* U+1E03 */ 0x1E04,
01148 /* U+1E04 */ 0x1E05,
01149 /* U+1E05 */ 0x1E06,
01150 /* U+1E06 */ 0x1E07,
01151 /* U+1E07 */ 0x1E08,
01152 /* U+1E08 */ 0x1E09,
01153 /* U+1E09 */ 0x1E0A,
01154 /* U+1E0A */ 0x1E0B,
01155 /* U+1E0B */ 0x1E0C,
01156 /* U+1E0C */ 0x1E0D,
01157 /* U+1E0D */ 0x1E0E,
01158 /* U+1E0E */ 0x1E0F,
01159 /* U+1E0F */ 0x1E10,
01160 /* U+1E10 */ 0x1E11,
01161 /* U+1E11 */ 0x1E12,
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,
01340 /* U+1EC2 */ 0x1EC3,
01341 0x00,
01342 /* U+1EC4 */ 0x1EC5,
01343 0x00,
01344 /* U+1EC6 */ 0x1EC7,
01345 0x00,
01346 /* U+1EC8 */ 0x1EC9,
01347 0x00,
01348 /* U+1EC2 */ 0x1EC3,
01349 0x00,
01350 /* U+1ED0 */ 0x1ED1,
01351 /* U+1ED2 */ 0x1ED3,
01352 /* U+1ED4 */ 0x1ED5,
01353 0x00,
01354 /* U+1ED6 */ 0x1ED7,
01355 0x00,
01356 /* U+1ED8 */ 0x1ED9,
01357 0x00,
01358 /* U+1EE0 */ 0x1EE1,
01359 /* U+1EE2 */ 0x1EE3,
01360 /* U+1EE4 */ 0x1EE5,
01361 /* U+1EE6 */ 0x1EE7,
01362 /* U+1EE8 */ 0x1EE9,
01363 0x00,
01364 /* U+1EEA */ 0x1EEB,
01365 0x00,
01366 /* U+1EEC */ 0x1EED,
01367 0x00,
01368 /* U+1EE8 */ 0x1EE9,
01369 0x00,
01370 /* U+1EEA */ 0x1EEB,
01371 0x00,
01372 /* U+1EE2 */ 0x1EE3,
01373 0x00,
01374 /* U+1EE4 */ 0x1EE5,
01375 0x00,
01376 /* U+1EE6 */ 0x1EE7,
01377 0x00,
01378 /* U+1EE8 */ 0x1EE9,
01379 0x00,
01380 /* U+1EEA */ 0x1EEB,
01381 0x00,
01382 /* U+1EE2 */ 0x1EE3,
01383 0x00,
01384 /* U+1EE4 */ 0x1EE5,
01385 0x00,
01386 /* U+1EE6 */ 0x1EE7,
01387 0x00,
01388 /* U+1EE8 */ 0x1EE9,
01389 0x00,
01390 /* U+1EEA */ 0x1EEB,
01391 0x00,
01392 /* U+1EE2 */ 0x1EE3,
01393 0x00,
01394 /* U+1EE4 */ 0x1EE5,
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,
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,
01481 0x00,
01482 0x00,
01483 0x00,
01484 0x00,
01485 0x00,
01486 0x00,
01487 0x00,
01488 0x00,
01489 0x00,
01490 0x00,
01491 /* U+1F59 */ 0x1F51,
01492 0x00,
01493 /* U+1F5A */ 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 ox00,  
01515 ox00,  
01516 ox00,  
01517 ox00,  
01518 ox00,  
01519 ox00,  
01520 ox00,  
01521 ox00,  
01522 ox00,  
01523 ox00,  
01524 ox00,  
01525 ox00,  
01526 ox00,  
01527 ox00,  
01528 ox00,  
01529 ox00,  
01530 ox00,  
01531 ox00,  
01532 ox00,  
01533 ox00,  
01534 ox00,  
01535 ox00,  
01536 ox00,  
01537 ox00,  
01538 /* U+1F88 */ ox0,  
01539 /* U+1F89 */ ox0,  
01540 /* U+1F8A */ ox0,  
01541 /* U+1F8B */ ox0,  
01542 /* U+1F8C */ ox0,  
01543 /* U+1F8D */ ox0,  
01544 /* U+1F8E */ ox0,  
01545 /* U+1F8F */ ox0,  
01546 ox00,  
01547 ox00,  
01548 ox00,  
01549 ox00,  
01550 ox00,  
01551 ox00,  
01552 ox00,  
01553 ox00,  
01554 /* U+1F98 */ ox0,  
01555 /* U+1F99 */ ox0,  
01556 /* U+1F9A */ ox0,  
01557 /* U+1F9B */ ox0,  
01558 /* U+1F9C */ ox0,  
01559 /* U+1F9D */ ox0,  
01560 /* U+1F9E */ ox0,  
01561 /* U+1F9F */ ox0,  
01562 ox00,  
01563 ox00,  
01564 ox00,  
01565 ox00,  
01566 ox00,  
01567 ox00,  
01568 ox00,  
01569 ox00,  
01570 /* U+1FA8 */ ox0,  
01571 /* U+1FA9 */ ox0,  
01572 /* U+1FAA */ ox0,  
01573 /* U+1FAB */ ox0,  
01574 /* U+1FAC */ ox0,  
01575 /* U+1FAD */ ox0,  
01576 /* U+1FAE */ ox0,  
01577 /* U+1FAF */ ox0,  
01578 ox00,  
01579 ox00,  
01580 ox00,  
01581 ox00,  
01582 ox00,  
01583 ox00,  
01584 ox00,  
01585 ox00,  
01586 /* U+1F88 */ 0x1FB0,  
01587 /* U+1F89 */ 0x1F81,  
01588 /* U+1F8A */ 0x1F82,  
01589 /* U+1F8B */ 0x1F83,  
01590 /* U+1F8C */ 0x0,  
01591 ox00,  
01592 ox00,  
01593 ox00,  
01594 ox00,  
01595 ox00,  
01596 ox00,  
01597 ox00,  
01598 ox00,  
01599 ox00,  
01600 ox00,
01601 0x00,
01602 /* U+1FC8 */ 0x1F72,
01603 /* U+1FC9 */ 0x1F73,
01604 /* U+1FCA */ 0x1F74,
01605 /* U+1FCB */ 0x1F75,
01606 /* U+1FCC */ 0x00,
01607 0x00,
01608 0x00,
01609 0x00,
01610 0x00,
01611 0x00,
01612 0x00,
01613 0x00,
01614 0x00,
01615 0x00,
01616 0x00,
01617 0x00,
01618 /* U+1FD8 */ 0x1FD0,
01619 /* U+1FD9 */ 0x1FD1,
01620 /* U+1FDB */ 0x1F77,
01621 /* U+1FD8 */ 0x1FD0,
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+1FEC */ 0x1FE5,
01635 /* U+1FEA */ 0x1F7A,
01636 /* U+1FE9 */ 0x1FE1,
01637 /* U+1FE8 */ 0x1FE0,
01638 /* U+1FEB */ 0x1F7B,
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 3j
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+210B */ 0x00,
01668 /* U+210C */ 0x00,
01669 /* U+210D */ 0x00,
01670 0x00,
01671 0x00,
01672 /* U+2110 */ 0x00,
01673 /* U+2111 */ 0x00,
01674 /* U+2112 */ 0x2113,
01675 0x00,
01676 0x00,
01677 /* U+2115 */ 0x00,
01678 0x00,
01679 0x00,
01680 /* U+2118 */ 0x00,
01681 /* U+2119 */ 0x00,
01682 /* U+211A */ 0x00,
01683 /* U+211B */ 0x00,
01684 /* U+211C */ 0x00,
01685 /* U+211D */ 0x00,
01686 0x00,
01687 0x00,
01688 0x00,
01689 0x00,
01690 0x00,
01691 0x00,
01688 0x00, 01689 0x00, 01690 0x00, 01691 0x00, 01692 /* U+2124 */ 0x0, 01693 0x00, 01694 0x00, 01695 0x00, 01696 /* U+2128 */ 0x0, 01697 0x00, 01698 0x00, 01699 0x00, 01700 /* U+212C */ 0x0, 01701 /* U+212D */ 0x0, 01702 0x00, 01703 0x00, 01704 /* U+2130 */ 0x0212F, 01705 /* U+2131 */ 0x0, 01706 /* U+2132 */ 0x0, 01707 /* U+2133 */ 0x0, 01708 ;
01709
01710 static const unsigned short ucs_table_24B6[] = {
01711 /* U+24B6 */ 0x24D0,
01712 /* U+24B7 */ 0x24D1,
01713 /* U+24B8 */ 0x24D2,
01714 /* U+24B9 */ 0x24D3,
01715 /* U+24BA */ 0x24D4,
01716 /* U+24BB */ 0x24D5,
01717 /* U+24BC */ 0x24D6,
01718 /* U+24BD */ 0x24D7,
01719 /* U+24BE */ 0x24D8,
01720 /* U+24BF */ 0x24D9,
01721 /* U+24C0 */ 0x24DA,
01722 /* U+24C1 */ 0x24DB,
01723 /* U+24C2 */ 0x24DC,
01724 /* U+24C3 */ 0x24DD,
01725 /* U+24C4 */ 0x24DE,
01726 /* U+24C5 */ 0x24DF,
01727 /* U+24C6 */ 0x24E0,
01728 /* U+24C7 */ 0x24E1,
01729 /* U+24C8 */ 0x24E2,
01730 /* U+24C9 */ 0x24E3,
01731 /* U+24CA */ 0x24E4,
01732 /* U+24CB */ 0x24E5,
01733 /* U+24CC */ 0x24E6,
01734 /* U+24CD */ 0x24E7,
01735 /* U+24CE */ 0x24E8,
01736 /* U+24CF */ 0x24E9,
01737};
01738
01739 static const unsigned short ucs_table_33CE[] = {
01740 /* U+33CE */ 0x0};
01741
01742
01743 static const unsigned short ucs_table_FF21[] = {
01744 /* U+FF21 */ 0xFF41,
01745 /* U+FF22 */ 0xFF42,
01746 /* U+FF23 */ 0xFF43,
01747 /* U+FF24 */ 0xFF44,
01748 /* U+FF25 */ 0xFF45,
01749 /* U+FF26 */ 0xFF46,
01750 /* U+FF27 */ 0xFF47,
01751 /* U+FF28 */ 0xFF48,
01752 /* U+FF29 */ 0xFF49,
01753 /* U+FF2A */ 0xFF4A,
01754 /* U+FF2B */ 0xFF4B,
01755 /* U+FF2C */ 0xFF4C,
01756 /* U+FF2D */ 0xFF4D,
01757 /* U+FF2E */ 0xFF4E,
01758 /* U+FF2F */ 0xFF4F,
01759 /* U+FF30 */ 0xFF50,
01760 /* U+FF31 */ 0xFF51,
01761 /* U+FF32 */ 0xFF52,
01762 /* U+FF33 */ 0xFF53,
01763 /* U+FF34 */ 0xFF54,
01764 /* U+FF35 */ 0xFF55,
01765 /* U+FF36 */ 0xFF56,
01766 /* U+FF37 */ 0xFF57,
01767 /* U+FF38 */ 0xFF58,
01768 /* U+FF39 */ 0xFF59,
01769 /* U+FF3A */ 0xFF5A,
01770};
10.203  dingbats_.h

/* dingbats */

static const char unicode_to_dingbats_1b_0020[] = {
    0x20, 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, 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, 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,
    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, 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, 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, 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, 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, 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, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 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 0x00,
0132 /* U+00A0 */ 0x20,
0133 */
0134 0135 static const char unicode_to_dingbats_1b_2192[] = {
0136 /* U+2192 */ (char)0xD5,
0137 0x00,
0138 /* U+2194 */ (char)0xD6,
0139 /* U+2195 */ (char)0xD7,
0140 */
0141 0142 static const char unicode_to_dingbats_1b_2460[] = {
0143 /* U+2460 */ (char)0xAC,
0144 /* U+2461 */ (char)0xAD,
0145 /* U+2462 */ (char)0xAE,
0146 /* U+2463 */ (char)0xAF,
0147 /* U+2464 */ (char)0xB0,
0148 /* U+2465 */ (char)0xB1,
0149 /* U+2466 */ (char)0xB2,
0150 /* U+2467 */ (char)0xB3,
0151 /* U+2468 */ (char)0xB4,
0152 /* U+2469 */ (char)0xB5,
0153 */
0154 0155 static const char unicode_to_dingbats_1b_25A0[] = {
0156 /* U+25A0 */ 0x6E,
0157 0x00,
0158 0x00,
0159 0x00,
0160 0x00,
0161 0x00,
0162 0x00,
0163 0x00,
0164 0x00,
0165 0x00,
0166 0x00,
0167 0x00,
0168 0x00,
0169 0x00,
0170 0x00,
0171 0x00,
0172 0x00,
0x00,
0x00, /* U+2660 */ (char)0xAB,
0x00,
0x00, /* U+2663 */ (char)0xA8,
0x00,
0x00, /* U+2665 */ (char)0xAA,
0x00, /* U+2666 */ (char)0xA9,
0x00 });

static const char unicode_to_dingbats_1b_2701[] = {
0x00, /* U+2701 */ 0x21,
0x00, /* U+2702 */ 0x22,
0x00, /* U+2703 */ 0x23,
0x00, /* U+2704 */ 0x24,
0x00, /* U+2706 */ 0x26,
0x00, /* U+2707 */ 0x27,
0x00, /* U+2708 */ 0x28,
0x00, /* U+2709 */ 0x29,
0x00, /* U+270C */ 0x2C,
0x00, /* U+270D */ 0x2D,
0x00, /* U+270E */ 0x2E,
0x00, /* U+270F */ 0x2F,
0x00, /* U+2710 */ 0x30,
0x00, /* U+2711 */ 0x31,
0x00, /* U+2712 */ 0x32,
0x00, /* U+2713 */ 0x33,
0x00, /* U+2714 */ 0x34,
0x00, /* U+2715 */ 0x35,
0x00, /* U+2716 */ 0x36,
0x00, /* U+2717 */ 0x37,
0x00, /* U+2718 */ 0x38,
0x00, /* U+2719 */ 0x39,
0x00, /* U+271A */ 0x3A,
0x00, /* U+271B */ 0x3B,
0x00, /* U+271C */ 0x3C,
0x00, /* U+271D */ 0x3D,
0x00, /* U+271E */ 0x3E,
0x00, /* U+271F */ 0x3F,
0x00, /* U+2720 */ 0x40,
0x00, /* U+2721 */ 0x41,
0x00, /* U+2722 */ 0x42,
0x00, /* U+2723 */ 0x43,
0x00, /* U+2724 */ 0x44,
0x00, /* U+2725 */ 0x45,
0x00, /* U+2726 */ 0x46,
0x00, /* U+2727 */ 0x47,
0x00, /* U+2729 */ 0x49,
0x00, /* U+272A */ 0x4A,
0x00, /* U+272B */ 0x4B,
0x00, /* U+272C */ 0x4C,
0x00, /* U+272D */ 0x4D,
0x00, /* U+272E */ 0x4E,
0x00, /* U+272F */ 0x4F,
0x00, /* U+2730 */ 0x50,
0x00, /* U+2731 */ 0x51,
0x00, /* U+2732 */ 0x52,
0x00, /* U+2733 */ 0x53,
0x00, /* U+2734 */ 0x54,
0x00, /* U+2735 */ 0x55,
0x00, /* U+2736 */ 0x56,
0x00, /* U+2737 */ 0x57,
0x00, /* U+2738 */ 0x58,
0x00, /* U+2739 */ 0x59,
0x00, /* U+273A */ 0x5A,
0x00, /* U+273B */ 0x5B,
0x00, /* U+273C */ 0x5C,
0x00, /* U+273D */ 0x5D,
0x00, /* U+273E */ 0x5E,
0x00, /* U+273F */ 0x5F,
0x00, /* U+2740 */ 0x60,
0x00, /* U+2741 */ 0x61,
0x00, /* U+2742 */ 0x62,
0x00, /* U+2743 */ 0x63,
0x00, /* U+2744 */ 0x64,
0x00, /* U+2745 */ 0x65,
0x00, /* U+2746 */ 0x66,
0x00, /* U+2747 */ 0x67,
0x00, /* U+2748 */ 0x68,
0x00, /* U+2749 */ 0x69,
0x00, /* U+274A */ 0x6A,
0x00, /* U+274B */ 0x6B,
0x00, /* U+274C */ 0x6C,
00434 /* U+274D */ 0x6D,
00435 /* U+274F */ 0x6F,
00436 /* U+2750 */ 0x70,
00437 /* U+2751 */ 0x71,
00438 /* U+2752 */ 0x72,
00439 /* U+2756 */ 0x76,
00440 /* U+2758 */ 0x78,
00441 /* U+2759 */ 0x79,
00442 /* U+275A */ 0x7A,
00443 /* U+275B */ 0x7B,
00444 /* U+275C */ 0x7C,
00445 /* U+275D */ 0x7D,
00446 /* U+275E */ 0x7E,
00447 /* U+2761 */ (char)\xA1,
00448 /* U+2762 */ (char)\xA2,
00449 /* U+2763 */ (char)\xA3,
00450 /* U+2764 */ (char)\xA4,
00451 /* U+2765 */ (char)\xA5,
00452 /* U+2766 */ (char)\xA6,
00453 /* U+2767 */ (char)\xA7,
00021 / * U+27A4 */ (char)0xE4,
00022 / * U+27A5 */ (char)0xE5,
00023 / * U+27A6 */ (char)0xE6,
00024 / * U+27A7 */ (char)0xE7,
00025 / * U+27A8 */ (char)0xE8,
00026 / * U+27A9 */ (char)0xE9,
00027 / * U+27AA */ (char)0xEA,
00028 / * U+27AB */ (char)0xEB,
00029 / * U+27AC */ (char)0xEC,
00030 / * U+27AD */ (char)0xED,
00031 / * U+27AE */ (char)0xEE,
00032 / * U+27AF */ (char)0xEF,
00033 0x00,
00034 / * U+27B1 */ (char)0xF1,
00035 / * U+27B2 */ (char)0xF2,
00036 / * U+27B3 */ (char)0xF3,
00037 / * U+27B4 */ (char)0xF4,
00038 / * U+27B5 */ (char)0xF5,
00039 / * U+27B6 */ (char)0xF6,
00040 / * U+27B7 */ (char)0xF7,
00041 / * U+27B8 */ (char)0xF8,
00042 / * U+27B9 */ (char)0xF9,
00043 / * U+27BA */ (char)0xFA,
00044 / * U+27BB */ (char)0xFB,
00045 / * U+27BC */ (char)0xFC,
00046 / * U+27BD */ (char)0xFD,
00047 / * U+27BE */ (char)0xFE,
00048 */};
00049
00050 static const char unicode_to_dingbats_1b_F8D7[] = {
00051 /* U+F8D7 */ (char)0x80,
00052 /* U+F8D8 */ (char)0x81,
00053 /* U+F8D9 */ (char)0x82,
00054 /* U+F8DA */ (char)0x83,
00055 /* U+F8DB */ (char)0x84,
00056 /* U+F8DC */ (char)0x85,
00057 /* U+F8DD */ (char)0x86,
00058 /* U+F8DE */ (char)0x87,
00059 /* U+F8DF */ (char)0x88,
00060 /* U+F8E0 */ (char)0x89,
00061 /* U+F8E1 */ (char)0x8A,
00062 /* U+F8E2 */ (char)0x8B,
00063 /* U+F8E3 */ (char)0x8C,
00064 /* U+F8E4 */ (char)0x8D,
00065 */};

10.204 spacing.h

00001 /* spacing */
00002
00003 static const unsigned short ucs_table_0300[] = {
00004 /* U+0300 */ 0x0060,
00005 /* U+0301 */ 0x00B4,
00006 /* U+0302 */ 0x005E,
00007 /* U+0303 */ 0x203E,
00008 /* U+0304 */ 0x00AF,
00009 /* U+0305 */ 0x02D8,
00010 /* U+0306 */ 0x02D9,
00011 /* U+0307 */ 0x00A8,
00012 /* U+0308 */ 0x0309,
00013 /* U+0309 */ 0x031F,
00014 /* U+030A */ 0x1FBD,
00015 /* U+030B */ 0x1FFE,
00016 /* U+030C */ 0x0315,
00017 /* U+030D */ 0x0316,
00018 /* U+030E */ 0x0317,
00019 /* U+030F */ 0x0318,
00020 /* U+0310 */ 0x0319,
00021 /* U+0311 */ 0x031A,
00022 /* U+0312 */ 0x031B,
00023 /* U+0313 */ 0x031C,
00024 /* U+0314 */ 0x031D,
00025 /* U+0315 */ 0x031E,
00026 /* U+0316 */ 0x031F,
00027 /* U+0317 */ 0x0320,
00028 /* U+0318 */ 0x0321,
00038 /* U+0322 */ 0x0322,
00039 /* U+0323 */ 0x0323,
00040 /* U+0324 */ 0x0324,
00041 /* U+0325 */ 0x0325,
00042 /* U+0326 */ 0x0326,
00043 /* U+0327 */ 0x00B8,
00044 /* U+0328 */ 0x02DB,
00045 /* U+0329 */ 0x0329,
00046 /* U+032A */ 0x032A,
00047 /* U+032B */ 0x032B,
00048 /* U+032C */ 0x032C,
00049 /* U+032D */ 0x032D,
00050 /* U+032E */ 0x032E,
00051 /* U+032F */ 0x032F,
00052 /* U+0330 */ 0x0330,
00053 /* U+0331 */ 0x0331,
00054 /* U+0332 */ 0x005F,
00055 /* U+0333 */ 0x2017,
00056 /* U+0334 */ 0x0334,
00057 /* U+0335 */ 0x0335,
00058 /* U+0336 */ 0x0336,
00059 /* U+0337 */ 0x0337,
00060 /* U+0338 */ 0x0338,
00061 /* U+0339 */ 0x0339,
00062 /* U+033A */ 0x033A,
00063 /* U+033B */ 0x033B,
00064 /* U+033C */ 0x033C,
00065 /* U+033D */ 0x033D,
00066 /* U+033E */ 0x033E,
00067 /* U+033F */ 0x033F,
00068 /* U+0340 */ 0x0340,
00069 /* U+0341 */ 0x0341,
00070 /* U+0342 */ 0x1FC0,
00071 /* U+0343 */ 0x0343,
00072 /* U+0344 */ 0x0344,
00073 /* U+0345 */ 0x037A,
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,
0100 /* U+0360 */ 0x0360,
0101 /* U+0361 */ 0x0361,
0102 0x00,
0103 0x00,
0104 static const unsigned short ucs_table_0483[] = { 0105 /* U+0483 */ 0x0483,
0106 /* U+0484 */ 0x0484,
0107 /* U+0485 */ 0x0485,
0108 /* U+0486 */ 0x0486,
0109 0x00,
0110 0x00,
0111 static const unsigned short ucs_table_0591[] = { 0112 /* U+0591 */ 0x0591,
0113 /* U+0592 */ 0x0592,
0114 /* U+0593 */ 0x0593,
0115 /* U+0594 */ 0x0594,
0116 /* U+0595 */ 0x0595,
0117 /* U+0596 */ 0x0596,
0118 /* U+0597 */ 0x0597,
0119 /* U+0598 */ 0x0598,
0120 /* U+0599 */ 0x0599,
0121 /* U+059A */ 0x059A,
0122 /* U+059B */ 0x059B,
0123 /* U+059C */ 0x059C,
0124 /* U+059D */ 0x059D,
00125 /* U+059E */ 0x059E,
00126 /* U+059F */ 0x059F,
00127 /* U+05A0 */ 0x05A0,
00128 /* U+05A1 */ 0x05A1,
00129 0x00,
00130 /* U+05A3 */ 0x05A3,
00131 /* U+05A4 */ 0x05A4,
00132 /* U+05A5 */ 0x05A5,
00133 /* U+05A6 */ 0x05A6,
00134 /* U+05A7 */ 0x05A7,
00135 /* U+05A8 */ 0x05A8,
00136 /* U+05A9 */ 0x05A9,
00137 /* U+05AA */ 0x05AA,
00138 /* U+05AB */ 0x05AB,
00139 /* U+05AC */ 0x05AC,
00140 /* U+05AD */ 0x05AD,
00141 /* U+05AE */ 0x05AE,
00142 /* U+05AF */ 0x05AF,
00143 /* U+05B0 */ 0x05B0,
00144 /* U+05B1 */ 0x05B1,
00145 /* U+05B2 */ 0x05B2,
00146 /* U+05B3 */ 0x05B3,
00147 /* U+05B4 */ 0x05B4,
00148 /* U+05B5 */ 0x05B5,
00149 /* U+05B6 */ 0x05B6,
00150 /* U+05B7 */ 0x05B7,
00151 /* U+05B8 */ 0x05B8,
00152 /* U+05B9 */ 0x05B9,
00153 0x00,
00154 /* U+05BB */ 0x05BB,
00155 /* U+05BC */ 0x05BC,
00156 0x00,
00157 0x00,
00158 /* U+05BD */ 0x05BD,
00159 0x00,
00160 /* U+05C1 */ 0x05C1,
00161 /* U+05C2 */ 0x05C2,
00162 0x00,
00163 /* U+05C4 */ 0x05C4,
00164 });
00165
00166 static const unsigned short ucs_table_064B[] = {
00167 /* U+064B */ 0xFE70,
00168 /* U+064C */ 0xFE72,
00169 /* U+064D */ 0xFE74,
00170 /* U+064E */ 0xFE76,
00171 /* U+064F */ 0xFE78,
00172 /* 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,
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 00332 static const unsigned short ucs_table_0901[] = {
00333 /* U+0901 */ 0x0901,
00334 /* U+0902 */ 0x0902,
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 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,
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,
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 0x00,
00777 0x00,
00778 0x00,
00779 0x00,
00780 0x00,
00781 0x00,
00782 0x00,
00783 0x00,
00784 0x00,
00785 0x00,
00786 0x00,
00787 0x00,
00788 0x00,
00789 0x00,
00790 0x00,
00791 0x00,
00792 0x00,
00793 0x00,
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,
00869 0x00,
00870 0x00,
00871 0x00,
00872 0x00,
00873 0x00,
00874 0x00,
00875 0x00,
00876 0x00,
00877 0x00,
00878 0x00,
00879 0x00,
00880 0x00,
00881 0x00,
00882 0x00,
00883 0x00,
00884 0x00,
00885 0x00,
00886 0x00,
00887 0x00,
00888 0x00,
00889 0x00,
00890 0x00,
00891 0x00,
00892 0x00,
00893 0x00,
00894 0x00,
00895 0x00,
00896 0x00,
00897 0x00,
00898 0x00,
00899 0x00,
00900 0x00,
00901 0x00,
00902 0x00,
00903 0x00,
00904 /* U+0B3C */ 0x0B3C,
00905 0x00,
00906 0x00,
00907 /* U+0B3F */ 0x0B3F,
0x00,
0x00 /* U+0B41 */ 0x0B41,
0x00 /* U+0B42 */ 0x0B42,
0x00 /* U+0B43 */ 0x0B43,
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 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,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 0x00,
0x00 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 /* U+0BCD */ 0x0BCD,
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 /* 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,
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 /* U+0CBF */ 0x0CBF,
01292 0x00,
01293 0x00,
01294 0x00,
01295 0x00,
01296 0x00,
01297 0x00,
01298 /* U+0CC6 */ 0x0CC6,
01299 0x00,
01300 0x00,
01301 0x00,
01302 0x00,
01303 0x00,
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,
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,
static const unsigned short ucs_table_0E31[] = {
    /* U+0E31 */ 0x0E31,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    /* U+0E34 */ 0x0E34,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    /* U+0E37 */ 0x0E37,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    /* U+0E3A */ 0x0E3A,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    /* U+0E47 */ 0x0E47,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    /* U+0E48 */ 0x0E48,
    0x00,
    0x00,
    0x00,
    0x00,
    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 0x00,
01566 0x00,
01567 0x00,
01568 0x00,
01569 0x00,
01570 0x00,
01571 0x00,
01572 0x00,
01573 0x00,
01574 0x00,
01575 0x00,
01576 0x00,
01577 0x00,
01578 0x00,
01579 0x00,
01580 0x00,
01581 0x00,
01582 0x00,
01583 0x00,
01584 0x00,
01585 0x00,
01586 0x00,
01587 0x00,
01588 0x00,
01589 0x00,
01590 0x00,
01591 0x00,
01592 0x00,
01593 0x00,
01594 0x00,
01595 0x00,
01596 0x00,
Generated by Doxygen
01778 /* U+0F86 */ 0x0F86,
01779 /* U+0F87 */ 0x0F87,
01780 0x00,
01781 0x00,
01782 0x00,
01783 0x00,
01784 0x00,
01785 0x00,
01786 0x00,
01787 0x00,
01788 /* U+0F90 */ 0x0F90,
01789 /* U+0F91 */ 0x0F91,
01790 /* U+0F92 */ 0x0F92,
01791 /* U+0F93 */ 0x0F93,
01792 /* U+0F94 */ 0x0F94,
01793 /* U+0F95 */ 0x0F95,
01794 0x00,
01795 /* U+0F97 */ 0x0F97,
01796 0x00,
01797 /* U+0F99 */ 0x0F99,
01798 /* U+0F9A */ 0x0F9A,
01799 /* U+0F9B */ 0x0F9B,
01800 /* U+0F9C */ 0x0F9C,
01801 /* U+0F9D */ 0x0F9D,
01802 /* U+0F9E */ 0x0F9E,
01803 /* U+0F9F */ 0x0F9F,
01804 /* U+0FA0 */ 0x0FA0,
01805 /* U+0FA1 */ 0x0FA1,
01806 /* U+0FA2 */ 0x0FA2,
01807 /* U+0FA3 */ 0x0FA3,
01808 /* U+0FA4 */ 0x0FA4,
01809 /* U+0FA5 */ 0x0FA5,
01810 /* U+0FA6 */ 0x0FA6,
01811 /* U+0FA7 */ 0x0FA7,
01812 /* U+0FA8 */ 0x0FA8,
01813 /* U+0FA9 */ 0x0FA9,
01814 /* U+0FAA */ 0x0FAA,
01815 /* U+0FAB */ 0x0FAB,
01816 /* U+0FAC */ 0x0FAC,
01817 /* U+0FAD */ 0x0FAD,
01818 0x00,
01819 0x00,
01820 0x00,
01821 /* U+0FBB */ 0x0FBB,
01822 /* U+0FBC */ 0x0FBC,
01823 /* U+0FBD */ 0x0FBD,
01824 /* U+0FBE */ 0x0FBE,
01825 /* U+0BF0 */ 0x0BF0,
01826 /* U+0BF1 */ 0x0BF1,
01827 /* U+0BF2 */ 0x0BF2,
01828 0x00,
01829 /* U+0FB8 */ 0x0FB8,
01830 0x00,
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+20E4 */ 0x20E4,
01851 0x00,
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 0x00,
01860 0x00,
01861 0x00,
01862 0x00,
01863 0x00,
01864 0x00,
Generated by Doxygen
# 10.205 symbol_.h

```c
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
01969 static const unsigned short ucs_table_FB1E[] = {
01970 /* U+FB1E */ 0xFB1E,
01971 }
01972
01973 static const unsigned short ucs_table_FE20[] = {
01974 /* U+FE20 */ 0xFE20,
01975 /* U+FE21 */ 0xFE21,
01976 /* U+FE22 */ 0xFE22,
01977 /* U+FE23 */ 0xFE23,
01978 
```

```c
00001 /* symbol */
00002
00003 static const char unicode_to_symbol_1b_0020[] = {
00004 /* U+0020 */ 0x20,
00005 /* U+0021 */ 0x21,
00006 0x00,
00007 /* U+0023 */ 0x23,
00008 0x00,
00009 /* U+0025 */ 0x25,
00010 /* U+0026 */ 0x26,
00011 0x00,
00012 /* U+0028 */ 0x28,
00013 /* U+0029 */ 0x29,
00014 0x00,
00015 /* U+002B */ 0x2B,
00016 /* 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,
```

Generated by Doxygen
0x00,
0144  /* U+00AC */ (char)0xD8,
0145 0x00,
0146 0x00,
0147 0x00,
0148  /* U+00B0 */ (char)0xB0,
0149  /* U+00B1 */ (char)0xB1,
0150 0x00,
0151 0x00,
0152 0x00,
0153  /* U+00B5 */ 0x6D,
0154 0x00,
0155 0x00,
0156 0x00,
0157 0x00,
0158 0x00,
0159 0x00,
0160 0x00,
0161 0x00,
0162 0x00,
0163 0x00,
0164 0x00,
0165 0x00,
0166 0x00,
0167 0x00,
0168 0x00,
0169 0x00,
0170 0x00,
0171 0x00,
0172 0x00,
0173 0x00,
0174 0x00,
0175 0x00,
0176 0x00,
0177 0x00,
0178 0x00,
0179 0x00,
0180 0x00,
0181 0x00,
0182 0x00,
0183 0x00,
0184 0x00,
0185 0x00,
0186 0x00,
0187  /* U+00D7 */ (char)0xB4,
0188 0x00,
0189 0x00,
0190 0x00,
0191 0x00,
0192 0x00,
0193 0x00,
0194 0x00,
0195 0x00,
0196 0x00,
0197 0x00,
0198 0x00,
0199 0x00,
0200 0x00,
0201 0x00,
0202 0x00,
0203 0x00,
0204 0x00,
0205 0x00,
0206 0x00,
0207 0x00,
0208 0x00,
0209 0x00,
0210 0x00,
0211 0x00,
0212 0x00,
0213 0x00,
0214 0x00,
0215 0x00,
0216 0x00,
0217 0x00,
0218 0x00,
0219  /* U+00F7 */ (char)0xB8,
0220 );
0221
0222 static const char unicode_to_symbol_1b_0192[] = {
0223  /* U+0192 */ (char)0xA6,
0224 );
0225
0226 static const char unicode_to_symbol_1b_0391[] = {
0227  /* U+0391 */ 0x41,
0228  /* U+0392 */ 0x42,
0229  /* U+0393 */ 0x47,
00317 /* U+2033 */ (char)0xb2,
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 /* U+2044 */ (char)0xa4,
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 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,
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,
0x00,
0x00, /* U+2190 */ (char)0xAC,
0x00, /* U+2191 */ (char)0xAD,
0x00, /* U+2192 */ (char)0xAE,
0x00, /* U+2193 */ (char)0xAF,
0x00, /* U+2194 */ (char)0xAB,
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,
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,
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,
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,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
0x00,
00839 0x00,
00840 0x00,
00841 0x00,
00842 0x00,
00843 0x00,
00844 0x00,
00845 0x00,
00846 0x00,
00847 /* U+2245 */ 0x40,
00848 0x00,
00849 0x00,
00850 /* U+2248 */ (char)0xBB,
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,
00869 0x00,
00870 0x00,
00871 0x00,
00872 0x00,
00873 0x00,
00874 /* U+2260 */ (char)0xB9,
00875 /* U+2261 */ (char)0xBA,
00876 0x00,
00877 0x00,
00878 /* U+2264 */ (char)0xA3,
00879 /* U+2265 */ (char)0xB3,
00880 0x00,
00881 0x00,
00882 0x00,
00883 0x00,
00884 0x00,
00885 0x00,
00886 0x00,
00887 0x00,
00888 0x00,
00889 0x00,
00890 0x00,
00891 0x00,
00892 0x00,
00893 0x00,
00894 0x00,
00895 0x00,
00896 0x00,
00897 0x00,
00898 0x00,
00899 0x00,
00900 0x00,
00901 0x00,
00902 0x00,
00903 0x00,
00904 0x00,
00905 0x00,
00906 0x00,
00907 0x00,
00908 /* U+2282 */ (char)0xCC,
00909 /* U+2283 */ (char)0xCD,
00910 /* U+2284 */ (char)0xCA,
00911 0x00,
00912 /* U+2286 */ (char)0xCD,
00913 /* U+2287 */ (char)0xCA,
00914 0x00,
00915 0x00,
00916 0x00,
00917 0x00,
00918 0x00,
00919 0x00,
00920 0x00,
00921 0x00,
00922 0x00,
00923 0x00,
00924 0x00,
00925 0x00,
00926 0x00,
00927 /* U+2295 */ (char)0xC5,
00928 0x00,
00929 /* U+2297 */ (char)0xC4,
00930 0x00,
00931 0x00,
00932 0x00,
00933 0x00,
00934 0x00,
00935 0x00,
00936 0x00,
00937 0x00,
00938 0x00,
00939 0x00,
00940 0x00,
00941 0x00,
00942 0x00,
00943 /* U+22A5 */ 0x5E,
00944 0x00,
00945 0x00,
00946 0x00,
00947 0x00,
00948 0x00,
00949 0x00,
00950 0x00,
00951 0x00,
00952 0x00,
00953 0x00,
00954 0x00,
00955 0x00,
00956 0x00,
00957 0x00,
00958 0x00,
00959 0x00,
00960 0x00,
00961 0x00,
00962 0x00,
00963 0x00,
00964 0x00,
00965 0x00,
00966 0x00,
00967 0x00,
00968 0x00,
00969 0x00,
00970 0x00,
00971 0x00,
00972 0x00,
00973 0x00,
00974 0x00,
00975 /* U+22C5 */ (char)0xD7,
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,
01061 0x00,
01062 0x00,
01063 0x00,
01064 0x00,
01065 0x00,
01066 /* U+2320 */ (char)0xF3,
01067 /* U+2321 */ (char)0xF5,
01068 0x00,
01069 0x00,
01070 0x00,
01071 0x00,
01072 0x00,
01073 0x00,
01074 0x00,
01075 /* U+2329 */ (char)0xE1,
01076 /* U+232A */ (char)0xF1,
01077 });
01078 01079 static const char unicode_to_symbol_1b_25CA[] = {
01080 /* U+25CA */ (char)0xE0,
01081 };
01082 01083 static const char unicode_to_symbol_1b_2660[] = {
01084 /* U+2660 */ (char)0xAA,
01085 0x00,
01086 0x00,
01087 /* U+2663 */ (char)0xA7,
01088 0x00,
01089 /* U+2665 */ (char)0xA9,
01090 /* U+2666 */ (char)0xA8,
01091 };;
01092 01093 static const char unicode_to_symbol_1b_F6D9[] = {
01094 /* U+F6D9 */ (char)0xD3,
01095 /* U+F6DA */ (char)0xD2,
01096 /* U+F6DB */ (char)0xD4,
01097 });
01098 01099 static const char unicode_to_symbol_1b_F8E5[] = {
01100 0x00,
1944

01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126

File Documentation

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
};

U+F8E5
U+F8E6
U+F8E7
U+F8E8
U+F8E9
U+F8EA
U+F8EB
U+F8EC
U+F8ED
U+F8EE
U+F8EF
U+F8F0
U+F8F1
U+F8F2
U+F8F3
U+F8F4
U+F8F5
U+F8F6
U+F8F7
U+F8F8
U+F8F9
U+F8FA
U+F8FB
U+F8FC
U+F8FD
U+F8FE

10.206
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

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

0x60,
(char)0xBD,
(char)0xBE,
(char)0xE2,
(char)0xE3,
(char)0xE4,
(char)0xE6,
(char)0xE7,
(char)0xE8,
(char)0xE9,
(char)0xEA,
(char)0xEB,
(char)0xEC,
(char)0xED,
(char)0xEE,
(char)0xEF,
(char)0xF4,
(char)0xF6,
(char)0xF7,
(char)0xF8,
(char)0xF9,
(char)0xFA,
(char)0xFB,
(char)0xFC,
(char)0xFD,
(char)0xFE,

imKStoUCS.c

#include "Xlibint.h"
#include "Ximint.h"
static unsigned short const
0x0104, 0x02d8,
0x0000, 0x0160, 0x015e,
0x0000, 0x0105, 0x02db,
0x0000, 0x0161, 0x015f,
0x0154, 0x0000, 0x0000,
0x010c, 0x0000, 0x0118,
0x0110, 0x0143, 0x0147,
0x0158, 0x016e, 0x0000,
0x0155, 0x0000, 0x0000,
0x010d, 0x0000, 0x0119,
0x0111, 0x0144, 0x0148,
0x0159, 0x016f, 0x0000,
};

keysym_to_unicode_1a1_1ff[] = {
0x0141, 0x0000, 0x013d, 0x015a,
0x0164, 0x0179, 0x0000, 0x017d,
0x0142, 0x0000, 0x013e, 0x015b,
0x0165, 0x017a, 0x02dd, 0x017e,
0x0102, 0x0000, 0x0139, 0x0106,
0x0000, 0x011a, 0x0000, 0x0000,
0x0000, 0x0000, 0x0150, 0x0000,
0x0170, 0x0000, 0x0000, 0x0162,
0x0103, 0x0000, 0x013a, 0x0107,
0x0000, 0x011b, 0x0000, 0x0000,
0x0000, 0x0000, 0x0151, 0x0000,
0x0171, 0x0000, 0x0000, 0x0163,

static unsigned short const
0x0126, 0x0000,
0x0000, 0x0130, 0x0000,
0x0000, 0x0127, 0x0000,
0x0000, 0x0131, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x011c, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x011d, 0x0000, 0x0000,
};

0x0000,
0x017b,
0x02c7,
0x017c,
0x0000,
0x010e,
0x0000,
0x0000,
0x0000,
0x010f,
0x0000,
0x02d9

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x01a0-0x01a7
0x01a8-0x01af
0x01b0-0x01b7
0x01b8-0x01bf
0x01c0-0x01c7
0x01c8-0x01cf
0x01d0-0x01d7
0x01d8-0x01df
0x01e0-0x01e7
0x01e8-0x01ef
0x01f0-0x01f7
0x01f8-0x01ff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

keysym_to_unicode_2a1_2fe[] = {
0x0000, 0x0000, 0x0000, 0x0124,
0x011e, 0x0134, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0125,
0x011f, 0x0135, 0x0000, 0x0000,
0x0000, 0x0000, 0x010a, 0x0108,
0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0120, 0x0000,
0x0000, 0x0000, 0x016c, 0x015c,
0x0000, 0x0000, 0x010b, 0x0109,
0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0121, 0x0000,
0x0000, 0x0000, 0x016d, 0x015d

0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,
0x0000,

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x02a0-0x02a7
0x02a8-0x02af
0x02b0-0x02b7
0x02b8-0x02bf
0x02c0-0x02c7
0x02c8-0x02cf
0x02d0-0x02d7
0x02d8-0x02df
0x02e0-0x02e7
0x02e8-0x02ef
0x02f0-0x02f7
0x02f8-0x02ff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static unsigned short const
0x0138,
0x0000, 0x0000, 0x0112,
0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0113,
0x0100, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0145, 0x014c,
0x0000, 0x0172, 0x0000,
0x0101, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000,
0x0000, 0x0146, 0x014d,
0x0000, 0x0173, 0x0000,
};

keysym_to_unicode_3a2_3fe[] = {
0x0156, 0x0000, 0x0128, 0x013b,
0x0122, 0x0166, 0x0000, 0x0000,
0x0157, 0x0000, 0x0129, 0x013c,
0x0123, 0x0167, 0x014a, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0116, 0x0000, 0x0000,
0x0136, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0168, 0x016a,
0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0117, 0x0000, 0x0000,
0x0137, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0169, 0x016b

0x0000,
0x0000,
0x0000,
0x014b,
0x012e,
0x012a,
0x0000,
0x0000,
0x012f,
0x012b,
0x0000,

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x03a0-0x03a7
0x03a8-0x03af
0x03b0-0x03b7
0x03b8-0x03bf
0x03c0-0x03c7
0x03c8-0x03cf
0x03d0-0x03d7
0x03d8-0x03df
0x03e0-0x03e7
0x03e8-0x03ef
0x03f0-0x03f7
0x03f8-0x03ff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static unsigned short const
0x3002, 0x3008,
0x30a3, 0x30a5, 0x30a7,
0x30fc, 0x30a2, 0x30a4,
0x30af, 0x30b1, 0x30b3,

keysym_to_unicode_4a1_4df[] = {
0x3009, 0x3001, 0x30fb, 0x30f2,
0x30a9, 0x30e3, 0x30e5, 0x30e7,
0x30a6, 0x30a8, 0x30aa, 0x30ab,
0x30b5, 0x30b7, 0x30b9, 0x30bb,

0x30a1,
0x30c3,
0x30ad,
0x30bd,

/*
/*
/*
/*

0x04a0-0x04a7
0x04a8-0x04af
0x04b0-0x04b7
0x04b8-0x04bf

*/
*/
*/
*/

Generated by Doxygen


else if (keysym > 0x209f && keysym < 0x20ad)
} else if (keysym > 0x169f && keysym < 0x16f7)
  return keysym_to_unicode_14a1_14ff[keysym - 0x14a1];
else if (keysym > 0x14a0 && keysym < 0x1500)
else if (keysym > 0x13bb && keysym < 0x13bf)
  return keysym_to_unicode_12a1_12fe[keysym - 0x12a1];
else if (keysym > 0x12a0 && keysym < 0x12ff)
else if (keysym > 0xe9f && keysym < 0xf00)
else if (keysym > 0xda0 && keysym < 0xdfa)
else if (keysym > 0xcde && keysym < 0xcfb)
  return keysym_to_unicode_aa1_afe[keysym - 0xaa1];
else if (keysym > 0x8a3 && keysym < 0x8ff)
else if (keysym > 0x8ad && keysym < 0x9ff)
else if (keysym > 0x9df && keysym < 0xa0f)
else if (keysym > 0x9f9 && keysym < 0xa00)
else if (keysym > 0x9d1 && keysym < 0xa01)
else if (keysym > 0x9c1 && keysym < 0xc0f)
else if (keysym > 0x9b0 && keysym < 0x9ff)
else if (keysym > 0xb7 && keysym < 0xc00)
else if (keysym > 0xa83 && keysym < 0xa84)
else if (keysym > 0x12a0 && keysym < 0x12ff)
else if (keysym > 0x12a0 && keysym < 0x12a1)
else if (keysym > 0x13bb && keysym < 0x13bf)
else if (keysym > 0x14a0 && keysym < 0x1500)
else if (keysym > 0x14a0 && keysym < 0x14a1)
else if (keysym > 0x11cf && keysym < 0x15f7)
else if (keysym > 0x144f && keysym < 0x1500)
else if (keysym > 0x114f && keysym < 0x115f)
else if (keysym > 0x114f && keysym < 0x115f)
else if (keysym > 0x11e7f && keysym < 0x11f7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
else if (keysym > 0x11e6f && keysym < 0x11e7f)
return keysym_to_unicode_20a0_20ac[ keysym - 0x20a0 ];
else
    return 0;
}
*/

/***********************************************************/

10.207  armscii_8.h

static const unsigned short armscii_8_2uni[96] = {
    /* 0xa0 */
    0x00a0, 0xfffd, 0x0587, 0x0589, 0x0029, 0x0028, 0x00bb, 0x00ab,
    0x2014, 0x002e, 0x055d, 0x002c, 0x002d, 0x058a, 0x2026, 0x055c,
    /* 0xb0 */
    0x055b, 0x055e, 0x0531, 0x0561, 0x0532, 0x0562, 0x0533, 0x0563,
    0x0534, 0x0564, 0x0535, 0x0565, 0x0536, 0x0566, 0x0537, 0x0567,
    /* 0xc0 */
    0x0538, 0x0539, 0x058a, 0x053a, 0x056a, 0x053b, 0x056b, 0x053c,
    0x056c, 0x053d, 0x056d, 0x053e, 0x056e, 0x053f, 0x056f,
    /* 0xd0 */
    0x0540, 0x0570, 0x0541, 0x0571, 0x0542, 0x0572, 0x0543, 0x0573,
    0x0544, 0x0574, 0x0545, 0x0575, 0x0546, 0x0576, 0x0547, 0x0577,
    /* 0xe0 */
    0x0548, 0x0578, 0x0549, 0x0579, 0x054a, 0x057a, 0x054b, 0x057b,
    0x054c, 0x057c, 0x054d, 0x057d, 0x054e, 0x057e, 0x054f, 0x057f,
    /* 0xf0 */
    0x0550, 0x0580, 0x0551, 0x0581, 0x0552, 0x0582, 0x0553, 0x0583,
    0x0554, 0x0584, 0x0555, 0x0585, 0x0556, 0x0586, 0x055a, 0xfffd,
    0x0026);

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;
    } else {
        unsigned short wc = armscii_8_2uni[ c - 0xa0 ];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    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, 0xa7, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0xa6, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
};

static const unsigned char armscii_8_page05[96] = {
    0x00, 0xb2, 0xb4, 0xb6, 0xb8, 0x0a, 0xb0, 0x0b, /* 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, 0xe2, 0xe4, /* 0xe8-0xef */
    0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe, /* 0xf8-0xff */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
};

static const unsigned char armscii_8_page20[24] = {
    0x00, 0x00, 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 */
};

Generated by Doxygen
```c
static int armscii_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0028) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0028 && wc < 0x0030)
        c = armscii_8_page00[wc-0x0028];
    else if (wc >= 0x0030 && wc < 0x00a0)
        c = wc;
    else if (wc >= 0x00a0 && wc < 0x00c0)
        c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)
        c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)
        c = armscii_8_page20[wc-0x2010];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static int ascii_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;
    }
    return RET_ILSEQ;
}

static int ascii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    return RET_ILSEQ;
}
```
01679 0x7a64, 0x7a69, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bd,
01679 0x77bf, 0x77b4, 0x77bd, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4,
01680 0x77bf, 0x77b4, 0x77bd, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4,
01680 0x77bf, 0x77b4, 0x77bd, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4,
01681 0x77bd, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4,
01681 0x77bd, 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4,
01682 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01682 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01683 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01683 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01684 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01684 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01685 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01685 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01686 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01686 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01687 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01687 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01688 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
01688 0x77b5, 0x77ab, 0x77bd, 0x77b7, 0x77b4, 0x77bf, 0x77b4, 0x77b5,
static int big5_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c1 = s[0];
    if ((c1 >= 0xa1 && c1 <= 0xc7) || (c1 >= 0xc9 && c1 <= 0xf9)) {
        if (n >= 2) {
            unsigned char c2 = s[1];
            if ((c2 >= 0x40 && c2 < 0x7f) || (c2 >= 0xa1 && c2 < 0xff)) {
                unsigned int i = 157 * (c1 - 0xa1) + (c2 - (c2 >= 0xa1 ? 0x62 : 0x40));
                unsigned short wc = 0xfffd;
                if (i < 6280) {
                    if (i < 6121)
                        wc = big5_2uni_pagea1[i];
                    else
                        if (i < 13932)
                            wc = big5_2uni_pagec9[i-6280];
                }
                if (wc != 0xfffd) {
                    *pwc = (ucs4_t) wc;
                    return 2;
                }
            }
        }
    }
    return RET_ILSEQ;
}

return RET_TOOFEW(0);
}

return RET_ILSEQ;

}

endif /* NEED_TOWC */

#define NEED_TOMB

static const unsigned short big5_2charset[13703] = {
    0xa246, 0xa247, 0xa244, 0xa1b1, 0xa258, 0xa1d3, 0xa150, 0xa1d1,
    0xa1d2, 0xa3be, 0xa3bc, 0xa3bd, 0xa3bf, 0xa3bb, 0xa344, 0xa345,
    0xa346, 0xa347, 0xa348, 0xa349, 0xa34a, 0xa34b, 0xa34c, 0xa34d,
    0xa34e, 0xa34f, 0xa350, 0xa351, 0xa352, 0xa353, 0xa354, 0xa355,
    0xa356, 0xa357, 0xa358, 0xa359, 0xa35a, 0xa35b, 0xa35c, 0xa35d,
    0xa35e, 0xa35f, 0xa360, 0xa361, 0xa362, 0xa363, 0xa364, 0xa365,
    0xa366, 0xa367, 0xa368, 0xa369, 0xa36a, 0xa36b, 0xa36c, 0xa36d,
    0xa36e, 0xa36f, 0xa370, 0xa371, 0xa372, 0xa373, 0xa78b, 0xc77b,
    0xc77c, 0xc77d, 0xc77e, 0xc77f, 0xc780, 0xc781, 0xc782, 0xc783,
    0xc784, 0xc785, 0xc786, 0xc787, 0xc788, 0xc789, 0xc78a, 0xc78b,
    0xc78c, 0xc78d, 0xc78e, 0xc78f, 0xc790, 0xc791, 0xc792, 0xc793,
    0xc794, 0xc795, 0xc796, 0xc797, 0xc798, 0xc799, 0xc79a, 0xc79b,
    0xc79c, 0xc79d, 0xc79e, 0xc79f, 0xc7a0, 0xc7a1, 0xc7a2, 0xc7a3,
    0xc7a4, 0xc7a5, 0xc7a6, 0xc7a7, 0xc7a8, 0xc7a9, 0xc7aa, 0xc7ab,
    0xc7ac, 0xc7ad, 0xc7ae, 0xc7af, 0xc946, 0x9c97, 0xc94b, 0xc94c,
    0xc94d, 0xc94e, 0xc94f, 0xc950, 0xc951, 0xc952, 0xc953, 0xc954,
    0xc955, 0xc956, 0xc957, 0xc958, 0xc959, 0xc95a, 0xc95b, 0xc95c,
    0xc95d, 0xc95e, 0xc95f, 0xc960, 0xc961, 0xc962, 0xc963, 0xc964,
    0xc965, 0xc966, 0xc967, 0xc968, 0xc969, 0xc96a, 0xc96b, 0xc96c,
    0xc96d, 0xc96e, 0xc96f, 0xc970, 0xc971, 0xc972, 0xc973, 0xc974,
    0xc975, 0xc976, 0xc977, 0xc978, 0xc979, 0xc97a, 0xc97b, 0xc97c,
    0xc97d, 0xc97e, 0xc97f, 0xc980, 0xc981, 0xc982, 0xc983, 0xc984,
    0xc985, 0xc986, 0xc987, 0xc988, 0xc989, 0xc98a, 0xc98b, 0xc98c,
    0xc98d, 0xc98e, 0xc98f, 0xc990, 0xc991, 0xc992, 0xc993, 0xc994,
    0xc995, 0xc996, 0xc997, 0xc998, 0xc999, 0xc99a, 0xc99b, 0xc99c,
    0xc99d, 0xc99e, 0xc99f, 0xc9a0, 0xc9a1, 0xc9a2, 0xc9a3, 0xc9a4,
    0xc9a5, 0xc9a6, 0xc9a7, 0xc9a8, 0xc9a9, 0xc9aa, 0xc9ab, 0xc9ac,
    0xc9ad, 0xc9ae, 0xc9af, 0xc9b0, 0xc9b1, 0xc9b2, 0xc9b3, 0xc9b4,
    0xc9b5, 0xc9b6, 0xc9b7, 0xc9b8, 0xc9b9, 0xc9ba, 0xc9bb, 0xc9bc,
    0xc9bd, 0xc9be, 0xc9bf, 0xc9c0, 0xc9c1, 0xc9c2, 0xc9c3, 0xc9c4,
    0xc9c5, 0xc9c6, 0xc9c7, 0xc9c8, 0xc9c9, 0xc9ca, 0xc9cb, 0xc9cc,
    0xc9cd, 0xc9ce, 0xc9cf, 0xc9d0, 0xc9d1, 0xc9d2, 0xc9d3, 0xc9d4,
    0xc9d5, 0xc9d6, 0xc9d7, 0xc9d8, 0xc9d9, 0xc9da, 0xc9db, 0xc9dc,
    0xc9dd, 0xc9de, 0xc9df, 0xc9e0, 0xc9e1, 0xc9e2, 0xc9e3, 0xc9e4,
    0xc9e5, 0xc9e6, 0xc9e7, 0xc9e8, 0xc9e9, 0xc9ea, 0xc9eb, 0xc9ec,
    0xc9ed, 0xc9ee, 0xc9ef, 0xc9f0, 0xc9f1, 0xc9f2, 0xc9f3, 0xc9f4,
    0xc9f5, 0xc9f6, 0xc9f7, 0xc9f8, 0xc9f9, 0xc9fa, 0xc9fb, 0xc9fc,
    0xc9fd, 0xc9fe, 0xc9ff, 0xca00, 0xca01, 0xca02, 0xca03, 0xca04,
static const Summary16 big5_uni2indx_page00[16] = {
  { 0, 0x0000 }, { 0, 0x0000 }, { 0, 0x0000 }, { 0, 0x0000 },
  { 0, 0x0000 }, { 0, 0x0000 }, { 0, 0x0000 }, { 0, 0x0000 },
  { 7, 0x0000 }, { 8, 0x0080 }, { 8, 0x0080 }, { 8, 0x0080 },
};

static const Summary16 big5_uni2indx_page02[38] = {
  { 9, 0x0000 }, { 9, 0x0000 }, { 9, 0x0000 }, { 9, 0x0000 },
  { 9, 0x0000 }, { 9, 0x0000 }, { 9, 0x0000 }, { 9, 0x0000 },
};

static const Summary16 big5_uni2indx_page20[44] = {
  { 118, 0x0000 }, { 118, 0x3318 }, { 124, 0x0064 }, { 127, 0x4824 },
  { 131, 0x0000 }, { 131, 0x0000 }, { 131, 0x0000 }, { 131, 0x0000 },
};

static const Summary16 big5_uni2indx_page24[37] = {
  { 172, 0x0000 }, { 172, 0x0318 }, { 178, 0x0064 }, { 181, 0x4824 },
  { 185, 0x0000 }, { 185, 0x0000 }, { 185, 0x0000 }, { 185, 0x0000 },
};

static const Summary16 big5_uni2indx_page30[62] = {
  { 252, 0xff2f }, { 265, 0x6037 }, { 272, 0x03ff }, { 279, 0x03ff },
  { 285, 0x0000 }, { 285, 0x0000 }, { 285, 0x0000 }, { 285, 0x0000 },
};

static const Summary16 big5_uni2indx_page4e[1307] = {
  { 502, 0xff8b }, { 515, 0xc373 }, { 523, 0x6840 }, { 527, 0x1b0f },
  { 535, 0xe9ac }, { 544, 0xf34c }, { 553, 0x0200 }, { 554, 0xc008 },
};
04029 04030 04031 04032 04033 /* 0x9500 */ 04034 /* 0x9500 */ 04035 /* 0x9500 */ 04036 /* 0x9500 */ 04037 /* 0x9500 */ 04038 /* 0x9600 */ 04039 /* 0x9600 */ 04040 /* 0x9600 */ 04041 /* 0x9600 */ 04042 /* 0x9600 */ 04043 /* 0x9600 */ 04044 /* 0x9600 */ 04045 /* 0x9600 */ 04046 /* 0x9600 */ 04047 /* 0x9600 */ 04048 /* 0x9600 */ 04049 /* 0x9600 */ 04050 /* 0x9600 */ 04051 /* 0x9600 */ 04052 / * 0x9700 */ 04053 / * 0x9700 */ 04054 / * 0x9700 */ 04055 / * 0x9700 */ 04056 / * 0x9700 */ 04057 / * 0x9700 */ 04058 / * 0x9700 */ 04059 /* 0x9800 */ 04060 /* 0x9800 */ 04061 /* 0x9800 */ 04062 /* 0x9800 */ 04063 /* 0x9800 */ 04064 /* 0x9800 */ 04065 / * 0x9900 */ 04066 / * 0x9900 */ 04067 / * 0x9900 */ 04068 / * 0x9900 */ 04069 / * 0x9900 */ 04070 / * 0x9900 */ 04071 / * 0x9900 */ 04072 / * 0x9900 */ 04073 / * 0x9900 */ 04074 / * 0x9900 */ 04075 / * 0x9900 */ 04076 / * 0x9900 */ 04077 / * 0x9900 */ 04078 / * 0x9900 */ 04079 / * 0x9900 */ 04080 / * 0x9900 */ 04081 / * 0x9900 */ 04082 / * 0x9900 */ 04083 / * 0x9900 */ 04084 / * 0x9900 */ 04085 / * 0x9900 */ 04086 / * 0x9900 */ 04087 /

04088 static const Summary16 big5_uni2indx_pagefa[1] = { 04089 / * 0xfa00 */ 04090 / * 0xfb00 */ 04091 / * 0xfc00 */ 04092 static const Summary16 big5_uni2indx_pagef[23] = { 04093 / * 0x0000 */ 04094 / * 0x0000 */ 04095 / * 0x0000 */ 04096 / * 0x0000 */ 04097 / * 0x0000 */ 04098 / * 0x0000 */ 04099 / * 0x0000 */ 04100 / * 0x0000 */ 04101 /

04102 04103 static int 04104 big5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) 04105 { 04106 if (n >= 2) { 04107 const Summary16 *summary = NULL; 04108 if (wc < 0x1000) { 04109 summary = &big5_uni2indx_page00[(wc+4)]; 04110 else if (wc >= 0x0200 && wc < 0x0460) { 04111 summary = &big5_uni2indx_page02[(wc-4)-0x200]; 04112 else if (wc >= 0x0400 && wc < 0x2200) { 04113 summary = &big5_uni2indx_page20[(wc-4)-0x200]; 04114 else if (wc >= 0x2400 && wc < 0x2650) { 04115 summary = &big5_uni2indx_page24[(wc-4)-0x240];

Generated by Doxygen
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 & 0xcccc) >> 2);
        used = (used & 0x0f0f) + ((used & 0xf0f0) >> 4);
        used = (used & 0x00ff) + (used >> 8);
        c = big5_2charset[summary->indx + used];
        r[0] = (c » 8); r[1] = (c & 0xff);
        return 2;
    }
}
return RET_ILSEQ;
return RET_TOOSMALL;
#endif /* NEED_TOMB */
if (n >= 2) {
    unsigned char c2 = s[1];
    if (c2 >= 0x21 && c2 <= 0x7e) {
        unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
        if (0) /* Unoptimized. */{
            unsigned char buf[2];
            buf[0] = (i / 157) + 0xa1;
            buf[1] = i % 157;
            return big5_mbtowc(conv,pwc,buf,2);
        } else {
            /* Inline the implementation of big5_mbtowc. */
            if (wc != 0xfffd) {
                *pwc = (ucs4_t) wc;
                return 2;
            }
        }
    } else {
        /* Unoptimized. */
        unsigned char buf[2];
        buf[0] = (i / 157) + 0xc9;
        buf[1] = i % 157;
        return big5_mbtowc(conv,pwc,buf,2);
    }
}

return RET_ILSEQ;

return RET_TOOFEW(0);
return RET_ILSEQ;
return RET_ILSEQ;
return RET_ILSEQ;

static int
big5_1_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c1 = s[0];
    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 (0) /* Unoptimized. */{
                    unsigned char buf[2];
                    buf[0] = (i / 157) + 0xc9;
                    buf[1] = i % 157;
                    return big5_mbtowc(conv,pwc,buf,2);
                } else {
                    /* Inline the implementation of big5_mbtowc. */
                    if (i < 7652) {
                        unsigned short wc = big5_2uni_pagec9[i];
                        if (wc != 0xfffd) {
                            *pwc = (ucs4_t) wc;
                            return 2;
                        }
                    }
                }
            }
        }
    }
    return RET_TOOFEW(0);
return RET_ILSEQ;
return RET_ILSEQ;
return RET_ILSEQ;
return RET_ILSEQ;

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 != 2) abort();
        unsigned char s1, s2;
        s1 = buf[0];
        s2 = buf[1];
        if (!(s1 >= 0xa1)) abort();
        if (0x21 <= s2 && s2 <= 0x7e) {
            r[0] = (t / 94) + 0xa1;
            r[1] = (t % 94) + 0x21;
            return 2;
        }
    }
    return RET_TOOSMALL;
return RET_ILSEQ;
return RET_ILSEQ;
return RET_ILSEQ;
return RET_ILSEQ;

static int
big5_5_wctomb (conv_t conv, ucs4_t wc, const unsigned char *s, int n)
{
    unsigned char c1 = s[0];
    if (c1 >= 0x21 && c1 <= 0x70) {
        if (n >= 2) {
            unsigned char c2 = s[1];
            if (c2 >= 0x21 && c2 <= 0x7e) {
                unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
                if (0) /* Unoptimized. */{
                    unsigned char buf[2];
                    buf[0] = (i / 157) + 0xa1;
                    buf[1] = i % 157;
                    return big5_mbtowc(conv,pwc,buf,2);
                } else {
                    /* Inline the implementation of big5_mbtowc. */
                    if (wc != 0xfffd) {
                        *pwc = (ucs4_t) wc;
                        return 2;
                    }
                }
            }
        }
    }
    return RET TOOFEW(0);
return RET ILSEQ;
return RET ILSEQ;
return RET ILSEQ;
return RET ILSEQ;

static int
big5_6_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 != 2) abort();
        unsigned char s1, s2;
        s1 = buf[0];
        s2 = buf[1];
        if (!(s1 >= 0xa1)) abort();
        if (0x21 <= s2 && s2 <= 0x7e) {
            r[0] = (t / 94) + 0xa1;
            r[1] = (t % 94) + 0x21;
            return 2;
        }
    }
    return RET TOOSMALL;
return RET ILSEQ;
return RET ILSEQ;
return RET ILSEQ;
return RET ILSEQ;

Generated by Doxygen
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 >= 0x80 && 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 const unsigned short cp1133_2uni_1[64] = {
  0xa0, 0x00, 0xfc, 0x00, 0x00, 0x00, 0xfe, 0x00, /* 0xa0-0xa7 */
  0x00, 0x00, 0x00, 0x00, 0xfd, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
};

static const unsigned char cp1133_page00[16] = {
  0xa0, 0x00, 0xfe, 0x00, 0x00, 0x00, 0xfe, 0x00, /* 0xa0 */
  0x00, 0x00, 0x00, 0x00, 0xfd, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
};

static int cp1133_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) {
    unsigned short wc = cp1133_2uni_2[c-0xf0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  } else if (c < 0x80) {
    unsigned short wc = cp1133_2uni_1[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  } else if (c < 0x80) {
    unsigned short wc = cp1133_2uni_2[c-0xf0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  } else {
    unsigned short wc = cp1133_2uni_1[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
  return RET_ILSEQ;
}

static const unsigned char cp1133_page00[16] = {
  0xa0, 0x00, 0xfe, 0x00, 0x00, 0x00, 0xfe, 0x00, /* 0xa0 */
  0x00, 0x00, 0x00, 0x00, 0xfd, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
};
static const unsigned char cp1133_page0e[96] = {
  0x00, 0xa1, 0xa2, 0x00, 0xa3, 0x00, 0x00, 0xa4, /* 0x80-0x87 */
  0xa5, 0x00, 0xa7, 0x00, 0x00, 0xa8, 0x00, 0x00, /* 0x88-0x8f */
  0x00, 0xa9, 0xaa, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
  0x00, 0xaa, 0xab, 0xac, /* 0x98-0x9f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
  0x00, 0xb4, 0xb5, 0xb6, 0xb8, 0xb9, 0x00, 0x00, /* 0xa8-0xaf */
  0xc0, 0xca, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, /* 0xb0-0xb7 */
  0xc7, 0xc8, 0xcb, 0xcc, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
  0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xda, 0x00, /* 0xc0-0xc7 */
  0x00, 0x00, 0x00, 0x00, 0xb4, 0xb5, 0xb6, 0xb8, /* 0xc8-0xcf */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf5, 0xf6, 0xff, /* 0xd0-0xdf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static int cp1133_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;
}

static int cp1251_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 = cp1251_page00[wc-0x00a0];
  else if (wc >= 0x0e80 && wc < 0x0ee0)
  c = cp1251_page0e[wc-0x0e80];
  else if (wc == 0x20ad)
  c = 0xdf;
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

10.212 cp1251.h


#ifdef NEED_TOWC

static const unsigned short cp1251_2uni[128] = {
  /* 0x80 */
  0x0402, 0x0403, 0x201a, 0x0453, 0x201e, 0x2026, 0x2020, 0x2021,
  0x20ac, 0x2030, 0x0409, 0x2039, 0x040a, 0x040c, 0x040b, 0x040f,
  /* 0x90 */
  0x0452, 0x2018, 0x2019, 0x201c, 0x201d, 0x2013, 0x2014,
  0xfffd, 0x2122, 0x0459, 0x203a, 0x045a, 0x045c, 0x045b, 0x045f,
  /* 0xa0 */
  0x00a0, 0x040e, 0x045e, 0x0408, 0x0404, 0x0400, 0x040f, 0x0401,
  /* 0xb0 */
  0x00b0, 0x00b1, 0x0406, 0x0456, 0x0491, 0x00b5, 0x00b6, 0x00b7,
  0x00b2, 0x0416, 0x0456, 0x0400, 0x0458, 0x0405, 0x0455, 0x0457,
  /* 0xc0 */
  0x0410, 0x0411, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416, 0x0417,
  0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e, 0x041f,
  /* 0xd0 */
  0x0420, 0x0421, 0x0422, 0x0423, 0x0424, 0x0425, 0x0426, 0x0427,
  0x0428, 0x0429, 0x042a, 0x042b, 0x042c, 0x042d, 0x042e, 0x042f,
  /* 0xe0 */
  0x0430, 0x0431, 0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437,
  0x0438, 0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e, 0x043f,
  /* 0xf0 */
  0x0440, 0x0441, 0x0442, 0x0443, 0x0444, 0x0445, 0x0446, 0x0447,
  0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d, 0x044e, 0x044f,
};

static int cp1251_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = 0;
  if (c < 0x80) {
    *pwc = (ucs4_t) c;
    return 1;
  }
  else if (c >= 0xe000 && c < 0xe0ff)
  *pwc = (ucs4_t) c;
  return 1;
  return RET_ILSEQ;
}
return RET_ILSEQ;
#endif /* NEED_TOWC */

static const unsigned char cp1251_page00[32] = {
    0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0xa6, 0xa7, /* 0xa0-0xa7 */
    0x00, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0x00, /* 0xa8-0xaf */
    0xb0, 0xb1, 0x00, 0x00, 0x00, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0xb8, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
};

cp1251_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 < 0x00c0)
        c = cp1251_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x0498)
        c = cp1251_page04[wc-0x0400];
    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;
}
static int cp1255_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 = cp1255_2uni[c-0x80];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  }
  return RET_ILSEQ;
}

static const unsigned char cp1255_page00[88] = {
  0xa0, 0xa1, 0xa2, 0xa3, 0x00, 0xa5, 0xa6, 0xa7, /* 0xa0-0xa7 */
  0xa8, 0xa9, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, /* 0xa8-0xb5 */
  0xb6, 0xb7, 0xb8, 0xb9, 0x00, 0xbb, 0xbc, 0xbd, /* 0xb6-0xbf */
  0xbe, 0xbf, 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, /* 0xb8-0xbf */
  0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcd, 0xce, /* 0xc0-0xc7 */
  0xcf, 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, /* 0xc8-0xcf */
  0xd7, 0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, /* 0xd0-0xd7 */
  0xdf, 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, /* 0xd8-0xdf */
  0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, /* 0xe0-0xe7 */
  0xef, 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, /* 0xe8-0xef */
  0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, /* 0xe8-0xef */
};

static const unsigned char cp1255_page02[32] = {
  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 */
  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, /* 0x30-0x3f */
};

static const unsigned char cp1255_page05[72] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x3f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x7f */
};

static const unsigned char cp1255_page20[56] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x8f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x9f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xff */
};

static int cp1255_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  }
}

Generated by Doxygen
else if (wc >= 0x00a0 && wc < 0x00f8)
  c = cp1255_page00[wc-0x00a0];
else if (wc == 0x0192)
  c = 0x83;
else if (wc >= 0x02c0 && wc < 0x02e0)
  c = cp1255_page02[wc-0x02c0];
else if (wc >= 0x05b0 && wc < 0x05f8)
  c = cp1255_page05[wc-0x05b0];
else if (wc >= 0x2008 && wc < 0x2040)
  c = cp1255_page20[wc-0x2008];
else if (wc == 0x20aa)
  c = 0xa4;
else if (wc == 0x20ac)
  c = 0x80;
else if (wc == 0x2122)
  c = 0x99;
if (c != 0) {
  *r = c;
  return 1;
}
return RET_ILSEQ;

static const unsigned short cp1256_2uni[128] = {
  /* 0x80 */
  0x20ac, 0x067e, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
  0x02c6, 0x2030, 0x0679, 0x2039, 0x0152, 0x0686, 0x0698, 0x0688,
  /* 0x90 */
  0x06af, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
  0x06a9, 0x2122, 0x0691, 0x203a, 0x0153, 0x200c, 0x200d, 0x06ba,
  /* 0xa0 */
  0x00a0, 0x060c, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
  0x00a8, 0x00a9, 0x06be, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
  /* 0xb0 */
  0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
  0x00b8, 0x00b9, 0x061b, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
  /* 0xc0 */
  0x06c1, 0x0621, 0x0622, 0x0623, 0x0624, 0x0625, 0x0626, 0x0627,
  0x0628, 0x0629, 0x062a, 0x062b, 0x062c, 0x062d, 0x062e, 0x062f,
  /* 0xd0 */
  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,
  /* 0xe0 */
  0x00e0, 0x0644, 0x00e2, 0x0645, 0x0646, 0x0647, 0x0648, 0x0649,
  0x00e0, 0x064a, 0x00e0, 0x064b, 0x064c, 0x064d, 0x064e, 0x064f,
  0x00f1, 0x0650, 0x0651, 0x0652, 0x0653, 0x0654, 0x0655, 0x0656,
  0x0657, 0x0658, 0x0659, 0x065a, 0x065b, 0x065c, 0x065d, 0x065e,
  0x065f, 0x0660, 0x0661, 0x0662, 0x0663, 0x0664, 0x0665, 0x0666,
  0x0667, 0x0668, 0x0669, 0x066a, 0x066b, 0x066c, 0x066d, 0x066e,
  0x066f, 0x0670, 0x0671, 0x0672, 0x0673, 0x0674, 0x0675, 0x0676,
  0x0677, 0x0678, 0x0679, 0x067a, 0x067b, 0x067c, 0x067d, 0x067e,
  0x067f, 0x0680, 0x0681, 0x0682, 0x0683, 0x0684, 0x0685, 0x0686,
  0x0687, 0x0688, 0x0689, 0x068a, 0x068b, 0x068c, 0x068d, 0x068e,
  0x068f, 0x0690, 0x0691, 0x0692, 0x0693, 0x0694, 0x0695, 0x0696,
  0x0697, 0x0698, 0x0699, 0x069a, 0x069b, 0x069c, 0x069d, 0x069e,
  0x069f, 0x06a0, 0x06a1, 0x06a2, 0x06a3, 0x06a4, 0x06a5, 0x06a6,
  0x06a7, 0x06a8, 0x06a9, 0x06ba, 0x06bb, 0x06bc, 0x06bd, 0x06be,
  0x06bf, 0x06c0, 0x06c1, 0x06c2, 0x06c3, 0x06c4, 0x06c5, 0x06c6,
2004

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
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135

File Documentation

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, 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, 0x83, 0x00, 0x00, 0x00, 0x00, 0x00,
};
static const unsigned char cp1256_page06[208] = {
0x00, 0x00, 0x00, 0x00, 0xa1, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xba, 0x00, 0x00, 0x00, 0xbf,
0x00, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd8,
0xd9, 0xda, 0xdb, 0x00, 0x00, 0x00, 0x00, 0x00,
0xdc, 0xdd, 0xde, 0xdf, 0xe1, 0xe3, 0xe4, 0xe5,
0xe6, 0xec, 0xed, 0xf0, 0xf1, 0xf2, 0xf3, 0xf5,
0xf6, 0xf8, 0xfa, 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,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x8a, 0x00, 0x00, 0x00, 0x00, 0x81, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x8d, 0x00,
0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x9a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x8e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x98, 0x00, 0x00, 0x00, 0x00, 0x00, 0x90,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x9f, 0x00, 0x00, 0x00, 0xaa, 0x00,
0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
};
static const unsigned char cp1256_page20[56] = {
0x00, 0x00, 0x00, 0x00, 0x9d, 0x9e, 0xfd, 0xfe,
0x00, 0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00,
0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00,
0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x85, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x8b, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00,
};

/*
/*
/*
/*
/*
/*
/*
/*

0x58-0x5f
0x60-0x67
0x68-0x6f
0x70-0x77
0x78-0x7f
0x80-0x87
0x88-0x8f
0x90-0x97

*/
*/
*/
*/
*/
*/
*/
*/

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

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
0x80-0x87
0x88-0x8f
0x90-0x97
0x98-0x9f
0xa0-0xa7
0xa8-0xaf
0xb0-0xb7
0xb8-0xbf
0xc0-0xc7
0xc8-0xcf
0xd0-0xd7

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

/*
/*
/*
/*
/*
/*
/*

0x08-0x0f
0x10-0x17
0x18-0x1f
0x20-0x27
0x28-0x2f
0x30-0x37
0x38-0x3f

*/
*/
*/
*/
*/
*/
*/

static int
cp1256_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 < 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;
}

10.215

cp936ext.h

00001 /*
00002 * "$Id$"
00003 *
00004 * Character encoding support for the Fast Light Tool Kit (FLTK).
00005 *
00006 * Copyright 1998-2010 by Bill Spitzak and others.
00007 *

Generated by Doxygen


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

http://www.fltk.org/str.php

Please report all bugs and problems on the following page:

http://www.fltk.org/str.php

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

endif /* NEED_TOWC */

static int

static const unsigned short cp936ext_2uni_pages[23766] = {

0x4f02, 0x4f04, 0x4f05, 0x4f0f, 0x4f12, 0x4f17, 0x4f1f,
0x4f20, 0x4f21, 0x4f23, 0x4f26, 0x4f29, 0x4f2e, 0x4f31,
0x4f33, 0x4f35, 0x4f37, 0x4f3e, 0x4f40, 0x4f41, 0x4f42,
0x4f44, 0x4f46, 0x4f4a, 0x4f4e, 0x4f4f, 0x4f52, 0x4f54,
0x4f56, 0x4f58, 0x4f5a, 0x4f5b, 0x4f5d, 0x4f5f, 0x4f60,
0x4f62, 0x4f64, 0x4f66, 0x4f68, 0x4f6a, 0x4f6b, 0x4f6c,
0x4f6d, 0x4f6e, 0x4f72, 0x4f74, 0x4f75, 0x4f76, 0x4f77,
0x4f78, 0x4f79, 0x4f7a, 0x4f7b, 0x4f7c, 0x4f7d, 0x4f7f,
0x4f80, 0x4f81, 0x4f82, 0x4f83, 0x4f84, 0x4f85, 0x4f86,
0x4f87, 0x4f88, 0x4f8a, 0x4f8c, 0x4f8d, 0x4f8e, 0x4f8f,
0x4f90, 0x4f92, 0x4f93, 0x4f95, 0x4f96, 0x4f97, 0x4f98,
0x4f99, 0x4f9c, 0x4f9e, 0x4fa1, 0x4fa2, 0x4fa4, 0x4fa5,
0x4fa6, 0x4fa7, 0x4fa8, 0x4fa9, 0x4faa, 0x4fab, 0x4fac,
0x4fad, 0x4fae, 0x4fb0, 0x4fb2, 0x4fb3, 0x4fb4, 0x4fb5,
0x4fb6, 0x4fb7, 0x4fb8, 0x4fb9, 0x4fba, 0x4fbb, 0x4fbc,
0x4fbd, 0x4fbe, 0x4fc0, 0x4fc1, 0x4fc2, 0x4fc6, 0x4fc7, 0x4fc8, 0x4fc9,
0x4fcf, 0x4fd0, 0x4fd2, 0x4fd3, 0x4fd4, 0x4fd5, 0x4fd6,
0x4fd7, 0x4fd8, 0x4fd9, 0x4fda, 0x4fdc, 0x4fdf, 0x4fef,
0x4fef, 0x4ff0, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x4ff8, 0x4ff9, 0x4ff2, 0x4ff4, 0x4ff5, 0x4ff6, 0x4ff7,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd, 0x00fd,
```c
static const unsigned short cp936ext_page0600[304] = {
    0xa1e2, 0x0000, 0xa1e1, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x20-0x27
};

static const unsigned short cp936ext_page04a0[232] = {
    0xa2c1, 0xa2c2, 0xa2c3, 0xa2c4, 0x0000, 0x0000, 0x0000, 0x0000, 0x98-0x9f
};

static const unsigned short cp936ext_page048c[64] = {
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xa853, 0xb8-0xbf
};
```
0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0x0f

0x08-0-
06180 \texttt{cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)}

06181 \}
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_page0080[wc - 0x0400];
else if (wc >= 0x2010 && wc < 0x2040)
c = cp936ext_page0402[wc - 0x2010];
else if (wc >= 0x2100 && wc < 0x21a0)
c = cp936ext_page0420[wc - 0x2100];
else if (wc >= 0x2208 && wc < 0x22c0)
c = cp936ext_page0441[wc - 0x2208];
else if (wc == 0x2312)
c = 0xa1d0;
else if (wc >= 0x2460 && wc < 0x24a0)
c = cp936ext_page048c[wc - 0x2460];
else if (wc >= 0x2500 && wc < 0x25e8)
c = cp936ext_page04a0[wc - 0x2500];
else if (wc >= 0x2600 && wc < 0x2648)
c = cp936ext_page04c0[wc - 0x2600];
else if (wc >= 0x3000 && wc < 0x3130)
c = cp936ext_page0600[wc - 0x3000];
else if (wc >= 0x3220 && wc < 0x3238)
c = cp936ext_page0644[wc - 0x3220];
else if (wc == 0x32a3)
c = 0x9a48;
else if (wc >= 0x3388 && wc < 0x33d8)
c = cp936ext_page0671[wc - 0x3388];
else if (wc >= 0x3400 && wc < 0x39fa)
c = cp936ext_page09c9[wc - 0x3400];
else if (wc == 0xf92c)
c = 0xf9dc;
else if (wc >= 0xf9f8)
c = cp936ext_pagelf2f[wc - 0xff98];
else if (wc >= 0x9f60 && wc < 0x9f9f)
c = cp936ext_pagelf3c[wc - 0x9f60];
else if (wc >= 0xfaf0)
c = cp936ext_pagelf41[wc - 0xfaf0];
else if (wc >= 0xfaf0)
c = cp936ext_pagelf41[wc - 0xfaf0];
else if (wc >= 0xffe0)
c = cp936ext_pagelfffc[wc - 0xffe0];
else if (c != 0) {
    r[0] = (c >> 8); r[1] = (c & 0xff);
    return 2;
} else if (c != 0) {
    r[0] = (c >> 8); r[1] = (c & 0xff);
    return 2;
}
return RET_ILSEQ;
return RET_TOOSMALL;
} #endif /* NEED_TOMB */
#endif /* CP936 */
#endif /* __APPLE__ WIN32 */
/*
 * End of "$Id$".
 */
10.216 gb2312.h


static const unsigned short gb2312_2uni_page21[831] = {
    0x3000, 0x3001, 0x3002, 0x30fb, 0x02c9, 0x02c7, 0x00a8, 0x3003,
    0x0305, 0x030c, 0x035e, 0x026e, 0x026c, 0x026a, 0x0268, 0x0266,
    0x3060, 0x3061, 0x3062, 0x3063, 0x3064, 0x0214, 0x0213, 0x0212,
    0x0210, 0x020e, 0x020c, 0x020a, 0x0208, 0x0206, 0x0204, 0x0202,
    0x0200, 0x01f8, 0x01f6, 0x01f4, 0x01f2, 0x01fa, 0x01f8, 0x01f6,
    0x01f4, 0x01f2, 0x01f0, 0x01e8, 0x01e6, 0x01e4, 0x01e2, 0x01e0,
    0x01c8, 0x01c6, 0x01c4, 0x01c2, 0x01c0, 0x01a8, 0x01a6, 0x01a4,
    0x01a2, 0x01a0, 0x01a8, 0x01a6, 0x01a4, 0x01a2, 0x01a0, 0x0198,
    0x0196, 0x0194, 0x0192, 0x0190, 0x0198, 0x0196, 0x0194, 0x0192,
    0x0190, 0x0188, 0x0186, 0x0184, 0x0182, 0x0180, 0x0168, 0x0166,
    0x0164, 0x0162, 0x0160, 0x0168, 0x0166, 0x0164, 0x0162, 0x0160,
    0x0148, 0x0146, 0x0144, 0x0142, 0x0140, 0x0128, 0x0126, 0x0124,
    0x0122, 0x0120, 0x0108, 0x0106, 0x0104, 0x0102, 0x0100, 0x0088,
    0x0086, 0x0084, 0x0082, 0x0080, 0x0068, 0x0066, 0x0064, 0x0062,
    0x0060, 0x0048, 0x0046, 0x0044, 0x0042, 0x0040, 0x0028, 0x0026,
    0x0024, 0x0022, 0x0020, 0x0008, 0x0006, 0x0004, 0x0002, 0x0000,

Generated by Doxygen
01061 );
01062
01063 static int
01064 gb2312_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
01065 {
01066     unsigned char c1 = (s[0] & 0x7F);
01067     if ((c1 >= 0x21 && c1 <= 0x29) || (c1 >= 0x30 && c1 <= 0x77)) {
01068         if (n >= 2) {
01069             unsigned char c2 = (s[1] & 0x7F);
01070             if (c2 >= 0x21 && c2 < 0x7F) {
01071                 unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
01072                 unsigned short wc = 0xfffd;
01073                 if (i < 1410) {
01074                     if (i < 831)
01075                         wc = gb2312_2uni_page21[i];
01076                     } else {
01077                         if (i < 8178)
01078                             wc = gb2312_2uni_page30[i-1410];
01079                     }
01080                 if (wc != 0xfffd) {
01081                     *pwc = (ucs4_t) wc;
01082                     return 2;
01083                 }
01084             }
01085             return RET_ILSEQ;
01086         }
01087         return RET_TOOFEW(0);
01088     } else {
01089         return RET_ILSEQ;
01090     }
01091 }
01092 #endif /* NEED_TOWC */
01093
01094 #ifdef NEED_TOMB
01095     static const unsigned short gb2312_2charset[7445] = {
01096         0x2168, 0x216c, 0x2127, 0x2140, 0x2141, 0x2824, 0x2822,
01097         0x282f, 0x2829, 0x2830, 0x2831, 0x2832, 0x2833, 0x2834,
01098         0x2835, 0x2836, 0x2837, 0x2838, 0x2839, 0x283a, 0x283b,
01099         0x283c, 0x283d, 0x283e, 0x283f, 0x2840, 0x2841, 0x2842,
01100         0x2843, 0x2844, 0x2845, 0x2846, 0x2847, 0x2848, 0x2849,
01101         0x284a, 0x284b, 0x284c, 0x284d, 0x284e, 0x284f, 0x2850,
01102         0x2851, 0x2852, 0x2853, 0x2854, 0x2855, 0x2856, 0x2857,
01103         0x2858, 0x2859, 0x285a, 0x285b, 0x285c, 0x285d, 0x285e,
01104         0x285f, 0x2860, 0x2861, 0x2862, 0x2863, 0x2864, 0x2865,
01105         0x2866, 0x2867, 0x2868, 0x2869, 0x286a, 0x286b, 0x286c,
01106         0x286d, 0x286e, 0x286f, 0x2870, 0x2871, 0x2872, 0x2873,
01107         0x2874, 0x2875, 0x2876, 0x2877, 0x2878, 0x2879, 0x287a,
01108         0x287b, 0x287c, 0x287d, 0x287e, 0x287f, 0x2880, 0x2881,
01109         0x2882, 0x2883, 0x2884, 0x2885, 0x2886, 0x2887, 0x2888,
01110         0x2889, 0x288a, 0x288b, 0x288c, 0x288d, 0x288e, 0x288f,
01111         0x2890, 0x2891, 0x2892, 0x2893, 0x2894, 0x2895, 0x2896,
01112         0x2897, 0x2898, 0x2899, 0x289a, 0x289b, 0x289c, 0x289d,
01113         0x289e, 0x289f, 0x2900, 0x2901, 0x2902, 0x2903, 0x2904,
01114         0x2905, 0x2906, 0x2907, 0x2908, 0x2909, 0x290a, 0x290b,
01115         0x290c, 0x290d, 0x290e, 0x290f, 0x2910, 0x2911, 0x2912,
01116         0x2913, 0x2914, 0x2915, 0x2916, 0x2917, 0x2918, 0x2919,
if (used & ((unsigned short) 1 « i)) {
    unsigned int i = wc & 0x0f;
    summary = &gb2312_uni2indx_pageff[(wc»4)-0xff0];
} else if (wc >= 0xff00 && wc < 0xfff0)
    summary = &gb2312_uni2indx_page9e[(wc»4)-0x9e0];
else if (wc >= 0x3000 && wc < 0x3230)
    summary = &gb2312_uni2indx_page20[(wc»4)-0x200];
else if (wc < 0x0460)
    const Summary16 *summary = NULL;
else
    summary = &gb2312_uni2indx_page00[(wc»4)];
/* 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 = gb2312_2charset[summary->indx + used];
r[0] = (c » 8); r[1] = (c & 0xff);
return 2;
}
return RET_ILSEQ;
return RET_TOOSMALL;
#endif /* NEED_TOMB */
```c
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, 0x83, 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 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xda-0xdf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static const unsigned char georgian_ps_page01[32] = {
    0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static const unsigned char georgian_ps_page02[32] = {
    0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

Generated by Doxygen
```
static const unsigned char georgian_ps_page10[40] = {
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc8, /* 0xd0-0xd7 */
  0xc9, 0xca, 0xcb, 0xcc, 0xd0, 0xd1, /* 0xd8-0xdf */
  0xd2, 0xd3, 0xd4, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, /* 0xe0-0xe7 */
  0xdb, 0xdc, 0xdd, 0xde, 0xdf, 0xe0, 0xe1, 0xe3, /* 0xe8-0xef */
  0xe4, 0xc7, 0xce, 0xd5, 0xe2, 0xe5, 0x00, 0x00, /* 0xf0-0xf7 */
};

static const unsigned char georgian_ps_page20[48] = {
  0x00, 0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, /* 0x18-0x1f */
  0x86, 0x87, 0x95, 0x00, 0x80, 0x00, 0x85, 0x00, /* 0x20-0x27 */
  0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, /* 0x28-0x2f */
  0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
  0x00, 0x8b, 0x9b, 0x00, 0x80, 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 < 0x0080) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x0080 && wc < 0x00a0)
    c = georgian_ps_page00[wc-0x0080];
  else if ((wc >= 0x00a0 && wc < 0x00c0) || (wc >= 0x00e6 && wc < 0x0100))
    c = wc;
  else if (wc >= 0x0150 && wc < 0x0198)
    c = georgian_ps_page01[wc-0x0150];
  else if (wc >= 0x02c0 && wc < 0x02e0)
    c = georgian_ps_page02[wc-0x02c0];
  else if (wc >= 0x10d0 && wc < 0x10f8)
    c = georgian_ps_page10[wc-0x10d0];
  else if (wc >= 0x2010 && wc < 0x2040)
    c = georgian_ps_page20[wc-0x2010];
  else if (wc == 0x2122)
    c = 0x99;
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

static int iso8859_1_mbtowc (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 >= 0x0080 && wc < 0x00a0)
    c = georgian_ps_page00[wc-0x0080];
  else if (wc == 0x0080 && wc < 0x0100)
    c = georgian_ps_page01[wc-0x0100];
  else if (wc >= 0x0150 && wc < 0x0198)
    c = georgian_ps_page02[wc-0x0150];
  else if (wc >= 0x02c0 && wc < 0x02e0)
    c = georgian_ps_page10[wc-0x02c0];
  else if (wc == 0x02122)
    c = 0x99;
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

10.219 iso8859_1.h

10.220 iso8859_10.h
static const unsigned short iso8859_10_2uni[96] = {
/* 0xa0 */
0x00a0, 0x0104, 0x0112, 0x0122, 0x012a, 0x0128, 0x0136, 0x00a7,
/* 0xb0 */
0x00b0, 0x0105, 0x0113, 0x0123, 0x012b, 0x0129, 0x0137, 0x00ad,
/* 0xc0 */
0x0100, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x012e,
/* 0xd0 */
0x00d0, 0x0145, 0x014c, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x0168,
/* 0xe0 */
0x0101, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x014b,
/* 0xf0 */
0x00f0, 0x0146, 0x014d, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x0169,
};

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;
}

static int
iso8859_10_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
unsigned char c = 0;
if (wc < 0xa0) {
  *r = wc;
  return 1;
}
else if (wc >= 0xa0 && wc < 0x180)
  c = iso8859_10_page00[wc-0x00a0];
else if (wc == 0x2015)
  c = 0xbd;
if (c != 0) {
  *r = c;
  return 1;
}
else {
  return RET_ILSEQ;
}
}

#define NEED_WCMB /*

Generated by Doxygen
10.221 iso8859_11.h

00002
00003 /* ISO8859-11 */
00004
00005 #ifdef NEED_TOWC
00006 static const unsigned short iso8859_11_2uni[96] = {
00007 / * 0xa0 */
00008 0x00a0, 0x0e01, 0x0e02, 0x0e03, 0x0e04, 0x0e05, 0x0e06, 0x0e07,
00009 / * 0xb0 */
00010 0x0e08, 0x0e09, 0x0e0a, 0x0e0b, 0x0e0c, 0x0e0d, 0x0e0e, 0x0e0f,
00011 / * 0xc0 */
00012 0x0e10, 0x0e11, 0x0e12, 0x0e13, 0x0e14, 0x0e15, 0x0e16, 0x0e17,
00013 / * 0xd0 */
00014 0x0e18, 0x0e19, 0x0e1a, 0x0e1b, 0x0e1c, 0x0e1d, 0x0e1e, 0x0e1f,
00015 / * 0xe0 */
00016 0x0e20, 0x0e21, 0x0e22, 0x0e23, 0x0e24, 0x0e25, 0x0e26, 0x0e27,
00017 / * 0xf0 */
00018 0x0e28, 0x0e29, 0x0e2a, 0x0e2b, 0x0e2c, 0x0e2d, 0x0e2e, 0x0e2f,
00019 / * 0xf0 */
00020 0x0e30, 0x0e31, 0x0e32, 0x0e33, 0x0e34, 0x0e35, 0x0e36, 0x0e37,
00021 / * 0xf0 */
00022 0x0e38, 0x0e39, 0x0e3a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e3f,
00023 / * 0xf0 */
00024 0x0e40, 0x0e41, 0x0e42, 0x0e43, 0x0e44, 0x0e45, 0x0e46, 0x0e47,
00025 / * 0xf0 */
00026 0x0e48, 0x0e49, 0x0e4a, 0x0e4b, 0x0e4c, 0x0e4d, 0x0e4e, 0x0e4f,
00027 / * 0xf0 */
00028 0x0e50, 0x0e51, 0x0e52, 0x0e53, 0x0e54, 0x0e55, 0x0e56, 0x0e57,
00029 / * 0xf0 */
00030 0x0e58, 0x0e59, 0x0e5a, 0x0e5b, 0xfffd, 0xfffd, 0xfffd, 0x0e5f,
00031};
00032 static int
00033 iso8859_11_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00034 {
00035 unsigned char c = *s;
00036 if (c < 0x80) {
00037 *pwc = (ucs4_t) c;
00038 return 1;
00039 }
00040 else { 0x80}
00041 return RET_ILSEQ;
00042 }
00043 return RET_ILSEQ;
00044 }
00045 return RET_ILSEQ;
00046 #endif /* NEED_TOWC */
00047
00048 #ifdef NEED_TOMB
00049 static const unsigned char iso8859_11_page0e[96] = {
00050 / * 0x00-0x07 */
00051 0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
00052 / * 0x08-0x0f */
00053 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
00054 / * 0x10-0x17 */
00055 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
00056 / * 0x28-0x2f */
00057 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
00058 / * 0x38-0x3f */
00059 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
00060 / * 0x48-0x4f */
00061 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00062 / * 0x58-0x5f */
00063 0x80, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
00064 };
00065 static int
00066 iso8859_11_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00067 {
00068 unsigned char c = 0;
00069 if (wc < 0x0080 || wc == 0x00a0) |
00070 return 1;
00071 }
00072 return RET_ILSEQ;
00073 }
00074 else if (wc >= 0x0e00 && wc <= 0x0e0f)
00075 c = iso8859_11_page0e[wc-0x0e00];
00076 if (c != 0) |
00077 *r = c;
00078 return 1;
00079 }
00080 return RET_ILSEQ;
00081 }
00082 #endif /* NEED_TOMB */
00083
00084 #ifdef NEED_TOMB
00085 static const unsigned char iso8859_11_page0e[96] = {
00086 / * 0x00-0x07 */
00087 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00088 / * 0x28-0x2f */
00089 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
00090 / * 0x38-0x3f */
00091 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00092 / * 0x48-0x4f */
00093 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
00094 / * 0x58-0x5f */
00095 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00096 / * 0x68-0x6f */
00097 0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
00098 / * 0x78-0x7f */
00099 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00100 / * 0x48-0x4f */
00101 0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
00102 / * 0x58-0x5f */
00103 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
```

/* ISO-8859-13 */

#ifndef NEED_TOWC
    static const unsigned short iso8859_13_2uni[96] = {
        /* 0xa0 */
        0x00a0, 0x201d, 0x00a2, 0x00a3, 0x00a4, 0x201e, 0x00a6, 0x00a7,
        /* 0xb0 */
        0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x021c, 0x00b5, 0x00b6, 0x00b7,
        /* 0xc0 */
        0x0104, 0x012e, 0x0100, 0x0106, 0x00c4, 0x00c5, 0x0118, 0x0119,
        /* 0xd0 */
        0x0160, 0x0143, 0x0145, 0x00d3, 0x014c, 0x00d5, 0x00d6, 0x00d7,
        /* 0xe0 */
        0x0105, 0x012f, 0x0101, 0x0107, 0x00e4, 0x00e5, 0x0119, 0x011a,
        /* 0xf0 */
        0x0161, 0x0144, 0x0146, 0x00f3, 0x014d, 0x00f5, 0x00f6, 0x00f7,
    };
#endif /* NEED_TOWC */

#ifndef NEED_TOMB
    static const unsigned char iso8859_13_page00[224] = {
        0xa0, 0x00, 0xa2, 0xa3, 0xa4, 0x00, 0xa6, 0xa7, /* 0xa0-0xa7 */
        0x00, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0x00, /* 0xa8-0xaf */
        0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
        0x00, 0xb9, 0x00, 0xbc, 0xbd, 0xbe, 0x00, /* 0xb8-0xbf */
        0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, /* 0xc0-0xc7 */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xff */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xff */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x0f */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x1f */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x2f */
    };
#endif /* NEED_TOMB */

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 int iso8859_13_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) {
        *r = wc;
        return 1;
    } else if (wc >= 0x0180 && wc < 0x02a0) {
        *r = wc;
        return 1;
    }
    return 0;
}
```

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;
#endif /* NEED_TOWC */

10.223 iso8859_14.h


/* ISO-8859-14 */

#ifdef NEED_TOWC
static const unsigned short iso8859_14_2uni[96] = {
    0xa0, 0x1e02, 0x1e03, 0x00a3, 0x010a, 0x010b, 0x1e0a, 0x00a7,
    0x1e80, 0x00a9, 0x1e82, 0x1e0b, 0x1ef2, 0x00ad, 0x00ae, 0x0178,
    0x1e1e, 0x1e1f, 0x0120, 0x0121, 0x1e40, 0x1e41, 0x00b6, 0x1e56,
    0x1e81, 0x1e57, 0x1e83, 0x1e60, 0x1ef3, 0x1e84, 0x1e85, 0x1e61,
    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
    0x0174, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
    0x0175, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x016b,
    0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x0177, 0x00ff,
};

static int iso8859_14_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    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 */

#ifdef NEED_TOMB
static const unsigned char iso8859_14_page00[96] = {
    0xa0, 0x00, 0x00, 0xa3, 0x00, 0x00, 0x00, 0xa7, / * 0xa0-0xa7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xd0, 0xe0, 0xef,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x08-0x0f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x10-0x1f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x20-0x2f */
};
static const unsigned char iso8859_14_page01_0[32] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x10-0x1f */
};
static const unsigned char iso8859_14_page01_1[16] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x20-0x2f */
};
static const unsigned char iso8859_14_page1e_0[136] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x20-0x2f */
};
#endif /* NEED_TOWC */
static const unsigned char iso8859_14_page1e_1[8] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb7, 0xb9,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xb7, 0xbf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf7,
    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, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

static const unsigned char iso8859_14_page1e_0[8] = {
    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, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xf7, 0xf7, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xd7, 0xf7, 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, 0x00,
};

static int iso8859_14_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 < 0x0108) {
        c = iso8859_14_page00[wc-0x00a0];
    } else if (wc >= 0x0108 && wc < 0x0128) {
        c = iso8859_14_page01_0[wc-0x0108];
    } else if (wc >= 0x0170 && wc < 0x0180) {
        c = iso8859_14_page01_1[wc-0x0170];
    } else if (wc >= 0x1e00 && wc < 0x1e88) {
        c = iso8859_14_page1e_0[wc-0x1e00];
    } else if (wc >= 0x1ef0 && wc < 0x1ef8) {
        c = iso8859_14_page1e_1[wc-0x1ef0];
    }
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}
#endif /* NEED_TOMB */

static int iso8859_15_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0xa0 && c < 0xc0) {
        *pwc = (ucs4_t) iso8859_15_2uni[c-0xa0];
    } else {
        *pwc = (ucs4_t) c;
    }
    return 1;
}
#endif /* NEED_TOMC */

static int iso8859_15_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0xa0 && c < 0xc0) {
        *pwc = (ucs4_t) iso8859_15_2uni[c-0xa0];
    } else {
        *pwc = (ucs4_t) c;
    }
    return 1;
}
#endif /* NEED_TOMC */

static int iso8859_15_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0xa0 && c < 0xc0) {
        *pwc = (ucs4_t) iso8859_15_2uni[c-0xa0];
    } else {
        *pwc = (ucs4_t) c;
    }
    return 1;
}
#endif /* NEED_TOMC */

Generated by Doxygen
0004 0xbe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0xb8, 0x00, /* 0x78-0x7f */
0003 ];

0045 static int
0046 iso8859_15_wctomb (conv_t conv, unsigned char r, ucs4_t wc, int n)
0047 { 0048 unsigned char c = 0;
0049 if (wc < 0x00a0) {
0050 *r = wc;
0051 return 1;
0052 } 0053 else if (wc >= 0x00a0 && wc < 0x00c0)
0054 c = iso8859_15_page00[wc-0x00a0]; 0055 else if (wc >= 0x00c0 && wc < 0x0100)
0056 c = wc; 0057 else if (wc >= 0x0150 && wc < 0x0180)
0058 c = iso8859_15_page01[wc-0x0150]; 0059 else if (wc == 0x20ac)
0060 c = 0xa4; 0061 if (c != 0) {
0062 *r = c;
0063 return 1;
0064 }
0065 return RET_ILSEQ;
0066 }
0067 */ #endif /* NEED_TOMB */

10.225 iso8859_16.h


0002

0003 /* ISO-8859-16 */

0004 static const unsigned short iso8859_16_2uni[96] = {
0005 / * 0xa0 */
0006 0x00a0, 0x0104, 0x0105, 0x0141, 0x20ac, 0x201e, 0x0160, 0x00a7,
0007 0x0161, 0x00a9, 0x0218, 0x00ab, 0x0179, 0x00ad, 0x017a, 0x017b,
0008 / * 0xb0 */
0009 0x00b0, 0x00b1, 0x010c, 0x0142, 0x017d, 0x201d, 0x00b6, 0x00b7,
0010 0x017e, 0x010d, 0x0219, 0x00bb, 0x0152, 0x0153, 0x0178, 0x017c,
0011 / * 0xc0 */
0012 0x00c0, 0x00c1, 0x00c2, 0x0102, 0x00c4, 0x0105, 0x00c6, 0x00c7,
0013 0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
0014 / * 0xd0 */
0015 0x0110, 0x0143, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x0150, 0x00d5, 0x015a,
0016 0x0170, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x0118, 0x021a, 0x00df,
0017 / * 0xe0 */
0018 0x00e0, 0x00e1, 0x00e2, 0x0103, 0x00e4, 0x0107, 0x00e5, 0x00e6,
0019 0x00e7, 0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
0020 / * 0xf0 */
0021 0x0111, 0x0144, 0x00f2, 0x00f3, 0x00f4, 0x0151, 0x00f5, 0x00f6,
0022 0x0171, 0x00f8, 0x00fa, 0x00fb, 0x00fc, 0x0119, 0x021b, 0x00ff,
0023 / * 0x0100 */
0024 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0025 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0026 
0027 }

0028 static int

0029 iso8859_16_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
0030 { 0031 unsigned char c = *s;
0032 else if (c < 0xa0)
0033 *pwc = (ucs4_t) iso8859_16_2uni[c-0xa0];
0034 else if (wc >= 0x00a0 && wc < 0x00c0)
0035 c = wc;
0036 else if (wc >= 0x00c0 && wc < 0x0100)
0037 c = wc; 0038 else if (wc >= 0x0150 && wc < 0x0180)
0039 c = iso8859_16_page01[wc-0x0150];
0040 else if (wc == 0x20ac)
0041 c = 0xa4; 0042 if (c != 0) {
0043 *r = c;
0044 return 1;
0045 }
0046 return RET_ILSEQ;
0047 }
0048 */ #endif /* NEED_TOMB */

Generated by Doxygen
# ISO-8859-2

## DECLARE ISO-8859-2 MAPS

```c
#define NEED_TOWC

define iso8859_2_2uni [96] = 
    /* 0xa0 */ 0xa0, 0x0104, 0x02d8, 0x0141, 0x00a4, 0x013d, 0x015a, 0x00a7, 
                       0x00a8, 0x0160, 0x015e, 0x0164, 0x0179, 0x00ad, 0x017d, 0x017b, 
    /* 0xb0 */ 0x0154, 0x00c1, 0x00c2, 0x0102, 0x00c4, 0x0139, 0x0106, 0x00c7, 
                       0x010c, 0x00c9, 0x0118, 0x00cb, 0x011a, 0x00cd, 0x00ce, 0x010e, 
    /* 0xc0 */ 0x0110, 0x0143, 0x0147, 0x00d3, 0x00d4, 0x0150, 0x00d6, 0x00d7, 
                       0x0016, 0x015a, 0x0166, 0x00da, 0x00d8, 0x00dc, 0x00d4, 0x00d2, 0x00df, 
    /* 0xd0 */ 0x001c, 0x00e0, 0x00e1, 0x00e4, 0x00e5, 0x013a, 0x007f, 0x0080, 
                       0x010d, 0x00f0, 0x00f2, 0x00f4, 0x00f5, 0x00f6, 0x00f7, 
    /* 0xe0 */ 0x00f8, 0x0165, 0x0167, 0x00f9, 0x00fa, 0x0171, 0x00fc, 0x00fd, 0x0163, 0x02d9, 

static const unsigned char iso8859_16_page02[8] = { 
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 

static const unsigned char iso8859_16_page20[8] = { 
    0x00, 0x00, 0x00, 0x00, 0x00, 0xb5, 0xa5, 0x00, 

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 = 0xa4; 
    } if (c != 0) { 
        *r = c; 
        return 1; 
    } return RET_ILSEQ; 
} 
```

## DECLARE ISO-8859-2 MAPS

```c
static const unsigned char 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 = 0xa4; 
    } if (c != 0) { 
        *r = c; 
        return 1; 
    } return RET_ILSEQ; 
} 
```

## DECLARE ISO-8859-2 MAPS

```c
static const unsigned char 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 = 0xa4; 
    } if (c != 0) { 
        *r = c; 
        return 1; 
    } return RET_ILSEQ; 
} 
```

## DECLARE ISO-8859-2 MAPS

```c
static const unsigned char 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 = 0xa4; 
    } if (c != 0) { 
        *r = c; 
        return 1; 
    } return RET_ILSEQ; 
} 
```

## DECLARE ISO-8859-2 MAPS

```c
static const unsigned char 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 = 0xa4; 
    } if (c != 0) { 
        *r = c; 
        return 1; 
    } return RET_ILSEQ; 
} 
```
```c
#include <conv.h>

static const unsigned char iso8859_2_page00[224] = {
  0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
  0xa8, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
  0xb0, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
  0xb8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
  0x00, 0xc1, 0xc2, 0x00, 0xc4, 0x00, 0x00, 0x07, /* 0xc0-0xc7 */
  0x00, 0xc9, 0x00, 0xcb, 0x00, 0xcd, 0xce, 0x00, /* 0xc8-0xcf */
  0x00, 0x00, 0x00, 0xd3, 0xd4, 0x00, 0xd6, 0xd7, /* 0xd0-0xd7 */
  0x00, 0x00, 0xda, 0x00, 0xdc, 0xdd, 0x00, 0xdf, /* 0xd8-0xdf */
  0x00, 0xe1, 0xe2, 0x00, 0xe4, 0x00, 0x00, 0xe7, /* 0xe0-0xe7 */
  0x00, 0xe9, 0x00, 0xeb, 0x00, 0xed, 0xee, 0x00, /* 0xe8-0xef */
  0x00, 0x00, 0x00, 0xf3, 0xf4, 0x00, 0xf6, 0xf7, /* 0xf0-0xf7 */
  0x00, 0x00, 0xc3, 0xe3, 0xa1, 0xb1, 0xc6, 0xe6, /* 0x00-0x07 */
  0x00, 0x00, 0x00, 0x00, 0xc8, 0xe8, 0xcf, 0xef, /* 0x08-0x0f */
  0xd0, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0xca, 0xea, 0xc6, 0xec, 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 */
  0xf2, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
  0xd5, 0xf5, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0xad, 0xb6, 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_2_page02[32] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb7, /* 0xc0-0xc7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
};

static int
iso8859_2_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_2_page00[wc-0x00a0];
  else if (wc >= 0x02c0 && wc < 0x02e0)
    c = iso8859_2_page02[wc-0x02c0];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

static const unsigned char iso8859_3_2uni[96] = {
  0x00a0, 0x0126, 0x02d8, 0x00a3, 0x00a4, 0xfffd, 0x0124, 0x00a7, /* 0xa0 */
  0x00a8, 0x00a9, 0x00ba, 0x015e, 0x011e, 0x0134, 0x00ad, 0xfffd, 0x011b, /* 0xa1 */
  0x00bd, 0x00be, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x0125, 0x00b7, /* 0xb0 */
  0x00b8, 0x00b9, 0x00ba, 0x015f, 0x011f, 0x0135, 0x00bd, 0xfffd, 0x011c, /* 0xb1 */
  0x00c0, /* 0xc0 */
  0x00c6, 0x00c1, 0x00c2, 0xfffd, 0x00c4, 0x010a, 0x0108, 0x00c7, /* 0xe0 */
  0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf, /* 0xe1 */
  0x00d0, /* 0xe2 */
  0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x0120, 0x00d6, 0x00d7, /* 0xe3 */
  0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x011c, 0x00df, /* 0xe4 */
  0x00ee, 0x00e1, 0x00e2, 0xfffd, 0x00e4, 0x010b, 0x0109, 0x00e7, /* 0xe5 */
};
```
static int iso8859_3_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_3_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}

#ifdef NEED_TOMB
static const unsigned char iso8859_3_page00[96] = {
    0xa0, 0x00, 0x00, 0xa3, 0xa4, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
    0xa8, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
    0xb0, 0x00, 0xb2, 0xb3, 0xb4, 0xb5, 0x00, 0xb7, /* 0xb0-0xb7 */
    0xb8, 0x00, 0x00, 0x00, 0x00, 0xbd, 0x00, 0x00, /* 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0x00, 0xc4, 0x00, 0x00, 0xc7, /* 0xc0-0xc7 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xc8-0xcf */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0x00, 0xd6, 0xd7, /* 0xd0-0xd7 */
    0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0x00, 0xdf, 0xe0, /* 0xd8-0xdf */
    0xe0, 0xe1, 0xe2, 0xe0, 0xe4, 0x00, 0xe7, /* 0xe0-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
    0xf0, 0xf1, 0xf2, 0xf3, 0x00, 0xf6, 0xf7, /* 0xf0-0xf7 */
    0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0x00, 0x00, /* 0xf8-0xff */
};

static const unsigned char iso8859_3_page01[120] = {
    0xc6, 0xe6, 0xc5, 0xe5, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
    0x00, 0x00, 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 */
};

static const unsigned char iso8859_3_page02[8] = {
    0xa2, 0xff, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
};
#endif /* NEED_TOMB */

static int iso8859_3_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0xa0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0x0100)
        c = iso8859_3_page00[wc-0x00a0];
    else if (wc >= 0x0108 && wc < 0x0180)
        c = iso8859_3_page01[wc-0x0108];
    else if (wc >= 0x02d8 && wc < 0x02e0)
        c = iso8859_3_page02[wc-0x02d8];
    else if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

/* ISO-8859-4 */

#ifdef NEED_TOWC
  static const unsigned short iso8859_4_2uni[96] =
    /* 0xa0 */
    0x00a0, 0x0104, 0x0138, 0x0156, 0x00a4, 0x0128, 0x013b, 0x00a7,
    0x00a8, 0x0160, 0x0112, 0x0122, 0x0166, 0x00ad, 0x017d, 0x00af,
    /* 0xb0 */
    0x00b0, 0x0105, 0x02db, 0x0157, 0x00b4, 0x0129, 0x013c, 0x02c7,
    0x00b8, 0x0161, 0x0113, 0x0123, 0x0167, 0x014a, 0x011e, 0x014b,
    /* 0xc0 */
    0x0100, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x012a,
    0x010c, 0x00c9, 0x0118, 0x00cb, 0x0116, 0x00cd, 0x00ce, 0x012a,
    /* 0xd0 */
    0x0110, 0x0145, 0x014c, 0x0136, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00d8, 0x0175, 0x00da, 0x0160, 0x00dc, 0x0166, 0x00dd, 0x016f,
    /* 0xe0 */
    0x0101, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x012f,
    0x010d, 0x00e9, 0x0119, 0x00eb, 0x0117, 0x00ed, 0x00ee, 0x012b,
    /* 0xf0 */
    0x0111, 0x0146, 0x014d, 0x0137, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
    0x0119, 0x014d, 0x0137, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
    0x012a, 0x014f, 0x0156, 0x00f8, 0x0167, 0x014a, 0x011e, 0x014b,
  }

static int iso8859_4_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_4_2uni[c-0xa0];
  return 1;
}
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
  static const unsigned char iso8859_4_page00[224] =
    /* 0xa0-0xa7 */
    0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0x00, 0xa7,
    /* 0xa8-0xaf */
    0xa8, 0x00, 0x00, 0x00, 0xad, 0xb0, 0x00, 0x00,
    /* 0xb0-0xb7 */
    0xb0, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0x00,
    /* 0xb8-0xbf */
    0xb8, 0x00, 0x00, 0x00, 0xbf, 0x00, 0x00, 0x00,
    /* 0xc0-0xc7 */
    0xc0, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xc8-0xcf */
    0xc8, 0x00, 0x00, 0x00, 0x00, 0xc4, 0x00, 0x00,
    /* 0xd0-0xdf */
    0xd0, 0x00, 0x00, 0x00, 0xdf, 0x00, 0x00, 0x00,
    /* 0xe0-0xef */
    0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xf0-0xff */
    0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  }

  static const unsigned char iso8859_4_page02[32] =
    /* 0xc0-0xc7 */
    0xc0, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xc8-0xcf */
    0xc8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xd0-0xdf */
    0xd0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xe0-0xef */
    0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0xff */
    0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

static int iso8859_4_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, const unsigned char *s, int n)
{
  unsigned char c = *s;
  if (c < 0xa0) {
    *r = wc;
    return 1;
  }
  else
    *r = *(ues4_t *) iso8859_4_2uni[-0xa0];
  return 0;
}
#endif /* NEED_TOMB */

#endif /* NEED_TOWC */


00086        return 1;
00087    }
00088 else if (wc >= 0x00a0 && wc < 0x0180)
00089    c = iso8859_4_page00[wc-0x00a0];
00090 else if (wc >= 0x02c0 && wc < 0x02e0)
00091    c = iso8859_4_page02[wc-0x02c0];
00092 if (c != 0) {
00093    *r = c;
00094    return 1;
00095 }
00096  return RET_ILSEQ;
00097 }
00098 #endif /* NEED_TOMB */
```c
00070  c = iso8859_5_page00[wc-0x00a0];
00071  } else if (wc >= 0x0400 && wc < 0x0460)
00072      c = iso8859_5_page04[wc-0x0400];
00073  else if (wc == 0x2116)
00074      c = 0xf0;
00075  if (c != 0) {
00076        *r = c;
00077        return 1;
00078    }
00079    return RET_ILSEQ;
00080  }
00081  #endif /* NEED_TOMB */
```

10.230 iso8859_6.h

```c
00002
00003 /*
00004 * ISO-8859-6
00005 */
00006
00007 #ifdef NEED_TOWC
00008 static const unsigned short iso8859_6_2uni[96] = {
00009    0xa0, 0xfffd, 0xfffd, 0xfffd, 0x00a4, 0xfffd, 0xfffd, 0x00a6,
00010    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
00011    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
00012    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
00013    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
00014    0xfffd, 0xfffd, 0xfffd, 0x061b, 0xfffd, 0xfffd, 0xfffd, 0x061f,
00015    0xa0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
00016    0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
00017    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
00018    0x00b8, 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
00019    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
00020    0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
00021    0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
00022    0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
00023    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
00024    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
00025    0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
00026    0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00ff,
00027    0x00ff,
00028  }
00029
00030 static int 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 short iso8859_6_page00[16] = {
00050    0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
00051    0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
00052    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
00053    0x00b8, 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
00054    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
00055    0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
00056    0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
00057    0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
00058    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
00059    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
00060    0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
00061    0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00ff,
00062    0x00ff,
00063    0x00ff,
00064  }
00065
00066 static int iso8859_6_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00067 {
00068    unsigned char c = 0;
00069    if (wc < 0x00a0) {
```

Generated by Doxygen
```c
00071  *r = wc;
00072  return 1;
00073 }
00074 else if (wc >= 0x00a0 && wc < 0x00b0)
00075     c = iso8859_6_page00[wc-0x00a0];
00076 else if (wc >= 0x0608 && wc < 0x0658)
00077     c = iso8859_6_page06[wc-0x0608];
00078     if (c != 0) {
00079         *r = c;
00080         return 1;
00081     }
00082     return RET_ILSEQ;
00083 }
00084 #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 >= 0x0380 && wc < 0x03d0)
        c = iso8859_7_page03[wc-0x0380];
    else if (wc >= 0x2010 && wc < 0x2020)
        c = iso8859_7_page20[wc-0x2010];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}
#endif /* NEED_TOMB */

#define NEED_TOMB

test if NEED_TOMB */

10.232 iso8859_8.h

static const unsigned short iso8859_8_2uni[96] = {
    0xa0, 0xfffd, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
    0x00a8, 0x00a9, 0x00d7, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
    0x00b8, 0x00b9, 0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5,
    0x00c6, 0x00c7, 0x00c8, 0x00c9, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
    0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00ce, 0x00cf,
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
    0x00e8, 0x00e9, 0x00ea, 0x00ef, 0x00f0, 0x00f1, 0x00f2, 0x00f3,
    0x00f4, 0x00f5, 0x00f6, 0x00f7, 0x00f8, 0x00f9, 0x00fa, 0x00ff,
    0x0100, 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107,
    0x0108, 0x0109, 0x010a, 0x010b, 0x010c, 0x010d, 0x010e, 0x010f,
    0x0110, 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117,
    0x0118, 0x0119, 0x011a, 0x011b, 0x011c, 0x011d, 0x011e, 0x011f,
    0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127,
    0x0128, 0x0129, 0x012a, 0x012b, 0x012c, 0x012d, 0x012e, 0x012f,
    0x0130, 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137,
    0x0138, 0x0139, 0x013a, 0x013b, 0x013c, 0x013d, 0x013e, 0x013f,
    0x0140, 0x0141, 0x0142, 0x0143, 0x0144, 0x0145, 0x0146, 0x0147,
    0x0148, 0x0149, 0x014a, 0x014b, 0x014c, 0x014d, 0x014e, 0x014f,
    0x0150, 0x0151, 0x0152, 0x0153, 0x0154, 0x0155, 0x0156, 0x0157,
    0x0158, 0x0159, 0x015a, 0x015b, 0x015c, 0x015d, 0x015e, 0x015f,
    0x0160, 0x0161, 0x0162, 0x0163, 0x0164, 0x0165, 0x0166, 0x0167,
10.233 iso8859_9.h

00059 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-Ox9f */
00060 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, /* Oxf0-Oxff */
00061 */
00062 static const unsigned char iso8859_8_page05[32] = {
00063 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xd0-0xd7 */
00064 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* Ox8-0xff */
00065 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* Ox0-0xe7 */
00066 0xf8, 0xf9, 0xfa, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00067 */
00068 static const unsigned char iso8859_8_page20[20] = {
00069 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0xf9, /* 0x08-0x0f */
00070 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0d, /* Ox10-0x17 */
00071 */
00072 */
00073 static int
00074 iso8859_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00075 {
00076 unsigned char c = 0;
00077 if (wc < 0x00a0) {
00078 *r = wc;
00079 return 1;
00080 }
00081 else if (wc >= 0x00a0 && wc < 0x00f8)
00082 c = iso8859_8_page00[wc-0x00a0];
00083 else if (wc >= 0x05d0 && wc < 0x05f0)
00084 c = iso8859_8_page05[wc-0x05d0];
00085 else if (wc >= 0x2008 && wc < 0x2018)
00086 c = iso8859_8_page20[wc-0x2008];
00087 if (c != 0) {
00088 *r = c;
00089 return 1;
00090 }
00091 return RET_ILSEQ;
00092 }
00093 #endif /* NEED_TOMB */
00094 */
10.233 iso8859_9.h

00002 */
00003 */
00004 * ISO-8859-9
00005 */
00006 */
00007 #ifdef NEED_TOWC
00008 static const unsigned short iso8859_9_2uni[48] = {
00009 /* 0xd0 */
00010 0x011e, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
00011 0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x0130, 0x015e, 0x00df,
00012 /* 0xe0 */
00013 0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
00014 0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
00015 /* 0xf0 */
00016 0x011f, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
00017 0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x0131, 0x015f, 0x00ff,
00018 */
00019 */
00020 static int
00021 iso8859_9_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00022 {
00023 unsigned char c = 0;
00024 if (c >= 0xd0)
00025 *pwc = (ucs4_t) iso8859_9_2uni[c-0xd0];
00026 else
00027 *pwc = (ucs4_t) c;
00028 return 1;
00029 }
00030 #endif /* NEED_TOWC */
00031 */
00032 #ifdef NEED_TOMB
00033 static const unsigned char iso8859_9_page00[48] = {
00034 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, /* 0xd0-0xd7 */
00035 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x00, 0x00, 0x0f, /* Ox8-0xff */
00036 0x0e, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, /* Ox0-0xe7 */
00037 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, /* Ox8-0xff */
00038 0x1e, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, /* Ox0-0xff */
00039 0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, /* Ox8-0xff */
00040 */
00041 static const unsigned char iso8859_9_page01[72] = {
00042 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00043 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00044 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00045 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00046 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */
00047 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* Ox8-0xff */

Generated by Oxygen
static int iso8859_9_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x00d0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00d0 && wc < 0x0100)
    c = iso8859_9_page00[wc-0x00d0];
    else if (wc >= 0x0118 && wc < 0x0160)
    c = iso8859_9_page01[wc-0x0118];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}


static const unsigned short iso8859_9e_2uni[96] = {
    /* 0xa0 */
    0x00a0, 0x017d, 0x00a2, 0x00a3, 0x20ac, 0x00a5, 0x012c, 0x00a7,
    0x016c, 0x00a9, 0x01e6, 0x00ab, 0x014a, 0x00ad, 0x00ae, 0x01d1,
    /* 0xb0 */
    0x00b0, 0x017e, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x0112, 0x00b7,
    0x0113, 0x01bd, 0x01be, 0x00b9, 0x01e7, 0x00bb, 0x014b, 0x00bd,
    0x0178, 0x01d2,
    /* 0xc0 */
    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x011f, 0x00c7,
    0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
    /* 0xd0 */
    0x011f, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00d9, 0x00da, 0x00db, 0x00dc, 0x0130, 0x015e, 0x00df,
    /* 0xe0 */
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x0129, 0x00e7,
    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
    /* 0xf0 */
    0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
    0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x0131, 0x015f, 0x00ff,
    };
10.235 jisx0201.h

```c
00002
00003 /*
00004 * JISX0201.1976-0
00005 */
00006 #ifdef NEED_TOWC
00007
00008 static int
00009 jisx0201_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00010 {
00011 unsigned char c = 0;
00012 if (wc < 0x0080) {
00013     *r = wc;
00014     return 1;
00015 } else if (wc >= 0x010a && wc < 0x01a0) {
00016     *r = c + 0xfec0;
00017     return 1;
00018 } else if (wc >= 0x010d0 && wc < 0x01e8)
00019     return RET_ILSEQ;
00020     if (wc >= 0x01d0 && wc < 0x01e8)
00021     return RET_ILSEQ;
00022 }
00023 #endif /* NEED_TOWC */
00024
00025 #ifdef NEED_TOMB
00026
00027 static int
00028 jisx0201_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00029 {
00030     unsigned char c = *r;
00031     if (c < 0x80)
00032         return 1;
00033     else if (c == 0x5c)
00034         *r = 0x00a5;
00035     else if (c == 0x7e)
00036         *r = 0x203e;
00037     else
00038         *r = c;
00039     return 1;
00040 }
00041 #endif /* NEED_TOMB */
00042
```

10.235 jisx0201.h

```c
00002
00003 /*
00004 * JISX0201.1976-0
00005 */
00006 #ifdef NEED_TOWC
00007
00008 static int
00009 jisx0201_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00010 {
00011 unsigned char c = 0;
00012 if (wc < 0x0080) {
00013     *r = wc;
00014     return 1;
00015 } else if (wc >= 0x010a && wc < 0x01a0) {
00016     *r = c + 0xfec0;
00017     return 1;
00018 } else if (wc >= 0x010d0 && wc < 0x01e8)
00019     return RET_ILSEQ;
00020     if (wc >= 0x01d0 && wc < 0x01e8)
00021     return RET_ILSEQ;
00022 }
00023 #endif /* NEED_TOWC */
00024
00025 #ifdef NEED_TOMB
00026
00027 static int
00028 jisx0201_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00029 {
00030     unsigned char c = *r;
00031     if (c < 0x80)
00032         return 1;
00033     else if (c == 0x5c)
00034         *r = 0x00a5;
00035     else if (c == 0x7e)
00036         *r = 0x203e;
00037     else
00038         *r = c;
00039     return 1;
00040 }
00041 #endif /* NEED_TOMB */
00042
```
static const unsigned short jisx0208_2uni_page21[690] = {
    0x25ef,
    0x2261, 0x2252, 0x226a, 0x226b, 0x221a, 0x223d, 0x221d, 0x2235,
    0x2229, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
    0x7e, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x2262, 0x2264, 0x00b0, 0x00a8, 0xff3e, 0x00b4, 0x00a8, 0xff3b,
    0x2260, 0x225b, 0x225c7, 0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
    0x226c, 0x226d, 0x226a, 0x226b, 0x221a, 0x223d, 0x2235, 0x2234,
    0x00b6, 0x00b5, 0x00b0, 0x2032, 0x2033, 0x2103, 0xffe5, 0xff04,
    0x2212, 0x00a5, 0xff09, 0xff08, 0x00b8, 0x00b7, 0x00f7,
static int
jisx0208_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c1 = (s[0] & 0x7F);
    if ((c1 >= 0x21 && c1 <= 0x28) || (c1 >= 0x30 && c1 <= 0x74)) {
        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 < 690)
                        wc = jisx0208_2uni_page21[i];
                    else
                        if (i < 7808)
                            wc = jisx0208_2uni_page30[i-1410];
                }
                if (wc != 0xfffd) {
                    *pwc = (ucs4_t) wc;
                    return 2;
                }
            }
            return RET_ILSEQ;
        }
        return RET_TOOFEW(0);
    }
    return RET_ILSEQ;
}
static const Summary16 jisx0208_uni2indx_pageff[15] = {
  /
  0xff00 */
  { 6788, 0xdf7a }, { 6800, 0xffff }, { 6816, 0xffff }, { 6832, 0xffff },
  { 6847, 0xffff }, { 6863, 0xffff }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0028 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
  { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 }, { 6877, 0x0000 },
};

static int
jisx0208_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  if (n >= 2) {
    const Summary16 *summary = NULL;
    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 < 0x3110)
      summary = &jisx0208_uni2indx_page30[(wc»4)-0x300];
    else if (wc >= 0x4e00 && wc < 0x9f00)
      summary = &jisx0208_uni2indx_page4e[(wc»4)-0x4e0];
    else if (wc >= 0xff00 && wc < 0xffff)
      summary = &jisx0208_uni2indx_pageff[(wc»4)-0xff0];
    if (summary) {
      unsigned short used = summary->used;
      unsigned short c;
      /* Keep in 'used' only the bits 0..i-1. */
      used &= (unsigned short) 1 « i;
      /* 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);
      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 */
if (wc != 0xfffd) {
    wc = jisx0212_2uni_page30[i-1410];
}
if (i < 7211) {
    wc = jisx0212_2uni_page29[i-752];
} else if (i < 1410) {
    wc = jisx0212_2uni_page22[i-94];
    if (i < 175) {
        unsigned short wc = 0xfffd;
        if (c2 >= 0x21 && c2 < 0x7f) {
            unsigned char c2 = (s[1] & 0x7F);
            if ((c1 == 0x22) || (c1 >= 0x26 && c1 <= 0x27) || (c1 >= 0x30 && c1 <= 0x39)) {
                return RET_ILSEQ;  // Japanese hiragana
            } else if (i < 658) {
                wc = jisx0212_juni_page26[i-140];
                if (i < 1410) {
                    wc = jisx0212_juni_page29[i-752];
                    if (i < 7211) {
                        wc = jisx0212_juni_page30[i-1410];
                    }
                } else if (wc != 0xfffd) {
                    return RET_TOOFEW(0);
                }
            } else if (i < 1027) {
                return RET_ILSEQ;
            }
        }
    }
    return RET_ILSEQ;
}
```c
static const Summary16 jisx0212_uni2indx_page4e[1307] = {
        ... 
};

static const Summary16 jisx0212_uni2indx_page21[3] = {
    ... 
};
```

```c
static const Summary16 jisx0212_uni2indx_page4e[1307] = {
    ... 
};
```

```c
static const Summary16 jisx0212_uni2indx_page21[3] = {
    ... 
};
```

```c
static const Summary16 jisx0212_uni2indx_page4e[1307] = {
    ... 
};
```
10.237 jisx0212.h 2177
static int
jisx0212_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (n >= 2) {
        const Summary16 *summary = NULL;
        if (wc < 0x0460)
            summary = &jisx0212_uni2indx_page00[(wc»4)];
        else if (wc >= 0x2100 && wc < 0x2130)
            summary = &jisx0212_uni2indx_page21[(wc»4)-0x210];
        else if (wc >= 0x4e00 && wc < 0x9fb0)
            summary = &jisx0212_uni2indx_page4e[(wc»4)-0x4e0];
        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 & 0xaaa) >> 1);
                used = (used & 0x3333) + ((used & 0xcccc) >> 2);
                used = (used & 0x0f0f) + ((used & 0xf0f0) >> 4);
                used = (used & 0x00ff) + (used >> 8);
                c = jisx0212_2charset[summary->indx + used];
                r[0] = (c » 8); r[1] = (c & 0xff);
                return 2;
            }
        }
        return RET_ILSEQ;
    }
    return RET_TOOSMALL;
}
#endif /* NEED_TOMB */
10.238 koi8_c.h

static int koi8_c_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    if (c < 0x80)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) koi8_c_2uni[c-0x80];
    return 1;
}

static const unsigned char koi8_c_page00[1] = {
    0xa0, / * 0xa0-0xa7 */
};

static const unsigned char koi8_c_page04[240] = {
    0x00, 0xb3, 0xb1, 0xb2, 0xb4, 0xb5, 0xb6, 0xb7, / * 0x00-0x07 */
    0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xc0, 0x0e, 0x0f, / * 0x08-0x0f */
    0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa, / * 0x10-0x17 */
    0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0, / * 0x18-0x1f */
    0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe7, 0xe8, 0xe9, / * 0x20-0x27 */
    0xea, 0xb8, 0xb9, 0xb1, 0xb2, 0xb4, 0xb5, 0xb6, / * 0x28-0x2f */
    0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xc6, 0xc7, / * 0x30-0x37 */
    0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, / * 0x38-0x3f */
    0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0x08, 0x09, 0xda, / * 0x40-0x4f */
    0xb8, 0xb9, 0xb1, 0xb2, 0xb4, 0xb5, 0xb6, 0xb7, / * 0x50-0x57 */
    0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xa0, 0xa1, 0 xa2, / * 0x58-0x5f */
    0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, / * 0x60-0x67 */
    0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe7, 0xe8, 0xe9, / * 0x70-0x7f */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, / * 0x80-0x87 */
    0xf8, 0xf9, 0xfa, 0xfb, 0xfe, 0xff, 0x00, 0x01, / * 0x90-0x97 */
    0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f, / * 0xa0-0xa7 */
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, / * 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, / * 0xc8-0xff */
};

static 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 < 0x040f) c = koi8_c_page04[wc-0x0400];
    else if (wc >= 0x2216 && wc < 0x2217) c = koi8_c_page22[wc-0x2216];
    return c;
}

static int koi8_c_cmnwctomb (conv_t conv, unsigned char *s, int n) {
    unsigned char c = 0;
    if (c < 0x0080) {
        *s = c;
        return 1;
    } else if (c >= 0x00a0 && c < 0x00a1) c = koi8_c_page00[c-0x00a0];
    else if (c >= 0x0400 && c < 0x040f) c = koi8_c_page04[c-0x0400];
    else if (c >= 0x2216 && c < 0x2217) c = koi8_c_page22[c-0x2216];
    return c;
}

Generated by Doxygen
2180

File Documentation

00098
if (c != 0) {
00099
*r = c;
00100
return 1;
00101
}
00102
return RET_ILSEQ;
00103 }

10.239
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

koi8_r.h

/* $XFree86: xc/lib/X11/lcUniConv/koi8_r.h,v 1.3 2000/11/29 17:40:34 dawes Exp $ */
/*
* KOI8-R
*/
/* Specification: RFC 1489 */
#ifdef NEED_TOWC
static const unsigned short koi8_r_2uni[128] = {
/* 0x80 */
0x2500, 0x2502, 0x250c, 0x2510, 0x2514, 0x2518,
0x252c, 0x2534, 0x253c, 0x2580, 0x2584, 0x2588,
/* 0x90 */
0x2591, 0x2592, 0x2593, 0x2320, 0x25a0, 0x2219,
0x2264, 0x2265, 0x00a0, 0x2321, 0x00b0, 0x00b2,
/* 0xa0 */
0x2550, 0x2551, 0x2552, 0x0451, 0x2553, 0x2554,
0x2557, 0x2558, 0x2559, 0x255a, 0x255b, 0x255c,
/* 0xb0 */
0x255f, 0x2560, 0x2561, 0x0401, 0x2562, 0x2563,
0x2566, 0x2567, 0x2568, 0x2569, 0x256a, 0x256b,
/* 0xc0 */
0x044e, 0x0430, 0x0431, 0x0446, 0x0434, 0x0435,
0x0445, 0x0438, 0x0439, 0x043a, 0x043b, 0x043c,
/* 0xd0 */
0x043f, 0x044f, 0x0440, 0x0441, 0x0442, 0x0443,
0x044c, 0x044b, 0x0437, 0x0448, 0x044d, 0x0449,
/* 0xe0 */
0x042e, 0x0410, 0x0411, 0x0426, 0x0414, 0x0415,
0x0425, 0x0418, 0x0419, 0x041a, 0x041b, 0x041c,
/* 0xf0 */
0x041f, 0x042f, 0x0420, 0x0421, 0x0422, 0x0423,
0x042c, 0x042b, 0x0417, 0x0428, 0x042d, 0x0429,
};

0x251c, 0x2524,
0x258c, 0x2590,
0x221a, 0x2248,
0x00b7, 0x00f7,
0x2555, 0x2556,
0x255d, 0x255e,
0x2564, 0x2565,
0x256c, 0x00a9,
0x0444, 0x0433,
0x043d, 0x043e,
0x0436, 0x0432,
0x0447, 0x044a,
0x0424, 0x0413,
0x041d, 0x041e,
0x0416, 0x0412,
0x0427, 0x042a,

static int
koi8_r_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
unsigned char c = *s;
if (c < 0x80)
*pwc = (ucs4_t) c;
else
*pwc = (ucs4_t) koi8_r_2uni[c-0x80];
return 1;
}
#endif /* NEED_TOWC */
#ifdef NEED_TOMB
static const unsigned char koi8_r_page00[88] = {
0x9a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xbf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x9c, 0x00, 0x9d, 0x00, 0x00, 0x00, 0x00, 0x9e,
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, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x9f,
};
static const unsigned char koi8_r_page04[88] = {
0x00, 0xb3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa,
0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0,
0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe8, 0xe3, 0xfe,
0xfb, 0xfd, 0xff, 0xf9, 0xf8, 0xfc, 0xe0, 0xf1,
0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xd6, 0xda,
0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0,
0xd2, 0xd3, 0xd4, 0xd5, 0xc6, 0xc8, 0xc3, 0xde,
0xdb, 0xdd, 0xdf, 0xd9, 0xd8, 0xdc, 0xc0, 0xd1,
0x00, 0xa3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
static const unsigned char koi8_r_page22[80] = {

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0xa0-0xa7
0xa8-0xaf
0xb0-0xb7
0xb8-0xbf
0xc0-0xc7
0xc8-0xcf
0xd0-0xd7
0xd8-0xdf
0xe0-0xe7
0xe8-0xef
0xf0-0xf7

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

0x00-0x07
0x08-0x0f
0x10-0x17
0x18-0x1f
0x20-0x27
0x28-0x2f
0x30-0x37
0x38-0x3f
0x40-0x47
0x48-0x4f
0x50-0x57

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

Generated by Doxygen


static const unsigned char koi8_r_page23[8] = {
  0x8b, 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 */
  0x97, 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 */
};

static const unsigned char koi8_r_page25[168] = {
  0x80, 0x00, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
  0x00, 0x00, 0x00, 0x00, 0x82, 0x00, 0x00, 0x00, /* 0x08-0x0f */
  0x83, 0x00, 0x00, 0x00, 0x84, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x85, 0x00, 0x00, 0x00, 0x86, 0x00, 0x00, 0x00, /* 0x18-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x87, 0x00, 0x00, 0x00, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, 0x00, /* 0x28-0x2f */
  0x00, 0x00, 0x00, 0x00, 0x89, 0x00, 0x00, 0x00, /* 0x30-0x37 */
  0x00, 0x00, 0x00, 0x00, 0x8a, 0x00, 0x00, 0x00, /* 0x38-0x3f */
  0x00, 0x00, 0x00, 0x00, 0x8b, 0x00, 0x00, 0x00, /* 0x40-0x47 */
  0x8c, 0x00, 0x00, 0x00, 0x8d, 0x00, 0x00, 0x00, /* 0x48-0x4f */
  0x8e, 0x00, 0x00, 0x00, 0x8f, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0xb0, 0x00, 0x00, 0x00, 0xb1, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0xa0, 0x00, 0x00, 0x00, 0xa1, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x94, 0x00, 0x00, 0x00, 0x95, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x8b, 0x00, 0x00, 0x00, 0x8c, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0x90, 0x00, 0x00, 0x00, 0x91, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x92, 0x00, 0x00, 0x00, 0x93, 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 */
  0xb0, 0x00, 0x00, 0x00, 0xb1, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
  0xb2, 0x00, 0x00, 0x00, 0xb3, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
  0xb4, 0x00, 0x00, 0x00, 0xb5, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
  0xb6, 0x00, 0x00, 0x00, 0xb7, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
  0xb8, 0x00, 0x00, 0x00, 0xb9, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
  0xb2, 0x00, 0x00, 0x00, 0xb3, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
  0xb4, 0x00, 0x00, 0x00, 0xb5, 0x00, 0x00, 0x00, /* 0xd8-0xdb */
  0xb6, 0x00, 0x00, 0x00, 0xb7, 0x00, 0x00, 0x00, /* 0xdc-0xdf */
  0xb8, 0x00, 0x00, 0x00, 0xb9, 0x00, 0x00, 0x00, /* 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static int koi8_r_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  }
  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];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

/* KOI8-U */

#error NEED_TOMB

Generated by Doxygen

10.240 koi8_u.h
static int koi8_u_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 < 0x00f8)
        c = koi8_u_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x0498)
        c = koi8_u_page04[wc-0x0400];
    else if (wc >= 0x2218 && wc < 0x2268)
        c = koi8_u_page22[wc-0x2218];
    else if (wc >= 0x2320 && wc < 0x2328)
        c = koi8_u_page23[wc-0x2320];
    else if (wc >= 0x2500 && wc < 0x25a8)
        c = koi8_u_page25[wc-0x2500];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}
unsigned char c1 = (s[0] & 0x7F);
{
    0x71ba, 0x72a7, 0x79a7, 0x7a00, 0x7fb2, 0x8a70,
    0x5e0c, 0x6199, 0x6198, 0x6231, 0x665e, 0x66e6, 0x7199, 0x71b9,
    0x6064, 0x8b4e, 0x9df8, 0x5147, 0x51f6, 0x5308, 0x6d36, 0x80f8,
    0x8af1, 0x8f1d, 0x9ebe, 0x4f11, 0x643a, 0x70cb, 0x7566, 0x8667,
    0x7104, 0x718f, 0x71f0, 0x543c, 0x5589, 0x55c5, 0x5e3f, 0x5f8c, 0x673d, 0x7166, 0x73dd,
    0x6548, 0x6585, 0x66c9, 0x689f, 0x6d8d, 0x6dc6,
    0x6ed1, 0x733e, 0x8c41, 0x95ca, 0x51f0, 0x5e4c, 0x5fa8, 0x604d,
    0x798d, 0x79be, 0x82b1, 0x83ef, 0x8a71, 0x8b41, 0x8ca8, 0x9774,
    0x6c5e, 0x6cd3, 0x6d2a, 0x70d8, 0x7d05, 0x8679, 0x8a0c, 0x9d3b,
    0x6e3e, 0x745a, 0x74e0, 0x7693, 0x795c, 0x7c6a, 0x86e1, 0x8a26,
    0x74f0, 0x76c3, 0x786a, 0x80f3, 0x822a,
    0x745a, 0x74e0, 0x7693, 0x795c, 0x7c6a, 0x86e1, 0x8a26,
    0x74f0, 0x76c3, 0x786a, 0x80f3, 0x822a,
    0x65a8, 0x675f, 0x692d, 0x6a0b, 0x6a7c, 0x6b48, 0x6d2f, 0x703e,
    0x9001, 0x5acc, 0x4fe0, 0x5354,
    0x516e, 0x5f57, 0x60e0, 0x6167, 0x617f, 0x64bb, 0x6667, 0x6841,
    0x6999, 0x6558, 0x589f, 0x65b5, 0x6565, 0x6597, 0x674f, 0x675d,
    0x6841, 0x6877, 0x68f3, 0x6a7c, 0x6a8b, 0x6a9b, 0x6b1f, 0x6b4e,
    0x6ed1, 0x733e, 0x8c41, 0x95ca, 0x51f0, 0x5e4c, 0x5fa8, 0x604d,
    0x74f0, 0x76c3, 0x786a, 0x80f3, 0x822a,
    0x65a8, 0x675f, 0x692d, 0x6a0b, 0x6a7c, 0x6b48, 0x6d2f, 0x703e,
    0x9001, 0x5acc, 0x4fe0, 0x5354,
    0x516e, 0x5f57, 0x60e0, 0x6167, 0x617f, 0x64bb, 0x6667, 0x6841,
    0x6999, 0x6558, 0x589f, 0x65b5, 0x6565, 0x6597, 0x674f, 0x675d,
    0x6841, 0x6877, 0x68f3, 0x6a7c, 0x6a8b, 0x6a9b, 0x6b1f, 0x6b4e,
if (wc != 0xfffd) {
    wc = ksc5601_2uni_page4a[i-3854];
}
if (i < 3760) {
    wc = ksc5601_2uni_page21[i];
} else if (i < 3854) {
    wc = ksc5601_2uni_page4a[i-3854];
}
if (i < 1410) {
    return RET_ILSEQ;
} else if (i < 3854) {
    return RET_TOOFEW(0);
} else if (wc == 0x0fffd) {
    return 2;
}
return RET_ILSEQ;
#endif /* NEED_TOWC */
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 int
ksc5601_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  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 >= 0xf900 && wc < 0xfa10)
      summary = &ksc5601_uni2indx_pagef9[(wc»4)-0xf90];
    else if (wc >= 0xff00 && wc < 0xfff0)
      summary = &ksc5601_uni2indx_pageff[(wc»4)-0xfff];
    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;
      }
    }
  }
  return 0;
};

Generated by Doxygen
```c
03001 } 3.002 return RET_ILSEQ;
03003 } 3.004 return RET_TOOSMALL;
03005 } 3.006 #endif /* NEED_TOMB */
```

## 10.242 mulelao.h

```c
00002 */
00003 */
00004 # MULELAO-1
00005 */
00006 static const unsigned short mulelao_2uni[96] = {
00007     /* 0xa0 */
00008     0x00a0, 0x0e81, 0x0e82, 0xfffd, 0x0e84, 0xfffd, 0x0e87,
00009     0x0e88, 0xfffd, 0x0e8b, 0xfffd, 0x0e8d, 0xfffd,
00010     /* 0xb0 */
00011     0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e94, 0x0e95, 0x0e96, 0x0e97,
00012     0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e99, 0x0e9a, 0x0e9b, 0x0e9c,
00013     0x0e9d, 0x0e9e, 0x0e9f,
00014     /* 0xc0 */
00015     0x0ea1, 0x0ea2, 0x0ea3, 0xfffd, 0x0ea5, 0xfffd, 0x0ea7,
00016     0x0ea8, 0x0ea9, 0x0eab, 0x0eac, 0x0ead, 0x0eae, 0x0eaf,
00017     /* 0xd0 */
00018     0x0eb0, 0x0eb1, 0x0eb2, 0x0eb3, 0x0eb4, 0x0eb5, 0x0eb6, 0x0eb7,
00019     0x0eb8, 0x0eb9, 0xfffd, 0x0ebb, 0x0ec0, 0x0ec1, 0x0ec2, 0x0ec3,
00020     0x0ec4, 0xfffd, 0x0ec5, 0xfffd,
00021     /* 0xe0 */
00022     0x0ec6, 0x0ec7, 0x0ec8, 0x0ec9, 0x0eda, 0x0ecb, 0x0ecc, 0x0ecd,
00023     0x0ece, 0x0ecf, 0xfffd, 0xfffd,
00024     /* 0xf0 */
00025     0x0ed0, 0x0ed1, 0x0ed2, 0x0ed3, 0x0ed4, 0x0ed5, 0x0ed6, 0x0ed7,
00026     0x0ed8, 0x0ed9, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
00027 }
```

```c
00028 static int
00029 mulelao_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00030 {
00031     unsigned char c = *s;
00032     if (c < 0xa0) {
00033         *pwc = (ucs4_t) c;
00034         return 1;
00035     } else {
00036         unsigned short wc = mulelao_2uni[c-0xa0];
00037         if (wc != 0xfffd) {
00038             *pwc = (ucs4_t) wc;
00039             return 1;
00040         } else {
00041             return RET_ILSEQ;
00042         }
00043         return RET_ILSEQ;
00044     }
00045     static const unsigned char mulelao_page0e[96] = {
00046         /* 0x80 */
00047         0x00, 0xa1, 0xa2, 0x00, 0xa4, 0xa5, 0x00, 0xa7, /* 0x80-0x87 */
00048         0x88, 0x89, 0x0a, 0x8c, 0x8d, 0x8e, 0x00, /* 0x88-0x8f */
00049         0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
00050         0x85, 0x86, 0x0a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f, /* 0x89-0x8f */
00051         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x00, 0x07, /* 0x80-0x87 */
00052         0x8c, 0x8d, 0x0a, 0x8b, 0x8c, 0x8d, 0x8e, 0x00, /* 0x88-0x8f */
00053         0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x8f */
00054         0x86, 0x87, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x86-0x8f */
00055         0x8c, 0x8d, 0x0a, 0x8b, 0x8c, 0x8d, 0x8e, 0x00, /* 0x88-0x8f */
00056         0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x8f */
00057         0x8e, 0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x8f */
```
static const unsigned char tatar_cyr_page20[48] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
    0x91, 0x92, 0x93, 0x94, 0x84, 0x00, /* 0x18-0x27 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
    0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
    0x00, 0x8b, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
}

static const unsigned char tatar_cyr_page21[24] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb9, 0x00, /* 0x10-0x17 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
    0x00, 0x00, 0x99, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
}

static const unsigned char tatar_cyr_page22[1] = {
    0xb0, /* 0x16-0x16 */
}

static int tatar_cyr_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 < 0x00bc)
        c = tatar_cyr_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x04ef)
        c = tatar_cyr_page04[wc-0x0400];
    else if (wc >= 0x2010 && wc < 0x203b)
        c = tatar_cyr_page20[wc-0x2010];
    else if (wc == 0x20ac)
        c = 0x88;
    else if (wc >= 0x2110 && wc < 0x2123)
        c = tatar_cyr_page21[wc-0x2110];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

10.244 tcvn.h

static const unsigned short tcvn_2uni_1[32] = {
    0x0000, 0x00da, 0x1ee4, 0x0003, 0x1eea, 0x1eec, 0x1eee, 0x0007,
    0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
    0x0010, 0x1ee8, 0x1ef0, 0x1ef2, 0x1ef6, 0x1ef8, 0x00dd, 0x1ef4,
    0x0018, 0x0019, 0x001a, 0x001b, 0x001c, 0x001d, 0x001e, 0x001f,
}

static const unsigned short tcvn_2uni_2[128] = {
    0x00c0, 0x1ea2, 0x00c3, 0x00c1, 0x1ea0, 0x1eb6, 0x1ec2, 0x00c8,
    0x00c9, 0x0102, 0x00c2, 0x00ca, 0x00c4, 0x01a0, 0x01af, 0x0110,
    0x00e2, 0x00ea, 0x00f4, 0x01a1, 0x01b0, 0x0111, 0x01eb, 0x01e0,
    0x0300, 0x0309, 0x0303, 0x0301, 0x0323, 0x00e0, 0x00e3, 0x00e5,
    0x00eb, 0x01ea, 0x01ec, 0x01e0, 0x01e2, 0x01d9, 0x01ee, 0x0128,
    0x00e9, 0x1eb9, 0x1ee1, 0x1ed2, 0x1eb5, 0x1eaf, 0x1eb4,
    0x00e1, 0x00ea, 0x01e6, 0x01e8, 0x01e4, 0x01ec0, 0x01e7, 0x01e9,
    0x01ea9, 0x01eb, 0x01ed, 0x00e85, 0x01ec2, 0x01eb, 0x01ed,
    0x00eb, 0x01ea, 0x01ec, 0x01e1, 0x01e3, 0x01f5, 0x01ed0,
    0x00eb, 0x01ea, 0x01ec, 0x01e3, 0x01f5, 0x01ed0,
    0x00eb, 0x01ea, 0x01ec, 0x01e3, 0x01f5, 0x01ed0,
10.244 tcvn.h

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
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126

2221

};
static int
tcvn_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
unsigned char c = *s;
if (c < 0x20)
*pwc = (ucs4_t) tcvn_2uni_1[c];
else if (c < 0x80)
*pwc = (ucs4_t) c;
else
*pwc = (ucs4_t) tcvn_2uni_2[c-0x80];
return 1;
}
static const unsigned char tcvn_page00[96+184] = {
0xa0, 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, 0x00, 0x00, /*
0x80, 0x83, 0xa2, 0x82, 0x00, 0x00, 0x00, 0x00, /*
0x87, 0x8a, 0xa3, 0x00, 0x8d, 0x90, 0x00, 0x00, /*
0x00, 0x00, 0x92, 0x95, 0xa4, 0x94, 0x00, 0x00, /*
0x00, 0x9d, 0x01, 0x00, 0x00, 0x16, 0x00, 0x00, /*
0xb5, 0xb8, 0xa9, 0xb7, 0x00, 0x00, 0x00, 0x00, /*
0xcc, 0xd0, 0xaa, 0x00, 0xd7, 0xdd, 0x00, 0x00, /*
0x00, 0x00, 0xdf, 0xe3, 0xab, 0xe2, 0x00, 0x00, /*
0x00, 0xef, 0xf3, 0x00, 0x00, 0xfd, 0x00, 0x00, /*
/* 0x0100 */
0x00, 0x00, 0xa1, 0xa8, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0xa7, 0xae, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x8f, 0xdc, 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, 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, 0x00, 0x00, 0x00, 0x00, /*
0x9f, 0xf2, 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, 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, /*
0xa5, 0xac, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xa6, /*
0xad, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /*
};
static const unsigned char tcvn_page03[40] = {
0xb0, 0xb3, 0x00, 0xb2, 0x00, 0x00, 0x00, 0x00, /*
0x00, 0xb1, 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, 0xb4, 0x00, 0x00, 0x00, 0x00, /*
};
static const unsigned char tcvn_page1e[96] = {
0x84, 0xb9, 0x81, 0xb6, 0xc4, 0xca, 0xc1, 0xc7, /*
0xc2, 0xc8, 0xc3, 0xc9, 0x86, 0xcb, 0xc0, 0xbe, /*
0xaf, 0xbb, 0xba, 0xbc, 0xbf, 0xbd, 0x85, 0xc6, /*
0x8b, 0xd1, 0x88, 0xce, 0x89, 0xcf, 0xda, 0xd5, /*
0xc5, 0xd2, 0xcd, 0xd3, 0xd9, 0xd4, 0x8c, 0xd6, /*
0x8e, 0xd8, 0x91, 0xde, 0x96, 0xe4, 0x93, 0xe1, /*
0xff, 0xe8, 0xdb, 0xe5, 0xe0, 0xe6, 0xf0, 0xe7, /*
0x97, 0xe9, 0x9b, 0xed, 0x98, 0xea, 0x99, 0xeb, /*
0x9a, 0xec, 0x9c, 0xee, 0x02, 0xf4, 0x9e, 0xf1, /*
0x11, 0xf8, 0x04, 0xf5, 0x05, 0xf6, 0x06, 0xf7, /*
0x12, 0xf9, 0x13, 0xfa, 0x17, 0xfe, 0x14, 0xfb, /*
0x15, 0xfc, 0x00, 0x00, 0x00, 0x00, 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
0x80-0x87
0x88-0x8f
0x90-0x97
0x98-0x9f
0xa0-0xa7
0xa8-0xaf
0xb0-0xb7

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

0x00-0x07
0x08-0x0f
0x10-0x17
0x18-0x1f
0x20-0x27

*/
*/
*/
*/
*/

0xa0-0xa7
0xa8-0xaf
0xb0-0xb7
0xb8-0xbf
0xc0-0xc7
0xc8-0xcf
0xd0-0xd7
0xd8-0xdf
0xe0-0xe7
0xe8-0xef
0xf0-0xf7
0xf8-0xff

*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/
*/

static int
tcvn_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
unsigned char c = 0;
if (wc < 0x0080 && (wc >= 0x0020 || (0x00fe0076 & (1 « wc)) == 0)) {
*r = wc;
return 1;
}
else if (wc >= 0x00a0 && wc < 0x01b8)
c = tcvn_page00[wc-0x00a0];
else if (wc >= 0x0300 && wc < 0x0328)
c = tcvn_page03[wc-0x0300];

Generated by Doxygen


```c
if (wc >= 0x1ea0 && wc < 0x1f00)
    c = tcvn_page1e[wc-0x1ea0];
if (c != 0) {
    *r = c;
    return 1;
}
return RET_ILSEQ;
```
```c
00075    return 1;
00076 }
00077 return RET_ILSEQ;
00078 }

10.246 ucs2be.h

00001 /*
00002 * UCS-2BE = UCS-2 big endian
00003 */
00005
00006 static int
00007 ucs2be_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00008 {
00009     if (n >= 2) {
00010         if (s[0] >= 0xd8 && s[0] < 0xe0) {
00011             return RET_ILSEQ;
00012         } else {
00013             *pwc = (s[0] « 8) + s[1];
00014             return 2;
00015         }
00016     }
00017     return RET_TOOFEW(0);
00018 }
00019
00020 static int
00021 ucs2be_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
00022 {
00023     if (wc < 0x10000 && !(wc >= 0xd800 && wc < 0xe000)) {
00024         if (n >= 2) {
00025             r[0] = (unsigned char) (wc » 8);
00026             r[1] = (unsigned char) wc;
00027             return 2;
00028         } else
00029             return RET_TOOSMALL;
00030     }
00031     return RET_ILSEQ;
00032 }

10.247 utf8.h

00002
00003 /*
00004 * UTF-8
00005 */
00006 /* Specification: RFC 2279 */
00007
00008 static int
00009 utf8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
00010 {
00011     unsigned char c = s[0];
00012     if (c < 0x80) {
00013         *pwc = c;
00014         return 1;
00015     } else if (c < 0xc2) {
00016         return RET_ILSEQ;
00017     } else if (c < 0xe0) {
00018         return RET_TOOFEW(0);
00019     } else if (c < 0xe0) {
00020         if (n > 2) {
00021             return RET_TOOFEW(0);
00022             if (((s[1] ^ 0x80) ^ 0x80) < 0x40)
00023                 return RET_ILSEQ;
00024             if (((s[1] ^ 0x80) < 0x80) || (s[2] ^ 0x80) < 0x80)
00025                 return 2;
00026             } else
00027                 return RET_ILSEQ;
00028         } else if (n < 3) {
00029             return RET_TOOFEW(0);
00030             if (((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x80))
00031                 return RET_ILSEQ;
00032             if (((s[1] ^ 0x80) < 0x80) || (s[2] ^ 0x80) < 0x80)
00033                 return 3;
00034             if ((c < 0xf0) {
00035                 return RET_ILSEQ;
00036             }
00037             if (n < 4) {
00038                 return RET_TOOFEW(0);
00039                 if (((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x80)
```
00041         && (s[3] ^ 0x80) < 0x40
00042         && (c >= 0xf1 || s[1] >= 0x90))
00043     return RET_ILSEQ;
00044 *pwc = ((ucs4_t) (c & 0x07) « 18)
00045         | ((ucs4_t) (s[1] ^ 0x80) « 12)
00046         | ((ucs4_t) (s[2] ^ 0x80) « 6)
00047         | (ucs4_t) (s[3] ^ 0x80);
00048     return 4;
00049 } else if (c < 0xfc) {
00050     if (n < 5)
00051         return RET_TOOFEW();
00052     if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40
00053         && (s[3] ^ 0x80) < 0x40 && (s[4] ^ 0x80) < 0x40
00054         && (c >= 0xf9 || s[1] >= 0x88))
00055     return RET_ILSEQ;
00056     *pwc = ((ucs4_t) (c & 0x03) « 24)
00057         | ((ucs4_t) (s[1] ^ 0x80) « 18)
00058         | ((ucs4_t) (s[2] ^ 0x80) « 12)
00059         | ((ucs4_t) (s[3] ^ 0x80) « 6)
00060         | (ucs4_t) (s[4] ^ 0x80);
00061     return 5;
00062 } else if (c < 0xfe) {
00063     if (n < 6)
00064         return RET_TOOFEW();
00065     if (!((s[1] ^ 0x80) < 0x40 && (s[2] ^ 0x80) < 0x40
00066         && (s[3] ^ 0x80) < 0x40 && (s[4] ^ 0x80) < 0x40
00067         && (s[5] ^ 0x80) < 0x40
00068         && (c >= 0xfd || s[1] >= 0x84))
00069     return RET_ILSEQ;
00070     *pwc = ((ucs4_t) (c & 0x01) « 30)
00071         | ((ucs4_t) (s[1] ^ 0x80) « 24)
00072         | ((ucs4_t) (s[2] ^ 0x80) « 18)
00073         | ((ucs4_t) (s[3] ^ 0x80) « 12)
00074         | ((ucs4_t) (s[4] ^ 0x80) « 6)
00075         | (ucs4_t) (s[5] ^ 0x80);
00076     return 6;
00077 } else
00078     return RET_ILSEQ;
00079 }
00080
00081 static int
00082 utf8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) /* n == 0 is acceptable */
00083 {
00084     int count;
00085     if (wc < 0x80)
00086         count = 1;
00087     else if (wc < 0x800)
00088         count = 2;
00089     else if (wc < 0x10000)
00090         count = 3;
00091     else if (wc < 0x20000)
00092         count = 4;
00093     else if (wc < 0x400000)
00094         count = 5;
00095     else if (wc <= 0x7fffffff)
00096         count = 6;
00097     else
00098         return RET_ILSEQ;
00099     if (n < count)
00100         return RET_TOOSMALL;
00101 switch (count) { /* note: code falls through cases! */
00102     case 6: r[5] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x40000000;
00103     case 5: r[4] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x20000000;
00104     case 4: r[3] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x10000000;
00105     case 3: r[2] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x80000000;
00106     case 2: r[1] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0xc0;
00107     case 1: r[0] = wc;
00108     }
00109 return count;
00110 }

10.248 viscii.h

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, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007,
00012 0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
static const unsigned short viscii_2uni_2[128] = {
    0x0018, 0x0019, 0x001a, 0x001b, 0x001c, 0x001d, 0x001e, 0x001f,
    0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027,
    0x0028, 0x0029, 0x002a, 0x002b, 0x002c, 0x002d, 0x002e, 0x002f,
    0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037,
    0x0038, 0x0039, 0x003a, 0x003b, 0x003c, 0x003d, 0x003e, 0x003f,
    0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047,
    0x0048, 0x0049, 0x004a, 0x004b, 0x004c, 0x004d, 0x004e, 0x004f,
    0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057,
    0x0058, 0x0059, 0x005a, 0x005b, 0x005c, 0x005d, 0x005e, 0x005f,
    0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067,
    0x0068, 0x0069, 0x006a, 0x006b, 0x006c, 0x006d, 0x006e, 0x006f,
    0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077,
    0x0078, 0x0079, 0x007a, 0x007b, 0x007c, 0x007d, 0x007e, 0x007f,
    0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087,
    0x0088, 0x0089, 0x008a, 0x008b, 0x008c, 0x008d, 0x008e, 0x008f,
    0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097,
    0x0098, 0x0099, 0x009a, 0x009b, 0x009c, 0x009d, 0x009e, 0x009f,
    0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
    0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
    0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
    0x00b8, 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
    0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
    0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
    0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
    0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
    0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
    0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
    0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
    0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00ff,
};
```c
static int
viscki_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 < 0x01b8)
        c = viscki_page00[wc-0x00c0];
    else if (wc >= 0x1ea0 && wc < 0x1f00)
        c = viscki_page1e[wc-0x1ea0];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}
```

10.249  
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!
 */

/*
 * 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
 * the cursor by 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.
 * In all these cases, there is no ambiguity about which width a
 * particular character shall use. For characters in the East Asian Ambiguous (A)
 * 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.
 */

http://www.unicode.org/unicode/reports/tr11/
```
00009 * 00010 * Markus Kuhn -- 2007-05-26 (Unicode 5.0) 00011 * 00012 * Permission to use, copy, modify, and distribute this software 00013 * for any purpose and without fee is hereby granted. The author 00014 * disclaims all warranties with regard to this software. 00015 * 00016 * Latest version: http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c 00017 */
00018
00019 /* FLTK - avoid possible problems on systems with 32-bit wchar_t. 00020 * Don't include wchar.h, and change wchar_t to unsigned int. 00021 * Can we guarantee sizeof(unsigned int) >= 4 ? 00022 */
00023
00024 #if 0
00025 #include <wchar.h>
00026 #endif
00027
00028 struct interval { 00029  unsigned int first; 00030  unsigned int last; 00031 };
00032
00033 /* auxiliary function for binary search in interval table */
00034
00035 static int bisearch(wchar_t ucs, const struct interval *table, int max) { 00036
00037     int min = 0;
00038     int mid;
00039
00040     if (ucs < table[0].first || ucs > table[max].last)
00041         return 0;
00042     while (max >= min) {
00043         mid = (min + max) / 2;
00044         if (ucs > table[mid].last)
00045             min = mid + 1;
00046         else if (ucs < table[mid].first)
00047             max = mid - 1;
00048         else
00049             return 1;
00050     } 00051     return 0;
00052 }
00053
00054 int mk_wcwidth(wchar_t ucs) 00055 {
00056     int min = 0;
00057     int mid;
00058
00059     if (ucs < table[0].first || ucs > table[max].last)
00060         return 0;
00061     while (max >= min) {
00062         mid = (min + max) / 2;
00063         if (ucs > table[mid].last)
00064             min = mid + 1;
00065         else if (ucs < table[mid].first)
00066             max = mid - 1;
00067         else
00068             return 1;
00069     }
00070     return 0;
00071 }
00072
00073 /* The following two functions define the column width of an ISO 10646 00074 * character as follows: 00075 * 00076 * - The null character (U+0000) has a column width of 0. 00077 * 00078 * - Other C0/C1 control characters and DEL will lead to a return 00079 * value of -1. 00080 * 00081 * - Non-spacing and enclosing combining characters (general 00082 * category code Mn or Me in the Unicode database) have a 00083 * column width of 0. 00084 * 00085 * - SOFT HYPHEN (U+00AD) has a column width of 1. 00086 * 00087 * - Other format characters (general category code Cf in the Unicode 00088 * database) and ZERO WIDTH SPACE (U+200B) have a column width of 0. 00089 * 00090 * - Hanul Jamo medial vowels and final consonants (U+1160-U+11FF) 00091 * have a column width of 0. 00092 * 00093 * - Spacing characters in the East Asian Wide (W) or East Asian 00094 * Full-width (F) category as defined in Unicode Technical 00095 * Report #11 have a column width of 2. 00096 * 00097 * - All remaining characters (including all printable 00098 * ISO 8859-1 and WGL4 characters, Unicode control characters, 00099 * etc.) have a column width of 1. 00100 */
00101
00102 int mk_wcwidth(unsigned int ucs) 00103 {
00104     int min = 0;
00105     int mid;
00106
00107     if (ucs < table[0].first || ucs > table[max].last)
00108         return 0;
00109     while (max >= min) {
00110         mid = (min + max) / 2;
00111         if (ucs > table[mid].last)
00112             min = mid + 1;
00113         else if (ucs < table[mid].first)
00114             max = mid - 1;
00115         else
00116             return 1;
00117     }
00118     return 0;
00119 }
00120
00121 /* FLTK: was 00122 */
00123
00124 /* FLTK: was 00125 */
00126
00127 int mk_wcwidth(wchar_t ucs) 00128 {
00129     int min = 0;
00130     int mid;
00131
00132     if (ucs < table[0].first || ucs > table[max].last)
00133         return 0;
00134     while (max >= min) {
00135         mid = (min + max) / 2;
00136         if (ucs > table[mid].last)
00137             min = mid + 1;
00138         else if (ucs < table[mid].first)
00139             max = mid - 1;
00140         else
00141             return 1;
00142     }
00143     return 0;
00144 }
00145
00146 Generated by Doxygen
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 }, { 0x0670, 0x0670 },
    { 0x070F, 0x070F }, { 0x0711, 0x0711 }, { 0x0730, 0x074A },
    { 0x07A6, 0x07B0 }, { 0x07EB, 0x07F3 }, { 0x0901, 0x0902 },
    { 0x093C, 0x093C }, { 0x0941, 0x0948 }, { 0x094D, 0x094D },
    { 0x0951, 0x0954 }, { 0x0962, 0x0963 }, { 0x0981, 0x0981 },
    { 0x09BC, 0x09BC }, { 0x09C1, 0x09C4 }, { 0x09CD, 0x09CD },
    { 0x09E2, 0x09E3 }, { 0x0A01, 0x0A02 }, { 0x0A3C, 0x0A3C },
    { 0x0A41, 0x0A42 }, { 0x0A47, 0x0A48 }, { 0x0A4B, 0x0A4D },
    { 0x0A70, 0x0A71 }, { 0x0A81, 0x0A82 }, { 0x0ABC, 0x0ABC },
    { 0x0AC1, 0x0AC5 }, { 0x0AC7, 0x0AC8 }, { 0x0ACD, 0x0ACD },
    { 0x0AE2, 0x0AE3 }, { 0x0B01, 0x0B01 }, { 0x0B3C, 0x0B3C },
    { 0x0B3F, 0x0B3F }, { 0x0B41, 0x0B41 }, { 0x0B82, 0x0B82 },
    { 0x0BC0, 0x0BC0 }, { 0x0BCD, 0x0BCD }, { 0x0C3E, 0x0C40 },
    { 0x0C46, 0x0C48 }, { 0x0CE2, 0x0CE3 }, { 0x0D41, 0x0D43 },
    { 0x0E31, 0x0E31 }, { 0x0F18, 0x0F19 }, { 0x0F35, 0x0F35 },
    { 0x1712, 0x1714 }, { 0x1732, 0x1734 }, { 0x1752, 0x1753 },
    { 0x1772, 0x1773 }, { 0x17B4, 0x17B5 }, { 0x17B7, 0x17BD },
    { 0x1920, 0x1922 }, { 0x1927, 0x1928 }, { 0x1932, 0x1932 },
    { 0x1A17, 0x1A18 }, { 0x1B00, 0x1B03 }, { 0x1B34, 0x1B34 },
    { 0x1B36, 0x1B3A }, { 0x1B3C, 0x1B3C }, { 0x1B42, 0x1B42 },
    { 0x1D242, 0x1D244 }, { 0x1E0001, 0x1E0001 }, { 0x1E0020, 0x1E007F },
    { 0x1E0010, 0x1E01BF }  
};
int mk_wcwidth(const wchar_t *pwcs, size_t n)
int mk_wcwidth(const unsigned int *pwcs, size_t n)

for (; *pwcs && n-- > 0; pwcs++)
    if ((w = mk_wcwidth(*pwcs)) < 0)
        return -1;
else
    width += w;
return width;

The following functions are the same as mk_wcwidth() and
mk_wcswidth(), except that spacing characters in the East Asian
Ambiguous (A) category as defined in Unicode Technical Report #11
have a column width of 2. This variant might be useful for users of
CJK legacy encodings who want to migrate to UCS without changing
the traditional terminal character-width behaviour. It is not
otherwise recommended for general use.

Generated by Doxygen
/* 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_wcwidth_cjk(const wchar_t *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 */
#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"

/**
 * \brief \ConvGen
 *
 * \param s \in char *
 * \param ucs \in unsigned int
 * \param enc \in int
 *
 * \ConvGen
 */
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;
                return 1;
            }
            else if (ucs > 0x0001) {
                s[0] = 0;
                s[1] = (char) ucs & 0xFF;
                return 1;
            }
            break;
        }
    case 2: /* iso8859-2 */
        if (ucs <= 0x00a0) {
            if (ucs >= 0x00a0) {
                s[0] = 0;
                s[1] = iso8859_2_page00[ucs-0x00a0];
                if (s[1]) return 2;
            }
            else if (ucs < 0x02e0) {
                if (ucs >= 0x02c0) {
                    s[0] = 0;
                    s[1] = iso8859_2_page02[ucs-0x02c0];
                    if (s[1]) return 2;
                }
            }
            break;
        }
    case 3: /* iso8859-3 */
        if (iso8859_3_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 3;
        }
    case 4: /* iso8859-4 */
        if (iso8859_4_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 4;
        }
    case 5: /* iso8859-5 */
        if (iso8859_5_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 5;
        }
    case 6: /* iso8859-6 */
        if (iso8859_6_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 6;
        }
    case 7: /* iso8859-7 */
        if (iso8859_7_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 7;
        }
    case 8: /* iso8859-8 */
        if (iso8859_8_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 8;
        }
    case 9: /* iso8859-9 */
        if (iso8859_9_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 9;
        }
    case 10: /* iso8859-10 */
        if (iso8859_10_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
            return 10;
        }
    case 25: /* iso8859-11 */
    }
if (iso8859_11_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 25;
}
break;
case 11: /* iso8859-13 */
if (iso8859_13_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 11;
}
break;
case 12: /* iso8859-14 */
if (iso8859_14_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 12;
}
break;
case 13: /* iso8859-15 */
if (iso8859_15_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 13;
}
break;
case 14: /* koi8-r */
if (koi8_r_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 14;
}
break;
case 15: /* big5 */
if (big5_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 15;
}
break;
case 16: /* ksc5601.1987-0 */
if (ksc5601_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 16;
}
break;
case 17: /* gb2312.1980-0 */
if (gb2312_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 17;
}
break;
case 18: /* jisx0201.1976-0 */
if (jisx0201_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 18;
}
break;
case 19: /* jisx0208.1983-0 */
if (jisx0208_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 19;
}
break;
case 20: /* jisx0212.1990-0 */
if (jisx0212_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
    return 20;
}
break;
case 21: /* symbol */
if (ucs <= 0x00DF7) {
    if (ucs >= 0x0020) {
        s[0] = 0;
        s[1] = unicode_to_symbol_1b_0020[ucs - 0x0020];
        if (s[1]) return 21;
    } else if (ucs <= 0x03D6) {
        if (ucs >= 0x0391) {
            s[0] = 0;
            s[1] = unicode_to_symbol_1b_0391[ucs - 0x0391];
            if (s[1]) return 21;
        } else if (ucs <= 0x332A) {
            if (ucs >= 0x2022) {
                s[0] = 0;
                s[1] = unicode_to_symbol_1b_2022[ucs - 0x2022];
                if (s[1]) return 21;
            } else if (ucs <= 0x25CA) {
                s[0] = 0;
                s[1] = unicode_to_symbol_1b_25CA[ucs - 0x25CA];
                if (s[1]) return 21;
            } else if (ucs <= 0x2666) {
                if (ucs >= 0x2660) {
00222     s[0] = 0;
00223     s[1] = unicode_to_symbol_1b_2660[ucs - 0x2660];
00224     if (s[1]) return 21;
00225 
00226 }
00227 else if (ucs <= 0xF6DB) {
00228     if (ucs >= 0xF6D9) {
00229         s[0] = 0;
00230         s[1] = unicode_to_symbol_1b_F6D9[ucs - 0xF6D9];
00231         if (s[1]) return 21;
00232     } else if (ucs <= 0xF8FE) {
00233         if (ucs >= 0xF8E5) {
00234             s[0] = 0;
00235             s[1] = unicode_to_symbol_1b_F8E5[ucs - 0xF8E5];
00236             if (s[1]) return 21;
00237         }
00238     }
00239     break;
00240 case 22: /* dingbats */
00241     if (ucs <= 0x00A0) {
00242         if (ucs >= 0x0020) {
00243             s[0] = 0;
00244             s[1] = unicode_to_dingbats_1b_0020[ucs - 0x0020];
00245             if (s[1]) return 22;
00246         }
00247     } else if (ucs <= 0x2195) {
00248         if (ucs >= 0x2192) {
00249             s[0] = 0;
00250             s[1] = unicode_to_dingbats_1b_2192[ucs - 0x2192];
00251             if (s[1]) return 22;
00252     }
00253     } else if (ucs <= 0x2466) {
00254         if (ucs >= 0x2460) {
00255             s[0] = 0;
00256             s[1] = unicode_to_dingbats_1b_2460[ucs - 0x2460];
00257             if (s[1]) return 22;
00258     }
00259     } else if (ucs <= 0x2666) {
00260         if (ucs >= 0x25A0) {
00261             s[0] = 0;
00262             s[1] = unicode_to_dingbats_1b_25A0[ucs - 0x25A0];
00263             if (s[1]) return 22;
00264     }
00265     }
00266     } else if (ucs <= 0x27BE) {
00267         if (ucs >= 0x2701) {
00268             s[0] = 0;
00269             s[1] = unicode_to_dingbats_1b_2701[ucs - 0x2701];
00270             if (s[1]) return 22;
00271     }
00272     } else if (ucs <= 0xF8E4) {
00273         if (ucs >= 0xF8D7) {
00274             s[0] = 0;
00275             s[1] = unicode_to_dingbats_1b_F8D7[ucs - 0xF8D7];
00276             if (s[1]) return 22;
00277     }
00278     break;
00279 case 23: /* koi8-u */
00280     if (koi8_u_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00281         return 23;
00282     }
00283     break;
00284 case 24: /* microsoft-cp1251 */
00285     if (cp1251_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00286         return 24;
00287     }
00288     break;
00289 case 26: /* gbk/cp936ext */
00290     if (cp936ext_wctomb(NULL, (unsigned char*)s, ucs, 2) > 0) {
00291         return 26;
00292     }
00293     break;
00294     default:
00295     break;
00296 }
00297 return -1;
00298 }
00299
00300 */const*/
00301 static int encoding_number(const char *enc) {
00302     if (!enc || !strcmp(enc, "iso10646-1")) {
00303         return 0;
00304     } else if (!strcmp(enc, "iso8859-1")) {
00305         return 1;
00306     } else if (!strcmp(enc, "iso8859-2")) {
00307         return 2;
00308     } else if (!strcmp(enc, "iso8859-3")) {
00309    } else if (!strcmp(enc, "iso8859-9")) {
00310    return 9;
00311 } else if (!strcmp(enc, "iso8859-10")) {
00312    return 10;
00313 } else if (!strcmp(enc, "iso8859-13")) {
00314    return 11;
00315 } else if (!strcmp(enc, "iso8859-14")) {
00316    return 12;
00317 } else if (!strcmp(enc, "iso8859-15")) {
00318    return 13;
00319 } else if (!strcmp(enc, "koi8-r")) {
00320    return 14;
00321 } else if (!strcmp(enc, "big5-0") || !strcmp(enc, "big5-eten-0") ||
00322    !strcmp(enc, "big5p-0")) {
00323    return 15;
00324 } else if (!strcmp(enc, "jisx0201.1976-0")) {
00325    return 18;
00326 } else if (!strcmp(enc, "jisx0208.1983-0") || !strcmp(enc, "jisx0208.1990-0")
00327    || !strcmp(enc, "jisx0208.1978-0")) {
00328    return 19;
00329 } else if (!strcmp(enc, "jisx0212.1990-0")) {
00330    return 20;
00331 } else if (!strcmp(enc, "symbol")) {
00332    return 21;
00333 } else if (!strcmp(enc, "cp1251")) {
00334    return 25;
00335 } else if (!strcmp(enc, "cp936") || !strcmp(enc, "gbk")) {
00336    return -1;
00337 }

10.251 utf8Utils.c

00001 /* "$Id: "$ */
00002 /*
00003 * Author: Jean-Marc Lienher ( http://oksid.ch )
00004 * Copyright 2000-2003 by O'ksi'D.
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 http://www.fltk.org/COPYING.php
00011 *
00012 * Please report all bugs and problems on the following page:
00013 *
00014 http://www.fltk.org/str.php
00015 */
00016
00017 */
00018 * Unicode to UTF-8 conversion functions.
00019 */
#if !_defined(WIN32) && !_defined(__APPLE__)
#include "../Xutf8.h"

/***
 NOTE : all functions are LIMITED to 24 bits Unicode values !!!
***/

int
XConvertUtf8ToUcs(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
          && (buf[1] & 0xC0) == 0x80
          && (buf[2] & 0xC0) == 0x80
          && (buf[3] & 0xC0) == 0x80
          && (buf[4] & 0xC0) == 0x80) {
            /* 0x00200000 - 0x03FFFFFF */
            *ucs = ((buf[0] & ~0xF8) « 24)
            + ((buf[1] & ~0x80) « 18)
            + ((buf[2] & ~0x80) « 12)
            + ((buf[3] & ~0x80) « 6)
            + (buf[4] & ~0x80);
            if (*ucs > 0x001FFFFF && *ucs < 0x01000000) return 5;
          }
        } else if (len > 3
        && (buf[1] & 0xC0) == 0x80
        && (buf[2] & 0xC0) == 0x80
        && (buf[3] & 0xC0) == 0x80) {
          /* 0x00010000 - 0x001FFFFF */
          *ucs = ((buf[0] & ~0xE0) « 12)
          + ((buf[1] & ~0x80) « 6)
          + (buf[2] & ~0x80);
          if (*ucs > 0x0000FFFF) return 4;
        }
      } else if (len > 2
      && (buf[1] & 0xC0) == 0x80
      && (buf[2] & 0xC0) == 0x80) {
        /* 0x00000800 - 0x0000FFFF */
        *ucs = ((buf[0] & ~0xF0) « 18) +
        ((buf[1] & ~0x80) « 12) +
        ((buf[2] & ~0x80) « 6) +
        (buf[3] & ~0x80);
        if (*ucs < 0x00000000) return 3;
      }
    } else if (len > 1 && (buf[1] & 0xC0) == 0x80) {
      /* 0x00000080 - 0x000000FF */
      *ucs = ((buf[0] & ~0xE0) « 12) +
      ((buf[1] & ~0x80) « 6) +
      (buf[2] & ~0x80);
      if (*ucs > 0x000000FF) return 2;
    }
  }
}

/* 0x00000000 - 0x0000007F */
*ucs = buf[0];
return 1;
}

/* bad utf-8 string */
return -1;

} /* bad utf-8 string */

int
XConvertUcsToUtf8(unsigned int ucs,
char *buf) {
  return 1;
}

if (ucs < 0x0000080) {
  buf[0] = ucs;
}
return 1;
} else if (ucs < 0x000800) {
    buf[0] = 0xC0 | (ucs >> 6);
    return 2;
} else if (ucs < 0x010000) {
    buf[0] = 0xE0 | (ucs >> 12);
    buf[1] = 0x80 | ((ucs >> 6) & 0x3F);
    return 3;
} else if (ucs < 0x020000) {
    buf[0] = 0xF0 | (ucs >> 18);
    buf[1] = 0x80 | ((ucs >> 12) & 0x3F);
    buf[2] = 0x80 | ((ucs >> 6) & 0x3F);
    return 4;
} else if (ucs < 0x03000000) {
    buf[0] = 0xF8 | (ucs >> 24);
    buf[1] = 0x80 | ((ucs >> 18) & 0x3F);
    buf[2] = 0x80 | ((ucs >> 12) & 0x3F);
    buf[3] = 0x80 | ((ucs >> 6) & 0x3F);
    buf[4] = 0x80 | (ucs & 0x3F);
    return 5;
} else if (ucs < 0x0300000000) {
    buf[0] = '?';
    return -1;
} return 1;

int XUtf8CharByteLen(const unsigned char *buf, int len) {
    unsigned int ucs;
    return XConvertUtf8ToUcs(buf, len, &ucs);
}

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 += cl;
        i += cl;
    }
    return nbc;
}

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 - 0x03FFFFFFF */
                            } else if (len > 4) {
                                /* 0x00020000 - 0x00010000 */
                                *ucs = ((buf[0] & -0x10) << 24) +
                                ((buf[1] & -0x80) << 18) +
                                ((buf[2] & -0x80) << 12) +
                                ((buf[3] & -0x80) << 6) +
                                (buf[4] & -0x80);
                                return 5;
                            } else if (len > 3) {
                                /* 0x00010000 - 0x0001FFFF */
                            } else {
                                /* 0x00001000 */
                            }
                        } else if (len > 2) {
                            /* 0x00000100 */
                        } else if (len > 1) {
                            /* 0x00000010 */
                        } else if (len > 0) {
                            /* 0x00000001 */
                        } else {
                            return -1;
                        }
                    } else if (len > 1) {
                        /* 0x00000010 */
                    } else if (len > 0) {
                        /* 0x00000001 */
                    } else {
                        return -1;
                    }
                } else if (len > 1) {
                    /* 0x00000001 */
                } else if (len > 0) {
                    /* 0x00000000 */
                } else {
                    return -1;
                }
            } else if (len > 1) {
                /* 0x00000000 */
            } else if (len > 0) {
                /* 0x00000000 */
            } else {
                return -1;
            }
        } else if (len > 1) {
            /* 0x00000000 */
        } else if (len > 0) {
            /* 0x00000000 */
        } else {
            return -1;
        }
    } else if (len > 0) {
        /* 0x00000000 */
    } else {
        return -1;
    }
    return -1;
}
*ucs = ((buf[0] & ~0xF0) << 18) +
  ((buf[1] & ~0x80) << 12) +
  ((buf[2] & ~0x80) << 6) +
  (buf[3] & ~0x80);
  return 4;
else if (len > 2) {
  /* 0x00000800 - 0x0000FFFF */
  *ucs = ((buf[0] & ~0xE0) << 12) +
  ((buf[1] & ~0x80) << 6) +
  (buf[2] & ~0x80);
  return 3;
else if (len > 1) {
  /* 0x00000080 - 0x000007FF */
  *ucs = ((buf[0] & ~0xC0) << 6) +
  (buf[1] & ~0x80);
  return 2;
else if (len > 0) {
  /* 0x00000000 - 0x0000007F */
  *ucs = buf[0];
  return 1;
  }
  *ucs = (unsigned int) '?'; /* bad utf-8 string */
  return -1;
  endif /* X11 only */
  * End of "$Id: "$.
*/
10.252 Ximint.h
10.253 Xlibint.h

Generated by Doxygen
Index

_\_FL\_DIAMOND\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_DIAMOND\_UP\_BOX
Enumerations.H, 1608
_\_FL\_EMBOSS\_LABEL
Enumerations.H, 1613
_\_FL\_ENGRAVED\_LABEL
Enumerations.H, 1613
_\_FL\_GLEAM\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GLEAM\_DOWN\_FRAME
Enumerations.H, 1608
_\_FL\_GLEAM\_ROUND\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GLEAM\_ROUND\_UP\_BOX
Enumerations.H, 1608
_\_FL\_GLEAM\_THIN\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GLEAM\_THIN\_UP\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_DOWN\_FRAME
Enumerations.H, 1608
_\_FL\_GTK\_ROUND\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_ROUND\_UP\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_THIN\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_THIN\_DOWN\_FRAME
Enumerations.H, 1608
_\_FL\_GTK\_UP\_BOX
Enumerations.H, 1608
_\_FL\_GTK\_UP\_FRAME
Enumerations.H, 1608
_\_FL\_ICON\_LABEL
Enumerations.H, 1613
_\_FL\_IMAGE\_LABEL
Enumerations.H, 1613
_\_FL\_MULTI\_LABEL
Enumerations.H, 1613
_\_FL\_OFLAT\_BOX
Enumerations.H, 1613
_\_FL\_OSHADOW\_BOX
Enumerations.H, 1608
_\_FL\_OVAL\_BOX
Enumerations.H, 1608
_\_FL\_OVAL\_FRAME
Enumerations.H, 1608
_\_FL\_PLASTIC\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_DOWN\_FRAME
Enumerations.H, 1608
_\_FL\_PLASTIC\_ROUND\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_ROUND\_UP\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_THIN\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_THIN\_UP\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_UP\_BOX
Enumerations.H, 1608
_\_FL\_PLASTIC\_UP\_FRAME
Enumerations.H, 1608
_\_FL\_RFLAT\_BOX
Enumerations.H, 1608
_\_FL\_ROUNDED\_BOX
Enumerations.H, 1607
_\_FL\_ROUNDED\_FRAME
Enumerations.H, 1607
_\_FL\_ROUND\_DOWN\_BOX
Enumerations.H, 1608
_\_FL\_ROUND\_UP\_BOX
Enumerations.H, 1608
_\_FL\_RSHADOW\_BOX
Enumerations.H, 1608
_\_FL\_SHADOW\_BOX
Enumerations.H, 1607
_\_FL\_SHADOW\_FRAME
Enumerations.H, 1607
_\_FL\_SHADOW\_LABEL
Enumerations.H, 1613
~\_Fl\_Browser, 347
~\_Fl\_Device

Generated by Doxygen
~Fl_Group
  Fl_Group, 683
~Fl_Help_View
  Fl_Help_View, 704
~Fl_Input_
  Fl_Input_, 777
~Fl_Native_File_Chooser
  Fl_Native_File_Chooser, 926
~Fl_Plugin_Manager
  Fl_Plugin_Manager, 981
~Fl_Preferences
  Fl_Preferences, 1025
~Fl_Shared_Image
  Fl_Shared_Image, 1169
~Fl_Table
  Fl_Table, 1247
~Fl_Table_Row
  Fl_Table_Row, 1267
~Fl_Text_Display
  Fl_Text_Display, 1312
~Fl_Widget
  Fl_Widget, 1516
~Fl_Window
  Fl_Window, 1556

A0
  Fl_Paged_Device, 968
A4
  Fl_Paged_Device, 968
abi-version.h, 1595
abi_check
  Fl, 300
abi_version
  Fl, 300
absolute_top_line_number
  Fl_Text_Display, 1312
activate
  Fl_Tree_Item, 1447
  Fl_Widget, 1516
active
  Fl_Widget, 1516
active_r
  Fl_Widget, 1517
add
  Fl_Browser, 347
  Fl_Chart, 417
  Fl_Check_Browser, 430
  Fl_File_ICON, 535
  Fl_Input_Choice, 800
  Fl_Menu_, 839
  Fl_Menu_Item, 873
  Fl_Shared_Image, 1169
  Fl_Sys_Menu_Bar, 1224
  Fl_Tree, 1411
  Fl_Tree_Item, 1447, 1448
  Fl_Tree_Item_Array, 1463
add_check
  Fl, 300
add_clipboard_notify
  Selection & Clipboard functions, 212
add_color
  Fl_File_ICON, 535
add_extra
  Fl_File_Chooser, 532
add_fd
  Fl, 300
add_handler
  Events handling functions, 201
add_idle
  Fl, 301
add_key_binding
  Fl_Text_Editor, 1353
add_modify_callback
  Fl_Text_Buffer, 1286
add_system_handler
  Events handling functions, 201
add_timeout
  Fl, 301
add_vertex
  Fl_File_ICON, 535
Adding and Extending Widgets, 60
addPlugin
  Fl_Plugin_Manager, 981
address
  Fl_Text_Buffer, 1286, 1287
Advanced FLTK, 106
align
  Fl_Widget, 1517
ALWAYS_ON
  Fl_Browser_, 375
angle1
  Fl_Dial, 499
api_version
  Fl, 301
append
  Fl_Text_Buffer, 1287
appendfile
  Fl_Text_Buffer, 1287
arc
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1006
arg
  Fl, 301
args
  Fl, 302
argument
  Fl_Menu_Item, 873, 874
  Fl_Widget, 1517
armscii_8.h, 1948
array
  Fl_Group, 684
as_gl_window
  Fl_Gl_Window, 629
  Fl_Widget, 1518
as_group
  Fl_Group, 684
  Fl_Widget, 1518
as_window
  Fl_Widget, 1518
  Fl_Window, 1556
ascii.h, 1949
atclose
  Windows handling functions, 198
autosize
  Fl_Chart, 417
awake
  Multithreading support functions, 251
B
  Fl_Color_Chooser, 474
background
  Fl, 303
background2
  Fl, 303
bbox
  Fl_Browser, 376
  Fl_Scroll, 1131
begin
  Fl_Group, 684
begin_complex_polygon
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1007
begin_line
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1007
begin_loop
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1007
begin_points
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1007
begin_polygon
  Fl_Graphics_Driver, 655
  Fl_PostScript_Graphics_Driver, 1007
belowmouse
  Events handling functions, 201, 202
big5.h, 1949
big5_emacs.h, 1997
BLOCK_CURSOR
  Fl_Text_Display, 1311
border
  Fl_Window, 1556
BOT
  Fl_Browser, 375
BOT_A
  Fl_Browser, 375
bottomline
  Fl_Browser, 347
bounds
  Fl_Chart, 417, 418
  Fl_Slider, 1201
box
  Fl_Widget, 1519
box_color
  Fl, 303
box_dh
  Fl, 303
box_dw
  Fl, 304
box_dx
  Fl, 304
box_dy
  Fl, 304
BROWSE_DIRECTORY
  Fl_Native_File_Chooser, 926
BROWSE_FILE
  Fl_Native_File_Chooser, 926
BROWSE_MULTI_DIRECTORY
  Fl_Native_File_Chooser, 926
BROWSE_MULTI_FILE
  Fl_Native_File_Chooser, 926
BROWSE_SAVE_DIRECTORY
  Fl_Native_File_Chooser, 926
BROWSE_SAVE_FILE
  Fl_Native_File_Chooser, 926
buffer
  Fl_Text_Display, 1312
buffer_modified_cb
  Fl_Text_Display, 1313
buffer_predelete_cb
  Fl_Text_Display, 1313
byte_at
  Fl_Text_Buffer, 1287
Cairo Support Functions and Classes, 255
  cairo_autolink_context, 256
  cairo_cc, 256
  cairo_make_current, 256
cairo_autolink_context
  Cairo Support Functions and Classes, 256
cairo_cc
  Cairo Support Functions and Classes, 256
cairo_make_current
  Cairo Support Functions and Classes, 256
calc_dimensions
  Fl_Tree, 1413
calc_item_height
  Fl_Tree_Item, 1448
calc_last_char
  Fl_Text_Display, 1313
calc_line_starts
  Fl_Text_Display, 1313
calc_tree
  Fl_Tree, 1413
callback
  Fl_Menu_Item, 874
  Fl_Table, 1247
  Fl_Widget, 1519, 1520
Callback function typedefs, 195
  Fl_Event_Dispatch, 196
callback_col
  Fl_Table, 1248
callback_context
  Fl_Table, 1248
callback_item
INDEX

Fl_Table, 1249
col_resize_min
  Fl_Table, 1249
col_to_x
  Fl_Text_Display, 1314
col_width
  Fl_Table, 1249
col_width_all
  Fl_Table, 1249
color
  Fl_GDI_Graphics_Driver, 605
  Fl_Graphics_Driver, 656
  Fl_PostScript_Graphics_Driver, 1008
  Fl_Quartz_Graphics_Driver, 1053
  Fl_Tooltip, 1391
  Fl_Widget, 1522
  Fl_Xlib_Graphics_Driver, 1583
Color & Font functions, 218
  fl_color, 220
  fl_color_average, 221
  fl_contrast, 221
  fl_font, 221
  fl_height, 221
  fl_latin1_to_local, 222
  fl_local_to_latin1, 222
  fl_local_to_mac_roman, 222
  fl_mac_roman_to_local, 223
  fl_show_colormap, 223
  fl_size, 224
  fl_text_extents, 224
  fl_width, 224
  fl_xpixel, 224, 225
  free_color, 225
  get_color, 225
  get_font, 226
  get_font_name, 226
  get_font_sizes, 226
  set_color, 226
  set_font, 227
  set_fonts, 227
color2
  Fl_Widget, 1523
color_average
  Fl_Image, 752
  Fl_Pixmap, 976
  Fl_RGB_Image, 1098
  Fl_Shared_Image, 1170
  Fl_Tiled_Image, 1374
column_char
  Fl_Browser, 348
column_widths
  Fl_Browser, 348
Common Dialogs classes and functions, 272
error, 282
fatal, 283
fl_alert, 273
fl_ask, 273
fl_beep, 274
fl_choice, 274
fl_choice_n, 275
fl_color_chooser, 276, 277
fl_dir_chooser, 278
fl_file_chooser, 279
fl_file_chooser_callback, 279
fl_file_chooser_ok_label, 280
fl_input, 280
fl_message, 280
fl_message_hotspot, 280, 281
fl_message_icon, 281
fl_message_title, 281
fl_message_title_default, 282
fl_password, 282
warning, 283
Common Widgets and Attributes, 14
compare
Events handling functions, 202
compose
Events handling functions, 202
connectorstyle
  Fl_Tree, 1416
contains
  Fl_Widget, 1523
context
  Fl_Gl_Window, 629
CONTEXT_CELL
  Fl_Table, 1246
CONTEXT_COL_HEADER
  Fl_Table, 1246
CONTEXT_ENDPAGE
  Fl_Table, 1246
CONTEXT_NONE
  Fl_Table, 1246
CONTEXT_RC_RESIZE
  Fl_Table, 1246
CONTEXT_ROW_HEADER
  Fl_Table, 1246
CONTEXT_STARTPAGE
  Fl_Table, 1246
CONTEXT_TABLE
  Fl_Table, 1246
context_valid
  Fl_Gl_Window, 629
COPIED_LABEL
  Fl_Widget, 1515
COPIED_TOOLTIP
  Fl_Widget, 1515
copy
  Fl_Bitmap, 323
  Fl_Image, 752, 753
  Fl_Input_, 778
  Fl_Menu_, 842
  Fl_Pixmap, 976
  Fl_RGB_Image, 1098
  Fl_Shared_Image, 1170
Generated by Doxygen
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Text_Display, 1311</td>
<td>1311</td>
</tr>
<tr>
<td>dingbats_.h, 1902</td>
<td>1902</td>
</tr>
<tr>
<td>direction</td>
<td>1381</td>
</tr>
<tr>
<td>Fl_Timer, 1381</td>
<td>1381</td>
</tr>
<tr>
<td>directory</td>
<td>926</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser, 926</td>
<td>926</td>
</tr>
<tr>
<td>dirent.h, 1595</td>
<td>1595</td>
</tr>
<tr>
<td>disable</td>
<td>1392</td>
</tr>
<tr>
<td>Fl_Tooltip, 1392</td>
<td>1392</td>
</tr>
<tr>
<td>disable_im</td>
<td>202</td>
</tr>
<tr>
<td>Events handling functions, 202</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>304</td>
</tr>
<tr>
<td>Fl, 304</td>
<td>304</td>
</tr>
<tr>
<td>Fl_Browser, 350</td>
<td>350</td>
</tr>
<tr>
<td>Fl_Browser_, 376</td>
<td>376</td>
</tr>
<tr>
<td>Fl_Tree, 1417</td>
<td>1417</td>
</tr>
<tr>
<td>display_insert</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Display, 1315</td>
<td>1315</td>
</tr>
<tr>
<td>displayed</td>
<td></td>
</tr>
<tr>
<td>Fl_Browser, 350</td>
<td>350</td>
</tr>
<tr>
<td>Fl_Browser_, 377</td>
<td>377</td>
</tr>
<tr>
<td>Fl_Tree, 1417</td>
<td>1417</td>
</tr>
<tr>
<td>dnd</td>
<td></td>
</tr>
<tr>
<td>Selection &amp; Clipboard functions, 213</td>
<td>213</td>
</tr>
<tr>
<td>dnd_text_ops</td>
<td></td>
</tr>
<tr>
<td>Fl, 304, 305</td>
<td>304, 305</td>
</tr>
<tr>
<td>do_callback</td>
<td></td>
</tr>
<tr>
<td>Fl_Menu_Item, 875</td>
<td>875</td>
</tr>
<tr>
<td>Fl_Widget, 1526</td>
<td>1526</td>
</tr>
<tr>
<td>do_widget_deletion</td>
<td></td>
</tr>
<tr>
<td>Safe widget deletion support functions, 254</td>
<td>254</td>
</tr>
<tr>
<td>down_box</td>
<td></td>
</tr>
<tr>
<td>Fl_Button, 395</td>
<td>395</td>
</tr>
<tr>
<td>Fl_File_Input, 547</td>
<td>547</td>
</tr>
<tr>
<td>Fl_Menu_, 842</td>
<td>842</td>
</tr>
<tr>
<td>draw</td>
<td></td>
</tr>
<tr>
<td>Fl_Adjuster, 320</td>
<td>320</td>
</tr>
<tr>
<td>Fl_Bitmap, 323</td>
<td>323</td>
</tr>
<tr>
<td>Fl_Box, 333</td>
<td>333</td>
</tr>
<tr>
<td>Fl_Browser_, 377</td>
<td>377</td>
</tr>
<tr>
<td>Fl_Button, 395</td>
<td>395</td>
</tr>
<tr>
<td>Fl_Cairo_Window, 409</td>
<td>409</td>
</tr>
<tr>
<td>Fl_Chart, 418</td>
<td>418</td>
</tr>
<tr>
<td>Fl_Choice, 447</td>
<td>447</td>
</tr>
<tr>
<td>Fl_Clock_Output, 464</td>
<td>464</td>
</tr>
<tr>
<td>Fl_Copy_Surface, 479</td>
<td>479</td>
</tr>
<tr>
<td>Fl.Counter, 488</td>
<td>488</td>
</tr>
<tr>
<td>Fl_Dial, 499</td>
<td>499</td>
</tr>
<tr>
<td>Fl_File_Icon, 536</td>
<td>536</td>
</tr>
<tr>
<td>Fl_File_Input, 548</td>
<td>548</td>
</tr>
<tr>
<td>Fl_FormsBitmap, 580</td>
<td>580</td>
</tr>
<tr>
<td>Fl_FormsPixmap, 586</td>
<td>586</td>
</tr>
<tr>
<td>Fl_FormsText, 593</td>
<td>593</td>
</tr>
<tr>
<td>Fl_Free, 600</td>
<td>600</td>
</tr>
<tr>
<td>Fl_GDI_Graphics_Driver, 605, 606</td>
<td>605, 606</td>
</tr>
<tr>
<td>Fl_GDI_Printer_Graphics_Driver, 613</td>
<td>613</td>
</tr>
<tr>
<td>Fl_GL_Window, 630</td>
<td>630</td>
</tr>
<tr>
<td>Fl_Glut_Window, 647</td>
<td>647</td>
</tr>
<tr>
<td>Fl_Graphics_Driver, 657, 658</td>
<td>657, 658</td>
</tr>
<tr>
<td>Fl_Group, 685</td>
<td>685</td>
</tr>
<tr>
<td>Fl_Help_View, 705</td>
<td>705</td>
</tr>
<tr>
<td>Fl_Image, 753</td>
<td>753</td>
</tr>
<tr>
<td>Fl_Image_Surface, 757</td>
<td>757</td>
</tr>
<tr>
<td>Fl_Input, 768</td>
<td>768</td>
</tr>
<tr>
<td>Fl_Label, 814</td>
<td>814</td>
</tr>
<tr>
<td>Fl_Light_Button, 821</td>
<td>821</td>
</tr>
<tr>
<td>Fl_Menu_Bar, 858</td>
<td>858</td>
</tr>
<tr>
<td>Fl_Menu_Button, 868</td>
<td>868</td>
</tr>
<tr>
<td>Fl_Pack, 965</td>
<td>965</td>
</tr>
<tr>
<td>Fl_Pixmap, 976</td>
<td>976</td>
</tr>
<tr>
<td>Fl_Positioner, 994</td>
<td>994</td>
</tr>
<tr>
<td>Fl_PostScript_Graphics_Driver, 1008, 1009</td>
<td>1008, 1009</td>
</tr>
<tr>
<td>Fl_Progress, 1048</td>
<td>1048</td>
</tr>
<tr>
<td>Fl_Quartz_Graphics_Driver, 1054, 1055</td>
<td>1054, 1055</td>
</tr>
<tr>
<td>Fl_Return_Button, 1094</td>
<td>1094</td>
</tr>
<tr>
<td>Fl_RGB_Image, 1098</td>
<td>1098</td>
</tr>
<tr>
<td>Fl_Roller, 1107</td>
<td>1107</td>
</tr>
<tr>
<td>Fl_Scroll, 1131</td>
<td>1131</td>
</tr>
<tr>
<td>Fl_Scrollbar, 1142</td>
<td>1142</td>
</tr>
<tr>
<td>Fl_Shared_Image, 1171</td>
<td>1171</td>
</tr>
<tr>
<td>Fl_Slider, 1201</td>
<td>1201</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar, 1225</td>
<td>1225</td>
</tr>
<tr>
<td>Fl_Table, 1249</td>
<td>1249</td>
</tr>
<tr>
<td>Fl_Tabs, 1280</td>
<td>1280</td>
</tr>
<tr>
<td>Fl_Text_Display, 1316</td>
<td>1316</td>
</tr>
<tr>
<td>Fl_Tiled_Image, 1374</td>
<td>1374</td>
</tr>
<tr>
<td>Fl_Timer, 1382</td>
<td>1382</td>
</tr>
<tr>
<td>Fl_Tree, 1418</td>
<td>1418</td>
</tr>
<tr>
<td>Fl_Tree_Item, 1449</td>
<td>1449</td>
</tr>
<tr>
<td>Fl_Value_Input, 1488</td>
<td>1488</td>
</tr>
<tr>
<td>Fl_Value_Output, 1498</td>
<td>1498</td>
</tr>
<tr>
<td>Fl_Value_Slider, 1507</td>
<td>1507</td>
</tr>
<tr>
<td>Fl_Widget, 1527</td>
<td>1527</td>
</tr>
<tr>
<td>Fl_Window, 1559</td>
<td>1559</td>
</tr>
<tr>
<td>Fl_Xlib_Graphics_Driver, 1584</td>
<td>1584</td>
</tr>
<tr>
<td>draw_box_active</td>
<td></td>
</tr>
<tr>
<td>Fl, 305</td>
<td>305</td>
</tr>
<tr>
<td>draw_cell</td>
<td></td>
</tr>
<tr>
<td>Fl_Table, 1249</td>
<td>1249</td>
</tr>
<tr>
<td>draw_child</td>
<td></td>
</tr>
<tr>
<td>Fl_Group, 685</td>
<td>685</td>
</tr>
<tr>
<td>draw_children</td>
<td></td>
</tr>
<tr>
<td>Fl_Group, 685</td>
<td>685</td>
</tr>
<tr>
<td>draw_cursor</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Display, 1316</td>
<td>1316</td>
</tr>
<tr>
<td>draw_decorated_window</td>
<td></td>
</tr>
<tr>
<td>Fl_Copy_Surface, 479</td>
<td>479</td>
</tr>
<tr>
<td>Fl_Image_Surface, 758</td>
<td>758</td>
</tr>
<tr>
<td>draw_empty</td>
<td></td>
</tr>
<tr>
<td>Fl_Image, 754</td>
<td>754</td>
</tr>
<tr>
<td>draw_horizontal_connector</td>
<td></td>
</tr>
<tr>
<td>Fl_Tree_Item, 1450</td>
<td>1450</td>
</tr>
<tr>
<td>draw_image</td>
<td></td>
</tr>
<tr>
<td>Fl_GDI_Graphics_Driver, 606, 607</td>
<td>606, 607</td>
</tr>
<tr>
<td>Fl_Graphics_Driver, 658</td>
<td>658</td>
</tr>
<tr>
<td>Fl_PostScript_Graphics_Driver, 1010</td>
<td>1010</td>
</tr>
</tbody>
</table>
end_complex_polygon
   Fl_Graphics_Driver, 659
   Fl_PostScript_Graphics_Driver, 1011

end_job
   Fl_Paged_Device, 969
   Fl_PostScript_File_Device, 998
   Fl_Printer, 1038
   Fl_System_Printer, 1230

end_line
   Fl_Graphics_Driver, 659
   Fl_PostScript_Graphics_Driver, 1011

end_loop
   Fl_Graphics_Driver, 659
   Fl_PostScript_Graphics_Driver, 1011

end_page
   Fl_Paged_Device, 969
   Fl_PostScript_File_Device, 998
   Fl_Printer, 1038
   Fl_System_Printer, 1230

end_points
   Fl_Graphics_Driver, 660
   Fl_PostScript_Graphics_Driver, 1011

end_polygon
   Fl_Graphics_Driver, 660
   Fl_PostScript_Graphics_Driver, 1011

enter_area
   Fl_Tooltip, 1392

entries
   Fl_Preferences, 1026

entry
   Fl_Preferences, 1026

entryExists
   Fl_Preferences, 1026

Enumerations.H, 1595, 1615
   _FL_DIAMOND_DOWN_BOX, 1608
   _FL_DIAMOND_UP_BOX, 1608
   _FL_EMBOSSED_LABEL, 1613
   _FL_ENGRAVED_LABEL, 1613
   _FL_GLEAM_DOWN_BOX, 1608
   _FL_GLEAM_DOWN_FRAME, 1608
   _FL_GLEAM_ROUND_DOWN_BOX, 1608
   _FL_GLEAM_ROUND_UP_BOX, 1608
   _FL_GLEAM_THIN_DOWN_BOX, 1608
   _FL_GLEAM_THIN_UP_BOX, 1608
   _FL_GLEAM_UP_BOX, 1608
   _FL_GLEAM_UP_FRAME, 1608
   _FL_ICON_LABEL, 1613
   _FL_IMAGE_LABEL, 1613
   _FL_MULTI_LABEL, 1613
   _FL_OFLAT_BOX, 1608
   _FL_OSHADOW_BOX, 1608
   _FL_OVAL_BOX, 1608
   _FL_OVAL_FRAME, 1608
   _FL_PLASTIC_DOWN_BOX, 1608
   _FL_PLASTIC_DOWN_FRAME, 1608
   _FL_PLASTIC_ROUND_DOWN_BOX, 1608
   _FL_PLASTIC_ROUND_UP_BOX, 1608
   _FL_PLASTIC_THIN_DOWN_BOX, 1608
   _FL_PLASTIC_THIN_UP_BOX, 1608
   _FL_PLASTIC_UP_BOX, 1608
   _FL_PLASTIC_UP_FRAME, 1608
   _FL_RFLAT_BOX, 1608
   _FL_ROUNDED_BOX, 1607
   _FL_ROUNDED_FRAME, 1607
   _FL_ROUND_DOWN_BOX, 1608
   _FL_ROUND_UP_BOX, 1608
   _FL_RSHADOW_BOX, 1608
   _FL_SHADOW_BOX, 1607
   _FL_SHADOW_FRAME, 1607
   _FL_SHADOW_LABEL, 1613
   _FL_ABI_VERSION, 1605
   FL_ACTIVATE, 1611
   FL_ALIGN_LEFT, 1614
   FL_ALIGN_TOP, 1614
   FL_API_VERSION, 1606
   FL_BORDER_BOX, 1607
   FL_BORDER_FRAME, 1607
   Fl_box, 1613
   Fl_Boxtype, 1607
   FL_CLOSE, 1611
   fl_color_cube, 1613
   Fl_Cursor, 1608
   FL_CURSOR_ARROW, 1609
   FL_CURSOR_CROSS, 1609
   FL_CURSOR_DEFAULT, 1609
   FL_CURSOR_E, 1609
   FL_CURSOR_HAND, 1609
   FL_CURSOR_INSERT, 1609
   FL_CURSOR_HELP, 1609
   FL_CURSOR_MOVE, 1609
   FL_CURSOR_N, 1609
   FL_CURSOR_NE, 1609
   FL_CURSOR_NESW, 1609
   FL_CURSOR_NONE, 1609
   FL_CURSOR_NS, 1609
   FL_CURSOR_NW, 1609
   FL_CURSOR_SW, 1609
   FL_CURSOR_S, 1609
   FL_CURSOR_SE, 1609
   FL_CURSOR_SE, 1609
   FL_CURSOR_W, 1609
   FL_CURSOR_W, 1609
   FL_CURSOR_WAIT, 1609
   FL_CURSOR_WE, 1609
   Fl_Damage, 1609
   FL_DAMAGE_ALL, 1609
   FL_DAMAGE_CHILD, 1609

Generated by Doxygen
find_wrap_range
  FL_Text_Display, 1318
find_x
  FL_Text_Display, 1319
findchar_backward
  FL_Text_Buffer, 1288
findchar_forward
  FL_Text_Buffer, 1289
first
  FL_Tree, 1420
first_selected_item
  FL_Tree, 1420
first_visible
  FL_Tree, 1420
first_visible_item
  FL_Tree, 1420
first_window
  Windows handling functions, 197
Fl, 291
  abi_check, 300
  abi_version, 300
  add_check, 300
  add_fd, 300
  add_idle, 301
  add_timeout, 301
  api_version, 301
  arg, 301
  args, 302
  background, 303
  background2, 303
  box_color, 303
  box_dh, 303
  box_dw, 304
  box_dx, 304
  box_dy, 304
  check, 304
  display, 304
  dnd_text_ops, 304, 305
  draw_box_active, 305
  FL_Option, 299
  flush, 305
  get_system_colors, 305
  gl_visual, 305
  help, 311
  idle, 311
  is_scheme, 305
  option, 306, 307
  OPTION_ARROW_FOCUS, 299
  OPTION_DND_TEXT, 299
  OPTION_FNFC_USES_GTK, 299
  OPTION_LAST, 299
  OPTION_SHOW_TOOLTIPS, 299
  OPTION_VISIBLE_FOCUS, 299
  own_colormap, 307
  readqueue, 307
  ready, 307
  release, 308
  reload_scheme, 308
remove_check, 308
remove_timeout, 308
repeat_timeout, 308
run, 308
scheme, 309
scrollbar_size, 309
set_box_color, 309
set_idle, 310
use_high_res_GL, 310
version, 310
visible_focus, 310
visual, 310
wait, 311
Fl.H, 1625, 1626
FL_ABI_VERSION
  Enumerations.H, 1605
fl_access
  Unicode and UTF-8 functions, 259
FL_ACTIVATE
  Enumerations.H, 1611
fl_add_symbol
  Drawing functions, 232
Fl_Adjuster, 312
  draw, 320
  Fl_Adjuster, 320
  handle, 320
  soft, 321
  value_damage, 321
Fl_Adjuster.H, 1632
fl_alert
  Common Dialogs classes and functions, 273
FL_ALIGN_LEFT
  Enumerations.H, 1614
FL_ALIGN_TOP
  Enumerations.H, 1614
FL_API_VERSION
  Enumerations.H, 1606
fl_arc
  Drawing functions, 233
  Fl_Graphics_Driver, 665
fl_arc.cxx, 1835
fl_arci.cxx, 1835
fl_ask
  Common Dialogs classes and functions, 273
fl_ask.cxx, 1835
fl_ask.H, 1632, 1634
Fl_Beep, 1634
  FL_BEEP_DEFAULT, 1634
  FL_BEEP_ERROR, 1634
  FL_BEEP_MESSAGE, 1634
  FL_BEEP_NOTIFICATION, 1634
  FL_BEEP_PASSWORD, 1634
  FL_BEEP_QUESTION, 1634
Fl_Beep
  fl_ask.H, 1634
fl beep
  Common Dialogs classes and functions, 274
FL_BEEP_DEFAULT
INDEX

fl_ask.H, 1634
FL_BEEP_ERROR
fl_ask.H, 1634
FL_BEEP_MESSAGE
fl_ask.H, 1634
FL_BEEP_NOTIFICATION
fl_ask.H, 1634
FL_BEEP_PASSWORD
fl_ask.H, 1634
FL_BEEP_QUESTION
fl_ask.H, 1634
fl_begin_complex_polygon
Drawing functions, 234
Fl_Graphics_Driver, 666
fl_begin_offscreen
Drawing functions, 234
fl_begin_points
Drawing functions, 234
Fl_Graphics_Driver, 666
Fl_Bitmap, 321
copy, 323
draw, 323
label, 324
uncache, 324
Fl_Bitmap.H, 1635
Fl_BMP_Image, 324
Fl_BMP_Image, 327
Fl_BMP_Image.H, 1636
FL_BORDER_BOX
Enumerations.H, 1607
FL_BORDER_FRAME
Enumerations.H, 1607
Fl_Box, 327
draw, 333
Fl_Box, 333
handle, 333
Fl_box
Enumerations.H, 1613
Fl_Box.H, 1636
Fl_Boxtype
Enumerations.H, 1607
fl_boxtype.cxx, 1836
    fl_internal_boxtype, 1838
    fl_rectbound, 1839
Fl_Browser, 334
    _remove, 347
    add, 347
    bottomline, 347
clear, 347
column_char, 348
column_widths, 348
data, 348, 350
display, 350
displayed, 350
find_line, 351
Fl_Browser, 346
format_char, 351, 352
full_height, 352
hide, 352
icon, 352, 353
incr_height, 353
insert, 353, 354
item_at, 354
item_draw, 354
item_first, 355
item_height, 355
item_last, 355
item_next, 355
item_prev, 356
item_select, 356
item_selected, 357
item_swap, 357
item_text, 357
item_width, 357
lineno, 358
lineposition, 358
load, 358
make_visible, 359
middeline, 359
move, 359
remove, 361
remove_icon, 361
select, 361
selected, 361
show, 362
size, 362
swap, 362, 363
text, 363
textsize, 363
topleft, 364
value, 364
visible, 364
Fl_Browser.H, 1637
Fl_Browser_., 365
    ALWAYS_ON, 375
    bbox, 376
    BOTH, 375
    BOTH_ALWAYS, 375
deleting, 376
deselect, 376
display, 376
displayed, 377
draw, 377
find_item, 377
Fl_Browser_., 375
full_height, 377
full_width, 378
handle, 378
has_scrollbar, 378
HORIZONTAL, 375
HORIZONTAL_ALWAYS, 375
hposition, 378, 379
hscrollbar, 378
incr_height, 379
inserting, 379
item_at, 379
Generated by Doxygen
<table>
<thead>
<tr>
<th>Item / Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_draw</td>
<td>380</td>
</tr>
<tr>
<td>item_first</td>
<td>380</td>
</tr>
<tr>
<td>item_height</td>
<td>380</td>
</tr>
<tr>
<td>item_last</td>
<td>380</td>
</tr>
<tr>
<td>item_next</td>
<td>380</td>
</tr>
<tr>
<td>item_prev</td>
<td>381</td>
</tr>
<tr>
<td>item_quick_height</td>
<td>381</td>
</tr>
<tr>
<td>item_select</td>
<td>381</td>
</tr>
<tr>
<td>item_selected</td>
<td>381</td>
</tr>
<tr>
<td>item_swap</td>
<td>382</td>
</tr>
<tr>
<td>item_text</td>
<td>382</td>
</tr>
<tr>
<td>item_width</td>
<td>382</td>
</tr>
<tr>
<td>ledgedge</td>
<td>382</td>
</tr>
<tr>
<td>new_list</td>
<td>383</td>
</tr>
<tr>
<td>position</td>
<td>383</td>
</tr>
<tr>
<td>redraw_line</td>
<td>383</td>
</tr>
<tr>
<td>redraw_lines</td>
<td>384</td>
</tr>
<tr>
<td>replacing</td>
<td>384</td>
</tr>
<tr>
<td>resize</td>
<td>384</td>
</tr>
<tr>
<td>scrollbar</td>
<td>387</td>
</tr>
<tr>
<td>scrollbar_left</td>
<td>387</td>
</tr>
<tr>
<td>scrollbar_right</td>
<td>384</td>
</tr>
<tr>
<td>scrollbar_size</td>
<td>384, 385</td>
</tr>
<tr>
<td>scrollbar_width</td>
<td>385</td>
</tr>
<tr>
<td>select</td>
<td>385</td>
</tr>
<tr>
<td>select_only</td>
<td>386</td>
</tr>
<tr>
<td>selection</td>
<td>386</td>
</tr>
<tr>
<td>sort</td>
<td>386</td>
</tr>
<tr>
<td>swapping</td>
<td>386</td>
</tr>
<tr>
<td>textfont</td>
<td>387</td>
</tr>
<tr>
<td>VERTICAL</td>
<td>375</td>
</tr>
<tr>
<td>VERTICAL_ALWAYS</td>
<td>375</td>
</tr>
<tr>
<td>Fl_Browser_.H</td>
<td>1639</td>
</tr>
<tr>
<td>Fl_Button</td>
<td>387</td>
</tr>
<tr>
<td>clear</td>
<td>395</td>
</tr>
<tr>
<td>down_box</td>
<td>395</td>
</tr>
<tr>
<td>draw</td>
<td>395</td>
</tr>
<tr>
<td>Fl_Button</td>
<td>394</td>
</tr>
<tr>
<td>handle</td>
<td>395</td>
</tr>
<tr>
<td>set</td>
<td>396</td>
</tr>
<tr>
<td>shortcut</td>
<td>396</td>
</tr>
<tr>
<td>value</td>
<td>396</td>
</tr>
<tr>
<td>Fl_Button.H</td>
<td>1641</td>
</tr>
<tr>
<td>Fl_Cairo.H</td>
<td>1642</td>
</tr>
<tr>
<td>Fl_Cairo_State</td>
<td>397</td>
</tr>
<tr>
<td>cc</td>
<td>398</td>
</tr>
<tr>
<td>Fl_Cairo_Window</td>
<td>398</td>
</tr>
<tr>
<td>draw</td>
<td>409</td>
</tr>
<tr>
<td>set_draw_cb</td>
<td>409</td>
</tr>
<tr>
<td>Fl_Cairo_Window.H</td>
<td>1642</td>
</tr>
<tr>
<td>fl_can_do_alpha_blending</td>
<td>234</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>234</td>
</tr>
<tr>
<td>FL_CAP_FLAT</td>
<td>232</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>232</td>
</tr>
<tr>
<td>FL_CAP_ROUND</td>
<td>232</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>232</td>
</tr>
<tr>
<td>FL_CAP_SQUARE</td>
<td>232</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>232</td>
</tr>
<tr>
<td>Fl_Chart</td>
<td>409</td>
</tr>
<tr>
<td>add</td>
<td>417</td>
</tr>
<tr>
<td>autosize</td>
<td>417</td>
</tr>
<tr>
<td>bounds</td>
<td>417, 418</td>
</tr>
<tr>
<td>draw</td>
<td>418</td>
</tr>
<tr>
<td>Fl_Chart.H</td>
<td>1643</td>
</tr>
<tr>
<td>FL_CHART_ENTRY</td>
<td>419</td>
</tr>
<tr>
<td>Fl_Check_Browser</td>
<td>419</td>
</tr>
<tr>
<td>add</td>
<td>430</td>
</tr>
<tr>
<td>handle</td>
<td>430</td>
</tr>
<tr>
<td>nchecked</td>
<td>430</td>
</tr>
<tr>
<td>nitems</td>
<td>430</td>
</tr>
<tr>
<td>remove</td>
<td>430</td>
</tr>
<tr>
<td>set_checked</td>
<td>430</td>
</tr>
<tr>
<td>Fl_Check_Browser.H</td>
<td>1644</td>
</tr>
<tr>
<td>Fl_Check_Button</td>
<td>431</td>
</tr>
<tr>
<td>Fl_Check_Button.H</td>
<td>1646</td>
</tr>
<tr>
<td>fl_chmod</td>
<td>259</td>
</tr>
<tr>
<td>Unicode and UTF-8 functions</td>
<td>259</td>
</tr>
<tr>
<td>Fl_Choice</td>
<td>438</td>
</tr>
<tr>
<td>draw</td>
<td>447</td>
</tr>
<tr>
<td>value</td>
<td>448</td>
</tr>
<tr>
<td>Fl_Choice.H</td>
<td>1646</td>
</tr>
<tr>
<td>fl_choice_n</td>
<td>275</td>
</tr>
<tr>
<td>Common Dialogs classes and functions</td>
<td>275</td>
</tr>
<tr>
<td>fl_circle</td>
<td>234</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>234</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>666</td>
</tr>
<tr>
<td>fl_clip</td>
<td>232</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>232</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>666</td>
</tr>
<tr>
<td>fl_clip_region</td>
<td>235</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>235</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>667</td>
</tr>
<tr>
<td>Fl_Clock</td>
<td>449</td>
</tr>
<tr>
<td>Fl_Clock.H</td>
<td>1647</td>
</tr>
<tr>
<td>Fl_Clock_Output</td>
<td>457</td>
</tr>
<tr>
<td>draw</td>
<td>464</td>
</tr>
<tr>
<td>Fl_Clock_Output.H</td>
<td>463</td>
</tr>
<tr>
<td>hour</td>
<td>464</td>
</tr>
<tr>
<td>minute</td>
<td>464</td>
</tr>
<tr>
<td>second</td>
<td>464</td>
</tr>
<tr>
<td>value</td>
<td>464, 465</td>
</tr>
<tr>
<td>Fl_CLOSE</td>
<td>1611</td>
</tr>
</tbody>
</table>
fl cmap.h, 1839
fl color
  Color & Font functions, 220
  Fl Graphics Driver, 667
fl color.cxx, 1842
fl color_average
  Color & Font functions, 221
Fl Color Chooser, 465
  b, 474
  Fl Color Chooser, 474
g, 474
  hsv, 474
  hsv2rgb, 475
  hue, 475
  mode, 475
  r, 475
  rgb, 475
  rgb2hsv, 476
  saturation, 476
  value, 476
fl color chooser
  Common Dialogs classes and functions, 276, 277
Fl Color Chooser.H, 1648
fl color cube
  Enumerations.H, 1613
Fl compose.cxx, 1843
fl contrast
  Color & Font functions, 221
fl copy offscreen
  Drawing functions, 235
  Fl Graphics Driver, 667
Fl Copy Surface, 476
  class_name, 479
  draw, 479
  draw decorated window, 479
  Fl Copy Surface, 478
  set current, 479
Fl Copy Surface.H, 1649
Fl Counter, 480
  draw, 488
  Fl Counter, 487
  handle, 488
  lstep, 488
  step, 489
Fl Counter.H, 1651
fl create offscreen
  Drawing functions, 235
Fl CString
  fl_types.h, 1782
Fl Cursor
  Enumerations.H, 1608
fl cursor
  Drawing functions, 236
FL CURSOR ARROW
  Enumerations.H, 1609
FL CURSOR CROSS
  Enumerations.H, 1609
FL CURSOR DEFAULT
  Enumerations.H, 1609
FL CURSOR E
  Enumerations.H, 1609
FL CURSOR HAND
  Enumerations.H, 1609
FL CURSOR HELP
  Enumerations.H, 1609
FL CURSOR INSERT
  Enumerations.H, 1609
FL CURSOR MOVE
  Enumerations.H, 1609
FL CURSOR N
  Enumerations.H, 1609
FL CURSOR NE
  Enumerations.H, 1609
FL CURSOR NESW
  Enumerations.H, 1609
FL CURSOR NONE
  Enumerations.H, 1609
FL CURSOR NS
  Enumerations.H, 1609
FL CURSOR NW
  Enumerations.H, 1609
FL CURSOR NWSW
  Enumerations.H, 1609
FL CURSOR S
  Enumerations.H, 1609
FL CURSOR SE
  Enumerations.H, 1609
FL CURSOR SW
  Enumerations.H, 1609
FL CURSOR W
  Enumerations.H, 1609
FL CURSOR WAIT
  Enumerations.H, 1609
FL CURSOR WE
  Enumerations.H, 1609
fl curve
  Drawing functions, 236
  Fl Graphics Driver, 668
fl curve.cxx, 1843
fl Damage
  Enumerations.H, 1609
FL DAMAGE ALL
  Enumerations.H, 1609
FL DAMAGE CHILD
  Enumerations.H, 1609
FL DAMAGE EXPOSE
  Enumerations.H, 1609
FL DAMAGE OVERLAY
  Enumerations.H, 1609
FL DAMAGE SCROLL
  Enumerations.H, 1609
FL DAMAGE USER1
  Enumerations.H, 1609
FL DAMAGE USER2
  Enumerations.H, 1609
FL DASH
INDEX

Drawing functions, 232
FL_DASHDOT
  Drawing functions, 232
FL_DASHDOTDOT
  Drawing functions, 232
FL_DEACTIVATE
  Enumerations.H, 1611
fl_decode_uri
  File names and URI utility functions, 284
fl_delete_offscreen
  Drawing functions, 236
Fl_Device, 489
  ~Fl_Device, 490
class_id, 490
class_name, 490
Fl_Device.H, 1652, 1653
  Fl_Draw_Image_Cb, 1653
Fl_Device_Plugin, 490
  print, 491
  rectangle_capture, 491
Fl_Dial, 492
  angle1, 499
draw, 499
  Fl_Dial, 499
  handle, 500
Fl_Dial.H, 1658
fl_dirChooser
  Common Dialogs classes and functions, 278
Fl_Display_Device, 500
  class_name, 502
FL_DND_DRAG
  Enumerations.H, 1612
FL_DND_ENTER
  Enumerations.H, 1612
FL_DND_LEAVE
  Enumerations.H, 1612
FL_DND_RELEASE
  Enumerations.H, 1612
fl_dnd_x.cxx, 1843
FL_DOT
  Drawing functions, 232
Fl_Double_Window, 502
  ~Fl_Double_Window, 513
  flush, 513
  hide, 513
  resize, 513
  show, 513
Fl_Double_Window.cxx, 1846
Fl_Double_Window.H, 1659
fl_down
  Enumerations.H, 1614
FL_DOWN_BOX
  Enumerations.H, 1607
FL_DOWN_FRAME
  Enumerations.H, 1607
FL_DRAG
  Enumerations.H, 1610
fl_draw
  Drawing functions, 236, 237
  Fl_Graphics_Driver, 668
fl_draw.H, 1659, 1664
fl_draw_box
  Drawing functions, 237
fl_draw_image
  Drawing functions, 238
  Fl_Graphics_Driver, 668, 669
Fl_Draw_Image_Cb
  Fl_Device.H, 1653
fl_draw_image_mono
  Drawing functions, 239
  Fl_Graphics_Driver, 670
fl_drawPixmap
  Drawing functions, 240
fl_draw_symbol
  Drawing functions, 240
FL_EMBOSSED_BOX
  Enumerations.H, 1607
FL_EMBOSSED_FRAME
  Enumerations.H, 1607
Fl_End, 514
FL_ENGRAVED_BOX
  Enumerations.H, 1607
FL_ENGRAVED_FRAME
  Enumerations.H, 1607
FL_ENTER
  Enumerations.H, 1610
Flow
  Enumerations.H, 1609
FL_Event
  Enumerations.H, 1612
FL_Event_Dispatch
  Callback function typedefs, 196
fl_eventnames
  Events handling functions, 211
FL_EXCEPT
  Enumerations.H, 1607
fl_expand_text
  Drawing functions, 241
Fl_Export.H, 1667
Fl_File_Browser, 514
  filetype, 527
  filter, 527
  Fl_File_Browser, 527
  iconsize, 527
  load, 527
Fl_File_Browser.H, 1668
Fl_File_Chooser, 528
  add_extra, 532
  filter, 532
  Fl_File_Chooser, 532
  iconsize, 532, 533
  preview, 533
  showHiddenButton, 533
  value, 533
fl_fileChooser
  Common Dialogs classes and functions, 279
Fl_File_Chooser.H, 1669
fl_fileChooserCallback
  Callback function typedefs, 196
Generated by Doxygen
Common Dialogs classes and functions, 279

**Fl_FileChooser**

Common Dialogs classes and functions, 280

**Fl_File_Icon**

- add, 535
- add_color, 535
- add_vertex, 535
- draw, 536
- find, 536
- label, 536
- labeltype, 536
- load, 537
- load_fdi, 537
- load_image, 537
- load_system_icons, 538
- next, 538
- type, 538

**Fl_File_Icon.H**, 1671

**Fl_File_Input**

- down_box, 547
- draw, 548
- errorcolor, 548
- handle, 548
- value, 548

**Fl_File_Input.H**, 1673

**Fl_File_Sort_F**

- draw, 580
- set, 580

**Fl_File_Icon.H**, 1675

**Fl_Float_Input**

- draw, 586
- set, 587

**Fl_Float_Input.H**, 1674

**Fl_FLTK_FileChooser**, 572

**FL_FOCUS**

- Enumerations.H, 1610
- Color & Font functions, 221
- Fl_Graphics_Driver, 670
- Fl_Font.H, 1846
- Fl_Font_Descriptor, 573
- fl_font_x.cxx, 1847
- Fl_Fontdesc, 574
- fl_fontnames
  - Events handling functions, 211
- Fl_Fontsize
  - Enumerations.H, 1607
- fl_fopen
  - Unicode and UTF-8 functions, 260
- Fl_FormsBitmap
  - draw, 574
  - set, 580
- Fl_FormsBitmap.H, 1675
- Fl_FormsPixmap
  - draw, 586
  - Fl_FormsPixmap, 586
  - Pixmap, 587
  - set, 587
- Fl_FormsPixmap.H, 1675
- Fl_FormsText
  - draw, 593
- fl_frame
  - Drawing functions, 241
  - Enumerations.H, 1614
- fl_frame2
  - Drawing functions, 241
- Fl_Free
  - draw, 593
  - Fl_Free, 599
  - handle, 600
- Fl_Free.H, 1676
- FL_FREE_BOXTYPE
  - Enumerations.H, 1608
- FL_FREE_LABELTYPE
  - Enumerations.H, 1613
- FL_FULLSCREEN
  - Enumerations.H, 1612
- fl_gap
  - Drawing functions, 242
  - Fl_Graphics_Driver, 670
- Fl_GDI_Graphics_Driver
  - class_name, 605
  - color, 605
  - copy_offscreen, 605
  - descent, 605
  - draw, 605, 606
  - draw_image, 606, 607
  - draw_image_mono, 607
  - font, 607
  - height, 607
  - rtl_draw, 608

Generated by Doxygen
global, 844
insert, 845
item_pathname, 845
menu, 846
mode, 846
mvalue, 846
picked, 847
remove, 847
replace, 847
size, 847
testShortcut, 847
text, 848
textcolor, 848
textFont, 848
textsize, 848
value, 848, 849
Fl_Menu_.H, 1698
Fl_Menu_Bar, 849
draw, 858
Fl_Menu_Bar, 857
handle, 858
Fl_Menu_Bar.H, 1699
Fl_Menu_Button, 859
draw, 868
Fl_Menu_Button, 868
handle, 868
popup, 868
POPUP1, 867
POPUP12, 867
POPUP123, 868
POPUP13, 867
POPUP2, 867
POPUP23, 867
POPUP3, 867
popupButtons, 867
Fl_Menu_Button.H, 1700
FL_MENU_DIVIDER
Fl_Menu_Item.H, 1701
FL_MENU_HORIZONTAL
Fl_Menu_Item.H, 1701
FL_MENU_INACTIVE
Fl_Menu_Item.H, 1701
FL_MENU_INVISIBLE
Fl_Menu_Item.H, 1701
FL_MENU_RADIO
Fl_Menu_Item.H, 1701
FL_MENU_TOGGLE
Fl_Menu_Item.H, 1701
FL_MENU_VALUE
Fl_Menu_Item.H, 1701
FL_SUBMENU
Fl_Menu_Item.H, 1701
FL_SUBMENU_POINTER
Fl_Menu_Item.H, 1701
FL_MENU_RADIO
Fl_Menu_Item.H, 1701
FL_MENU_TOGGLE
Fl_Menu_Item.H, 1701
FL_MENU_VALUE
Fl_Menu_Item.H, 1701
FL_Submenu
Fl_Menu_Item.H, 1701
fl_message
Common Dialogs classes and functions, 280
fl_message.H, 1704
fl_message_hotspot
Common Dialogs classes and functions, 280, 281
fl_message_icon
Common Dialogs classes and functions, 281
fl_message_title
Common Dialogs classes and functions, 281
fl_message_title_default
Common Dialogs classes and functions, 282
FL_MINOR_VERSION
Enumerations.H, 1606
fl_mkdir
Unicode and UTF-8 functions, 261
FL_MOUSEWHEEL
Enumerations.H, 1612
FL_MOVE
Enumerations.H, 1611
fl_mult_matrix
Drawing functions, 243
FL_Native_File_Chooser, 926
BROWSE_DIRECTORY, 926
BROWSE_FILE, 926
BROWSE_MULTI_DIRECTORY, 926
BROWSE_MULTI_FILE, 926
BROWSE_SAVE_DIRECTORY, 926
BROWSE_SAVE_FILE, 926
count, 926
directory, 926
ererrmsg, 926
filename, 926
filter, 927
filter_value, 927
Fl_Native_File_Chooser, 926
NEW_FOLDER, 925
NO_OPTIONS, 925
Option, 925
options, 927
preset_file, 928
PREVIEW, 925
SAVEAS_CONFIRM, 925
show, 928
title, 928
Type, 925
USE_FILTER_EXT, 925
Fl_Native_File_Chooser.H, 1706, 1707
Fl_Native_File_Chooser_common.cxx, 1853
Fl_Native_File_Chooser_FLTK.cxx, 1854
Fl_Native_File_Chooser_GTلك.cxx, 1859
Fl_Nice_Slider, 928
Fl_Nice_Slider.H, 1710
FL_NO_BOX
Enumerations.H, 1607
FL_NO_EVENT
Enumerations.H, 1610
FL_NO_LABEL
Enumerations.H, 1613
fl_nonspacing
Unicode and UTF-8 functions, 261
FL_NORMAL_LABEL
Enumerations.H, 1613
FL_NORMAL_SIZE
Enumerations.H, 1614
fl_not_clipped
Drawing functions, 243
Fl_Graphics_Driver, 671
Fl_Object.H, 1710
fl_old_shortcut
Drawing functions, 244
fl_open
Unicode and UTF-8 functions, 261
fl_open_callback
Mac OS X-specific symbols, 271
fl_open_uri
File names and URI utility functions, 289
Fl_Option
Fl, 299
Fl_Output, 936
Fl_Output.H, 944
Fl_Output.H, 1711
Fl_Overlay_Window, 945
draw_overlay, 956
Fl_Overlay_Window.H, 956
flush, 956
hide, 956
redraw_overlay, 956
resize, 957
show, 957
Fl_Overlay_Window.H, 1711
Fl_Pack, 957
draw, 965
Fl_Pack, 965
Fl_Pack.H, 1712
Fl_Paged_Device, 966
A0, 968
A4, 968
class_name, 969
detect_page, 969
LANDSCAPE, 968
LETTER, 968
margins, 969
ORIENTATION, 969
origin, 969, 970
Page_Format, 968
Page_Layout, 968
PORTRAIT, 968
print_widget, 970
print_window, 970
print_window_part, 971
printable_rect, 971
REVERSED, 969
rotate, 971
scale, 971
start_job, 972
start_page, 972
translate, 972
untranslate, 972
Fl_Paged_Device.cxx, 1867
<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Paged_Device.H</td>
<td>1713</td>
</tr>
<tr>
<td>Fl_Paged_Device::page_format</td>
<td>1592</td>
</tr>
<tr>
<td>fl_password</td>
<td>Common Dialogs classes and functions, 282</td>
</tr>
<tr>
<td>FL_PASTE</td>
<td>Enumerations.H, 1612</td>
</tr>
<tr>
<td>FL_PATCH_VERSION</td>
<td>Enumerations.H, 1606</td>
</tr>
<tr>
<td>fl_pie</td>
<td>Drawing functions, 245</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>672</td>
</tr>
<tr>
<td>Fl_Pixmap</td>
<td>973</td>
</tr>
<tr>
<td>color_average</td>
<td>976</td>
</tr>
<tr>
<td>copy</td>
<td>976</td>
</tr>
<tr>
<td>desaturate</td>
<td>976</td>
</tr>
<tr>
<td>draw</td>
<td>976</td>
</tr>
<tr>
<td>Fl_Pixmap</td>
<td>975</td>
</tr>
<tr>
<td>label</td>
<td>976, 977</td>
</tr>
<tr>
<td>uncache</td>
<td>977</td>
</tr>
<tr>
<td>Fl_Pixmap.H</td>
<td>1714</td>
</tr>
<tr>
<td>Fl_Plugin</td>
<td>977</td>
</tr>
<tr>
<td>Fl_Plugin.H</td>
<td>1716</td>
</tr>
<tr>
<td>Fl_Plugin_Manager</td>
<td>978</td>
</tr>
<tr>
<td>Fl_PLUGIN Manger, 981</td>
<td>~Fl_Plugin_Manager, 981</td>
</tr>
<tr>
<td>addPlugin</td>
<td>981</td>
</tr>
<tr>
<td>load</td>
<td>981</td>
</tr>
<tr>
<td>removePlugin</td>
<td>981</td>
</tr>
<tr>
<td>Fl_PNG_Image</td>
<td>981</td>
</tr>
<tr>
<td>Fl_PNG_Image.H</td>
<td>1716</td>
</tr>
<tr>
<td>Fl_PNM_Image</td>
<td>985</td>
</tr>
<tr>
<td>Fl_PNM_Image.H</td>
<td>1717</td>
</tr>
<tr>
<td>fl_polygon</td>
<td>Drawing functions, 245</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>672</td>
</tr>
<tr>
<td>fl_pop_clip</td>
<td>Drawing functions, 245</td>
</tr>
<tr>
<td>Fl_Graphics_Driver</td>
<td>672</td>
</tr>
<tr>
<td>Fl_Positioner</td>
<td>987</td>
</tr>
<tr>
<td>draw</td>
<td>994</td>
</tr>
<tr>
<td>Fl_Positioner</td>
<td>994</td>
</tr>
<tr>
<td>handle</td>
<td>994</td>
</tr>
<tr>
<td>Fl_Positioner.H</td>
<td>1717</td>
</tr>
<tr>
<td>Fl_PostScript.H</td>
<td>1718, 1719</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>995</td>
</tr>
<tr>
<td>class_name</td>
<td>998</td>
</tr>
<tr>
<td>end_job</td>
<td>998</td>
</tr>
<tr>
<td>end_page</td>
<td>998</td>
</tr>
<tr>
<td>margins</td>
<td>998</td>
</tr>
<tr>
<td>origin</td>
<td>999</td>
</tr>
<tr>
<td>printable_rect</td>
<td>999</td>
</tr>
<tr>
<td>rotate</td>
<td>999</td>
</tr>
<tr>
<td>scale</td>
<td>1000</td>
</tr>
<tr>
<td>start_job</td>
<td>1000, 1001</td>
</tr>
<tr>
<td>start_page</td>
<td>1001</td>
</tr>
<tr>
<td>translate</td>
<td>1001</td>
</tr>
<tr>
<td>Fl_PostScript_Graphics_Driver</td>
<td>1002</td>
</tr>
<tr>
<td>arc</td>
<td>1006</td>
</tr>
<tr>
<td>begin_complex_polygon</td>
<td>1007</td>
</tr>
<tr>
<td>begin_line</td>
<td>1007</td>
</tr>
<tr>
<td>begin_loop</td>
<td>1007</td>
</tr>
<tr>
<td>begin_points</td>
<td>1007</td>
</tr>
<tr>
<td>begin_polygon</td>
<td>1007</td>
</tr>
<tr>
<td>circle</td>
<td>1007</td>
</tr>
<tr>
<td>class_name</td>
<td>1007</td>
</tr>
<tr>
<td>clip_box</td>
<td>1007</td>
</tr>
<tr>
<td>locale_printf</td>
<td>1008</td>
</tr>
<tr>
<td>color</td>
<td>1008</td>
</tr>
<tr>
<td>curve</td>
<td>1008</td>
</tr>
<tr>
<td>descent</td>
<td>1008</td>
</tr>
<tr>
<td>draw</td>
<td>1008, 1009</td>
</tr>
<tr>
<td>draw_image</td>
<td>1010</td>
</tr>
<tr>
<td>draw_image_mono</td>
<td>1010</td>
</tr>
<tr>
<td>draw_scaled</td>
<td>1010</td>
</tr>
<tr>
<td>end_complex_polygon</td>
<td>1011</td>
</tr>
<tr>
<td>end_line</td>
<td>1011</td>
</tr>
<tr>
<td>end_loop</td>
<td>1011</td>
</tr>
<tr>
<td>end_points</td>
<td>1011</td>
</tr>
<tr>
<td>end_polygon</td>
<td>1011</td>
</tr>
<tr>
<td>font</td>
<td>1011</td>
</tr>
<tr>
<td>gap</td>
<td>1011</td>
</tr>
<tr>
<td>height</td>
<td>1011</td>
</tr>
<tr>
<td>line</td>
<td>1012</td>
</tr>
<tr>
<td>line_style</td>
<td>1012</td>
</tr>
<tr>
<td>loop</td>
<td>1012</td>
</tr>
<tr>
<td>not_clipped</td>
<td>1013</td>
</tr>
<tr>
<td>pie</td>
<td>1013</td>
</tr>
<tr>
<td>point</td>
<td>1013</td>
</tr>
<tr>
<td>polygon</td>
<td>1013</td>
</tr>
<tr>
<td>pop_clip</td>
<td>1013</td>
</tr>
<tr>
<td>push_clip</td>
<td>1014</td>
</tr>
<tr>
<td>push_no_clip</td>
<td>1014</td>
</tr>
<tr>
<td>rect</td>
<td>1014</td>
</tr>
<tr>
<td>rectf</td>
<td>1014</td>
</tr>
<tr>
<td>rtl_draw</td>
<td>1014</td>
</tr>
<tr>
<td>text_extents</td>
<td>1014</td>
</tr>
<tr>
<td>transformed_vertex</td>
<td>1015</td>
</tr>
<tr>
<td>vertex</td>
<td>1015</td>
</tr>
<tr>
<td>width</td>
<td>1015</td>
</tr>
<tr>
<td>xline</td>
<td>1015</td>
</tr>
<tr>
<td>xylne</td>
<td>1016</td>
</tr>
<tr>
<td>Fl_PostScript_Printer</td>
<td>1016</td>
</tr>
<tr>
<td>class_name</td>
<td>1020</td>
</tr>
<tr>
<td>start_job</td>
<td>1020</td>
</tr>
<tr>
<td>Fl_Preferences</td>
<td>1020</td>
</tr>
<tr>
<td>~Fl_Preferences</td>
<td>1025</td>
</tr>
<tr>
<td>deleteEntry</td>
<td>1025</td>
</tr>
<tr>
<td>deleteGroup</td>
<td>1026</td>
</tr>
<tr>
<td>entries</td>
<td>1026</td>
</tr>
<tr>
<td>entry</td>
<td>1026</td>
</tr>
<tr>
<td>entryExists</td>
<td>1026</td>
</tr>
<tr>
<td>Fl_Preferences</td>
<td>1023–1025</td>
</tr>
<tr>
<td>flush</td>
<td>1027</td>
</tr>
</tbody>
</table>
get, 1027–1029
getUserDataPath, 1030
group, 1030
groupExists, 1030
groups, 1031
ID, 1023
newUUID, 1031
Root, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
set, 1031–1033
size, 1033
SYSTEM, 1023
INDEX

FL_RGB_SCALING_NEAREST
  FL_image.H, 1687
fl_rmdir
  Unicode and UTF-8 functions, 262
Fl_Roller, 1100
draw, 1107
Fl_Roller, 1107
handle, 1107
Fl_Roller.H, 1730
fl_rotate
  Drawing functions, 247
  Fl_Graphics_Driver, 673
Fl_Round_Button, 1108
  Fl_Round_Button.H, 1115
Fl_Round_Button.H, 1730
Fl_Round_Clock, 1115
Fl_Round_Clock.H, 1731
fl_scale
  Drawing functions, 247
  Fl_Graphics_Driver, 673
FL_SCREEN_CONFIGURATION_CHANGED
  Enumerations.H, 1612
Fl_Scroll, 1122
  bbox, 1131
draw, 1131
Fl_Scroll, 1131
handle, 1131
  recalc_scrollbars, 1132
  resize, 1132
  scroll_to, 1133
  scrollbar_size, 1133
  xposition, 1134
  yposition, 1134
fl_scroll
  Drawing functions, 247
Fl_Scroll.H, 1731
Fl_Scroll::Fl_Region_LRTB, 1078
Fl_Scroll::Fl_Region_XYWH, 1078
Fl_Scroll::Fl_Scrollbar_Data, 1144
Fl_Scroll::ScrollInfo, 1592
Fl_Scrollbar, 1134
draw, 1142
Fl_Scrollbar, 1142
handle, 1143
linesize, 1143
value, 1143
Fl_Scrollbar.H, 1733
Fl_Secret_Input, 1144
  Fl_Secret_Input.H, 1735
Fl_Secret_Input.H, 1153
handle, 1153
Fl_Single_Window, 1182
  flush, 1192
  show, 1192
Fl_Single_Window.H, 1738
fl_size
  Color & Font functions, 224
Fl_Slider, 1192
  bounds, 1201
draw, 1201
Fl_Slider, 1200
handle, 1201
scrollvalue, 1201
slider_size, 1202
Fl_Slider.H, 1738
FL_SOLID
  Drawing functions, 232
Fl_Spinner, 1202
  Fl_Spinner, 1211
handle, 1211
maximum, 1211
minimum, 1211
resize, 1211
step, 1212
type, 1212

FL_SHARED_IMAGE, 1166
  ~Fl_Shared_Image, 1169
  add, 1169
color_average, 1170
compare, 1170
copy, 1170
desaturate, 1170
draw, 1171
find, 1171
Fl_Shared_Image, 1169
get, 1171, 1172
original, 1172
recount, 1172
release, 1172
scale, 1173
scaling_algorithm, 1173
uncache, 1173
Fl_Shared_Image.H, 1734, 1735
fl_register_images, 1735
FL_SHORTCUT
  Enumerations.H, 1611
fl_shortcut_label
  Drawing functions, 248, 249
FL_SHOW
  Enumerations.H, 1612
fl_show_colormap
  Color & Font functions, 223
fl_show_colormap.H, 1736
fl_show_input.H, 1737
Fl_SIMPLE_COUNTER, 1174
Fl_SIMPLE_COUNTER.H, 1737
Fl_SINGLE_WINDOW, 1182
  flush, 1192
  show, 1192
Fl_SINGLE_WINDOW.H, 1738
fl_size
  Color & Font functions, 224
Fl_SLIDER, 1192
  bounds, 1201
draw, 1201
Fl_SLIDER, 1200
handle, 1201
scrollvalue, 1201
slider_size, 1202
Fl_SLIDER.H, 1738
FL_SOLID
  Drawing functions, 232
Fl_SPINNER, 1202
  Fl_SPINNER, 1211
handle, 1211
maximum, 1211
minimum, 1211
resize, 1211
step, 1212
type, 1212
value, 1212
Fl_Spinner.H, 1739
fl_stat
  Unicode and UTF-8 functions, 262
Fl_String
  fl_types.h, 1782
FL_SUBMENU
  Fl_Menu_Item.H, 1701
FL_SUBMENU_POINTER
  Fl_Menu_Item.H, 1701
Fl_Surface_Device, 1212
  class_name, 1214
  set_current, 1214
  surface, 1214
Fl_Sys_Menu_Bar, 1214
  add, 1224
  clear, 1225
  clear_submenu, 1225
  draw, 1225
  Fl_Sys_Menu_Bar, 1224
  insert, 1225
  menu, 1226
  mode, 1226
  remove, 1226
  replace, 1226
Fl_Sys_Menu_Bar.H, 1741
fl_system
  Unicode and UTF-8 functions, 263
Fl_System_Printer, 1227
  class_name, 1230
  end_job, 1230
  end_page, 1230
  margins, 1230
  origin, 1230, 1231
  printable_rect, 1231
  rotate, 1231
  scale, 1231
  start_job, 1232
  start_page, 1232
  translate, 1232
  untranslate, 1232
Fl_Table, 1233
  ~Fl_Table, 1247
  callback, 1247
  callback_col, 1248
  callback_context, 1248
  callback_row, 1248
  child, 1248
  children, 1248
  clear, 1248
  col_header, 1248
  col_resize, 1249
  col_resize_min, 1249
  col_width, 1249
  col_width_all, 1249
  CONTEXT_CELL, 1246
  CONTEXT_COL_HEADER, 1246
  CONTEXT_ENDPAGE, 1246
  CONTEXT_NONE, 1246
  CONTEXT_RC_RESIZE, 1246
  CONTEXT_ROW_HEADER, 1246
  CONTEXT_STARTPAGE, 1246
  CONTEXT_TABLE, 1246
draw, 1249
  draw_cell, 1249
  Fl_Table, 1249
  get_selection, 1251
  handle, 1251
  is_interactive_resize, 1251
  is_selected, 1252
  resize, 1252
  row_header, 1252
  row_height, 1252
  row_height_all, 1252
  row_resize, 1252
  row_resize_min, 1252
  rows, 1253
  scrollbar_size, 1253
  set_selection, 1253
  tab_cell_nav, 1254
  table_box, 1254
  TableContext, 1246
top_row, 1254
  visible_cells, 1255
  when, 1255
Fl_Table.H, 1743
Fl_Table_Row, 1255
  ~Fl_Table_Row, 1267
  clear, 1267
  Fl_Table_Row, 1267
  handle, 1267
  row_selected, 1268
  rows, 1268
  select_all_rows, 1268
  select_row, 1268
type, 1268
Fl_Table_Row.H, 1749
Fl_Tabs, 1269
  client_area, 1279
draw, 1280
  Fl_Tabs, 1279
  handle, 1280
push, 1280
  value, 1281
  which, 1281
Fl_Tabs.H, 1751
Fl_Text_Buffer, 1281
  add_modify_callback, 1286
  address, 1286, 1287
append, 1287
appendfile, 1287
byte_at, 1287
char_at, 1288
copy, 1288
  count_displayed_characters, 1288
  count_lines, 1288
  generated by Doxygen
fl_transform_dy
Drawing functions, 250
Fl_Graphics_Driver, 674

fl_transform_x
Drawing functions, 250
Fl_Graphics_Driver, 674

fl_transform_y
Drawing functions, 250
Fl_Graphics_Driver, 674

fl_transformed_vertex
Drawing functions, 250
Fl_Graphics_Driver, 674

fl_translate
Drawing functions, 250
Fl_Graphics_Driver, 675

Fl_Tree, 1394
add, 1411
calc_dimensions, 1413
calc_tree, 1413
callback_item, 1414
callback_reason, 1414
clear, 1414
clear_children, 1414
close, 1414, 1415
closeicon, 1415
connectorstyle, 1416
deselect, 1416
deselect_all, 1417
display, 1417
displayed, 1417
draw, 1418
extend_selection, 1418
extend_selection_dir, 1418
find_clicked, 1419
find_item, 1419
first, 1420
first_selected_item, 1420
first_visible, 1420
first_visible_item, 1420
get_selected_items, 1421
handle, 1421
hposition, 1421, 1422
insert, 1422
insert_above, 1423
is_close, 1423
is_hscroll_visible, 1424
is_open, 1424
is_scrollbar, 1424
is_selected, 1425
is_vscroll_visible, 1425
item_clicked, 1426
item_draw_mode, 1426
item_labelbgcolor, 1426, 1427
item_labelfgcolor, 1427
item_labelfont, 1427
item_labelsize, 1427
item_pathname, 1427
item_reselect_mode, 1428
last, 1428
last_selected_item, 1428
last_visible, 1428
last_visible_item, 1429
load, 1429
next, 1429
next_item, 1429
next_selected_item, 1430
next_visible_item, 1431
open, 1431, 1432
open_toggle, 1433
openicon, 1433
prev, 1433
recalc_tree, 1434
remove, 1434
resize, 1434
root, 1434
root_label, 1434
scrollbar_size, 1435
select, 1435, 1436
select_all, 1436
select_only, 1437
select_toggle, 1437
selectbox, 1438
selectmode, 1438
set_item_focus, 1438
show_item, 1438, 1439
show_item_bottom, 1439
show_item_middle, 1439
show_item_top, 1439
show_self, 1439
showcollapse, 1440
showroot, 1440
sortorder, 1440
vposition, 1441

Fl_Tree.H, 1765, 1766
Fl_Tree_Reason, 1766
FL_TREE_REASON_CLOSED, 1766
FL_TREE_REASON_DESELECTED, 1766
FL_TREE_REASON_DRAGGED, 1766
FL_TREE_REASON_NONE, 1766
FL_TREE_REASON_OPENED, 1766
FL_TREE_REASON_RESELECTED, 1766
FL_TREE_REASON_SELECTED, 1766

Fl_Tree_Connector
Fl_Tree_Prefs.H, 1776
FL_TREE_CONNECTOR_DOTTED
FL_TREE_CONNECTOR_NONE
FL_TREE_CONNECTOR_SOLID
Fl_Tree_Prefs.H, 1776

Fl_Tree_Item, 1441
activate, 1447
add, 1447, 1448
calc_item_height, 1448
child, 1448
INDEX

FL_TREE_REASON_DESELECTED
  Fl_Tree.H, 1766
FL_TREE_REASON_DRAugged
  Fl_Tree.H, 1766
FL_TREE_REASON_NONE
  Fl_Tree.H, 1766
FL_TREE_REASON_OPENED
  Fl_Tree.H, 1766
FL_TREE_REASON_RESELECTED
  Fl_Tree.H, 1766
FL_TREE_REASON_SELECTED
  Fl_Tree.H, 1766
Fl_Tree_Select
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECT_MULTI
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECT_NONE
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECT_SINGLE
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECT_SINGLE_DRAGGABLE
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECTABLE_ALWAYS
  Fl_Tree_Prefs.H, 1777
FL_TREE_SELECTABLE_ONCE
  Fl_Tree_Prefs.H, 1777
Fl_Tree_Sort
  Fl_Tree_Prefs.H, 1777
FL_TREE_SORT_ASCENDING
  Fl_Tree_Prefs.H, 1777
FL_TREE_SORT_DESCENDING
  Fl_Tree_Prefs.H, 1777
FL_TREE_SORT_NONE
  Fl_Tree_Prefs.H, 1777
fl_types.h, 1781, 1782
  Fl_CString, 1782
  Fl_String, 1782
fl_xml_toUtf8
  Unicode and UTF-8 functions, 263
FL_UNFOCUS
  Enumerations.H, 1610
fl_unlink
  Unicode and UTF-8 functions, 263
FL_UP_BOX
  Enumerations.H, 1607
FL_UP_FRAME
  Enumerations.H, 1607
fl_utf8.h, 1783, 1784
fl_utf8back
  Unicode and UTF-8 functions, 264
fl_utf8bytes
  Unicode and UTF-8 functions, 264
fl_utf8decode
  Unicode and UTF-8 functions, 264
fl_utf8encode
  Unicode and UTF-8 functions, 264
fl_utf8from_mb
  Unicode and UTF-8 functions, 265
fl_utf8froma
  Unicode and UTF-8 functions, 265
fl_utf8fromwc
  Unicode and UTF-8 functions, 265
fl_utf8fwd
  Unicode and UTF-8 functions, 266
fl_utf8len
  Unicode and UTF-8 functions, 266
fl_utf8len1
  Unicode and UTF-8 functions, 266
fl_utf8locale
  Unicode and UTF-8 functions, 266
fl_utf8test
  Unicode and UTF-8 functions, 266
fl_utf8to_mb
  Unicode and UTF-8 functions, 267
fl_utf8toa
  Unicode and UTF-8 functions, 267
fl_utf8tou8
  Unicode and UTF-8 functions, 267
fl_utf8toUtf16
  Unicode and UTF-8 functions, 267
fl_utf8towc
  Unicode and UTF-8 functions, 268
fl_utf_strcasecmp
  Unicode and UTF-8 functions, 268
fl_utf_strncasecmp
  Unicode and UTF-8 functions, 268
fl_utf_tolower
  Unicode and UTF-8 functions, 269
fl_utf_toupper
  Unicode and UTF-8 functions, 269
Fl_Valuator, 1470
  Fl_Valuator, 1477
    format, 1477
    increment, 1478
    maximum, 1478
    minimum, 1478
    precision, 1478
    range, 1478
    round, 1479
    step, 1479
    value, 1479
    value_damage, 1479
Fl_Valuator.H, 1787
Fl_Value_Input, 1479
  cursor_color, 1487
  draw, 1488
  Fl_Value_Input, 1487
    handle, 1488
    resize, 1488
    shortcut, 1489
    soft, 1489
    textcolor, 1489
    textfont, 1489
    textsize, 1489, 1490
Fl_Value_Input.H, 1788
Fl_Value_Output, 1490
  draw, 1498
  Fl_Value_Output, 1497

Generated by Doxygen
Fl_Menu_Window, 891
Fl_Overlay_Window, 956
Fl_Preferences, 1027
Fl_Single_Window, 1192
Fl_Window, 1560
focus
Events handling functions, 208
Fl_Group, 686
font
Fl_GDI_Graphics_Driver, 607
Fl_Graphics_Driver, 660
Fl_PostScript_Graphics_Driver, 1011
Fl_Quartz_Graphics_Driver, 1056
Fi_Tooltip, 1393
Fl_Xlib_Graphics_Driver, 1586
FORCE_POSITION
Fl_Widget, 1515
force_position
Fi_Window, 1560
format
Fl_Valuator, 1477
format_char
Fl_Browser, 351, 352
Forms Compatibility, 129
forms.H, 1801
free_color
Color & Font functions, 225
free_icons
Fl_Window, 1560
free_position
Fi_Window, 1560
freenglut_teapot_data.h, 1874
full_height
Fi_Browser, 352
Fi_Browser, 377
full_width
Fi_Browser, 378
FULLSCREEN
Fl_Widget, 1515
fullscreen
Fi_Window, 1560
fullscreen_screens
Fi_Window, 1561
g
Fl_Color_Chooser, 474
gap
Fl_Graphics_Driver, 660
Fl_PostScript_Graphics_Driver, 1011
gb2312.h, 2076
georgian_academy.h, 2106
georgian_ps.h, 2107
get
Fi_Preferences, 1027–1029
Fl_Shared_Image, 1171, 1172
get_absolute_top_line_number
Fi_Text_Display, 1319
get_color
Color & Font functions, 225
get_font
Color & Font functions, 226
get_font_name
Color & Font functions, 226
get_font_sizes
Color & Font functions, 226
get_key
Events handling functions, 209
get_mouse
Events handling functions, 209
get_selected_items
Fi_Tree, 1421
get_selection
Fi_Table, 1251
get_system_colors
Fi, 305
getUserdataPath
Fi_Preferences, 1030
gl.h, 1810, 1813
gl_color, 1811
gl_draw, 1811–1813
gl_rect, 1813
gl_rectf, 1813
gl2opengl.h, 1814
gl_color
gl.h, 1811
gl_draw
gl.h, 1811–1813
gl_draw.H, 1815
gl_rect
gl.h, 1813
gl_rectf
gl.h, 1813
gl_texture_pile_height
Mac OS X-specific symbols, 271
gl_visual
Fi, 305
global
Fi_Menu_. 844
global_key_bindings
Fi_Text_Editor, 1358
glu.h, 1815
GLUT Compatibility, 126
glut.H, 1816
grab
Windows handling functions, 197
group
Fi_Preferences, 1030
GROUP_RELATIVE
Fi_Widget, 1515
groupExists
Fi_Preferences, 1030
groups
Fi_Preferences, 1031
h
Fi_Widget, 1527, 1528
handle
Events handling functions, 209
INDEX

```
FL_Adjuster, 320
FL_Box, 333
FL_Browser_, 378
FL_Button, 395
FL_Clock, 456
FL_Counter, 488
FL_Dial, 500
FL_File_Input, 548
FL_Free, 600
FL_Gl_Window, 630
FL_Glut_Window, 647
FL_Group, 686
FL_HELP_View, 705
FL_Input, 768
FL_Light_Button, 822
FL_Menu_Bar, 858
FL_Menu_Button, 868
FL_Positioner, 994
FL_REPEAT_Button, 1085
FL_Return_Button, 1094
FL_Roller, 1107
FL_Scroll, 1131
FL_Scrollbar, 1143
FL_Secret_Input, 1153
FL_Slider, 1201
FL_Spinner, 1211
FL_Table, 1251
FL_Table_Row, 1267
FL_Tabs, 1280
FL_Text_Display, 1319
FL_Text_Editor, 1353
FL_Tile, 1370
FL_Timer, 1382
FL_Tree, 1421
FL_Value_Input, 1488
FL_Value_Output, 1498
FL_Value_Slider, 1507
FL_Widget, 1528
FL_Window, 1561

handle_
Events handling functions, 209
handle_mouse
FL_Input, 780
handle_vline
FL_Text_Display, 1319
handletext
FL_Input, 780
Handling Events, 54
has_scrollbar
FL_Browser, 378
HEAVY_CURSOR
FL_Text_Display, 1311
height
FL_GDI_Graphics_Driver, 607
FL_Graphics_Driver, 660
FL_PostScript_Graphics_Driver, 1011
FL_QUARTZ_Graphics_Driver, 1056
FL_XLIB_Graphics_Driver, 1586

help
FL, 311

hide
FL_Browser, 352
FL_Double_Window, 513
FL_GL_Window, 630
FL_Menu_Window, 891
FL_Overlay_Window, 956
FL_Widget, 1528
FL_Window, 1561

hide_widgets
FL_Tree_Item, 1453

highlight
FL_Text_Buffer, 1289
highlight_data
FL_Text_Display, 1320
highlight_text
FL_Text_Buffer, 1289

highres_image
FL_Image_Surface, 758

HORIZONTAL
FL_Browser, 375
HORIZONTAL_ALWAYS
FL_Browser, 375

hotspot
FL_Window, 1562

hour
FL_Clock_Output, 464

hoverdelay
FL_Tooltip, 1393

hposition
FL_Browser, 378, 379
FL_Tree, 1421, 1422

hscrollbar
FL_Browser, 387

hsv
FL_Color_Chooser, 474

hsv2rgb
FL_Color_Chooser, 475

hue
FL_Color_Chooser, 475

color
FL_Browser, 352, 353
FL_Window, 1562

iconize
FL_Window, 1562

icons
FL_Window, 1563

iconsize
FL_File_Browser, 527
FL_File_Chooser, 532, 533

ID
FL_Preferences, 1023

delay
FL, 311

image
```
INDEX

Fl_Image_Surface, 758
Fl_Widget, 1528, 1529
imKStoUCS.c, 1944
in_selection
    Fl_Text_Display, 1321
INACTIVE
    Fl_Widget, 1515
inactive
    Fl_Image, 754
incr_height
    Fl_Browser, 353
    Fl_Browser_, 379
increment
    Fl_Valuator, 1478
index
    Fl_Input_, 780
init_sizes
    Fl_Group, 686
input
    Fl_Input_Choice, 800
input_type
    Fl_Input_, 780, 781
insert
    Fl_Browser, 353, 354
    Fl_Chart, 418
    Fl_Group, 687
    Fl_Input_, 781
    Fl_Menu_, 845
    Fl_Menu_Item, 876
    Fl_Sys_Menu_Bar, 1225
    Fl_Text_Buffer, 1289
    Fl_Text_Display, 1321
    Fl_Tree, 1422
    Fl_Tree_Item, 1453
    Fl_Tree_Item_Array, 1463
insert_
    Fl_Text_Buffer, 1290
insert_above
    Fl_Tree, 1423
    Fl_Tree_Item, 1453
insert_mode
    Fl_Text_Editor, 1353
insert_position
    Fl_Text_Display, 1321
insertfile
    Fl_Text_Buffer, 1290
inserting
    Fl_Browser_, 379
inside
    Fl_Widget, 1529
Introduction to FLTK, 4
INVISIBLE
    Fl_Widget, 1515
is_close
    Fl_Tree, 1423
is_hscroll_visible
    Fl_Tree, 1424
is_interactive_resize
    Fl_Table, 1251
is_label_copied
    Fl_Widget, 1529
is_open
    Fl_Tree, 1424
is_scheme
    Fl, 305
is_scrollbar
    Fl_Tree, 1424
is_selected
    Fl_Table, 1252
    Fl_Tree, 1425
is_vscroll_visible
    Fl_Tree, 1425
iso8859_1.h, 2108
iso8859_10.h, 2108
iso8859_11.h, 2110
iso8859_13.h, 2111
iso8859_14.h, 2112
iso8859_15.h, 2113
iso8859_16.h, 2114
iso8859_2.h, 2115
iso8859_3.h, 2116
iso8859_4.h, 2118
iso8859_5.h, 2119
iso8859_6.h, 2120
iso8859_7.h, 2121
iso8859_8.h, 2122
iso8859_9.h, 2123
iso8859_9e.h, 2124
item_at
    Fl_Browser, 354
    Fl_Browser_, 379
item_clicked
    Fl_Tree, 1426
item_draw
    Fl_Browser, 354
    Fl_Browser_, 380
item_draw_mode
    Fl_Tree, 1426
    Fl_Tree_Prefs, 1468
item_first
    Fl_Browser, 355
    Fl_Browser_, 380
item_height
    Fl_Browser, 355
    Fl_Browser_, 380
item_labelbgcolor
    Fl_Tree, 1426, 1427
    Fl_Tree_Prefs, 1468
item_labelbgcolor
    Fl_Tree, 1427
item_labelfont
    Fl_Tree, 1427
item_labelsize
    Fl_Tree, 1427
item_last
    Fl_Browser, 355
INDEX

Fl_Browser, 380
item_next
\hspace{1cm} Fl_Browser, 355
\hspace{1cm} Fl_Browser, 380
item_pathname
\hspace{1cm} Fl_Menu_Item, 845
\hspace{1cm} Fl_Tree, 1427
item_prev
\hspace{1cm} Fl_Browser, 356
\hspace{1cm} Fl_Browser, 381
item_quick_height
\hspace{1cm} Fl_Browser, 381
item_reselect_mode
\hspace{1cm} Fl_Tree, 1428
item_select
\hspace{1cm} Fl_Browser, 356
\hspace{1cm} Fl_Browser, 381
item_selected
\hspace{1cm} Fl_Browser, 357
\hspace{1cm} Fl_Browser, 381
item_swap
\hspace{1cm} Fl_Browser, 357
\hspace{1cm} Fl_Browser, 382
item_text
\hspace{1cm} Fl_Browser, 357
\hspace{1cm} Fl_Browser, 382
item_width
\hspace{1cm} Fl_Browser, 357
\hspace{1cm} Fl_Browser, 382

jisx0201.h, 2125
jisx0208.h, 2126
jisx0212.h, 2153
kf_backspace
\hspace{3cm} Fl_Text_Editor, 1353
kf_c_s_move
\hspace{3cm} Fl_Text_Editor, 1353
kf_copy
\hspace{3cm} Fl_Text_Editor, 1354
kf_ctrl_move
\hspace{3cm} Fl_Text_Editor, 1354
kf_cut
\hspace{3cm} Fl_Text_Editor, 1354
kf_default
\hspace{3cm} Fl_Text_Editor, 1354
kf_delete
\hspace{3cm} Fl_Text_Editor, 1354
kf_down
\hspace{3cm} Fl_Text_Editor, 1354
kf_end
\hspace{3cm} Fl_Text_Editor, 1355
kf_enter
\hspace{3cm} Fl_Text_Editor, 1355
kf_home
\hspace{3cm} Fl_Text_Editor, 1355
kf_ignore
\hspace{3cm} Fl_Text_Editor, 1355
kf_insert
\hspace{3cm} Fl_Text_Editor, 1355

Fl_Text_Editor, 1355
kf_left
\hspace{1cm} Fl_Text_Editor, 1355
kf_m_s_move
\hspace{1cm} Fl_Text_Editor, 1355
kf_meta_move
\hspace{1cm} Fl_Text_Editor, 1356
kf_move
\hspace{1cm} Fl_Text_Editor, 1356
kf_page_down
\hspace{1cm} Fl_Text_Editor, 1356
kf_page_up
\hspace{1cm} Fl_Text_Editor, 1356
kf_paste
\hspace{1cm} Fl_Text_Editor, 1356
kf_right
\hspace{1cm} Fl_Text_Editor, 1356
kf_select_all
\hspace{1cm} Fl_Text_Editor, 1357
kf_shift_move
\hspace{1cm} Fl_Text_Editor, 1357
kf_undo
\hspace{1cm} Fl_Text_Editor, 1357
kf_up
\hspace{1cm} Fl_Text_Editor, 1357
koi8_c.h, 2178
koi8_r.h, 2180
koi8_u.h, 2181
ksc5601.h, 2183

label
\hspace{1cm} Fl_Bitmap, 324
\hspace{1cm} Fl_File_Icon, 536
\hspace{1cm} Fl_Image, 754
\hspace{1cm} Fl_Menu_Item, 876
\hspace{1cm} Fl_Pixmap, 976, 977
\hspace{1cm} Fl_RGB_Image, 1099
\hspace{1cm} Fl_Tree_Item, 1453
\hspace{1cm} Fl_Widget, 1529, 1530
label_h
\hspace{1cm} Fl_Tree_Item, 1453
label_shortcut
\hspace{1cm} Fl_Widget, 1530
label_w
\hspace{1cm} Fl_Tree_Item, 1454
label_x
\hspace{1cm} Fl_Tree_Item, 1454
label_y
\hspace{1cm} Fl_Tree_Item, 1454
labela
\hspace{1cm} Fl_Multi_Label, 905
labelb
\hspace{1cm} Fl_Multi_Label, 905
labelbgcolor
\hspace{1cm} Fl_Tree_Item, 1454
labelcolor
\hspace{1cm} Fl_Menu_Item, 877
\hspace{1cm} Fl_Widget, 1531
labelfont

Generated by Doxygen
Fl_Menu_Item, 877
Fl_Widget, 1531
labels
Fl_Widget, 1532
labeltype
Fl_File_Icon, 536
Fl_Menu_Item, 877
Fl_Widget, 1532
LANDSCAPE
Fl_Paged_Device, 968
last
Fl_Tree, 1428
last_selected_item
Fl_Tree, 1428
last_visible
Fl_Tree, 1428
last_visible_item
Fl_Tree, 1429
ld
Fl_Image, 755
leftedge
Fl_Browser, 382
leftline
Fl_Help_View, 705
length
Fl_Text_Buffer, 1290
LETTER
Fl_Paged_Device, 968
line
Fl_Graphics_Driver, 660
Fl_PostScript_Graphics_Driver, 1012
line_end
Fl_Input, 781
Fl_Text_Buffer, 1290
Fl_Text_Display, 1322
line_start
Fl_Input, 781
Fl_Text_Buffer, 1291
Fl_Text_Display, 1322
line_style
Fl_Graphics_Driver, 661
Fl_PostScript_Graphics_Driver, 1012
line_text
Fl_Text_Buffer, 1291
lineno
Fl_Browser, 358
linenumber
Fl_Text_Display, 1322
linenumber_align
Fl_Text_Display, 1322
linenumberbgcolor
Fl_Text_Display, 1323
linenumberbgcolor
Fl_Text_Display, 1323
linenumberformat
Fl_Text_Display, 1323
linenumberformat
Fl_Text_Display, 1323
linenumber_width
Fl_Text_Display, 1323
lineposition
Fl_Browser, 358
linesize
Fl_Scrollbar, 1143
link
Fl_Help_View, 705
load
Fl_Browser, 358
Fl_File_Browser, 527
Fl_File_Icon, 537
Fl_Help_Dialog, 692
Fl_Help_View, 705
Fl_Plugin_Manager, 981
Fl_Tree, 1429
load_fti
Fl_File_Icon, 537
load_image
Fl_File_Icon, 537
load_system_icons
Fl_File_Icon, 538
loadfile
Fl_Text_Buffer, 1291
lock
Fl_Gl_Window, 630
Fl_Window, 1563
make_current
Fl_Gl_Window, 630
Fl_Window, 1563
make_overlay_current
Fl_Gl_Window, 630
make_visible
Fl_Browser, 359
manage_item_destroy
Fl_Tree_Item_Array, 1463
margin_height
Fl_Tooltip, 1393
margin_width
Fl_Tooltip, 1393

Mac OS X-specific symbols, 270
fl_mac_quit_early, 271
fl_mac_set_about, 271
gl_texture_pile_height, 271
mac.H, 1821, 1822
MAC_USE_ACCENTS_MENU
Fl_Widget, 1515
maintain_absolute_top_line_number
Fl_Text_Display, 1324
maintaining_absolute_top_line_number
Fl_Text_Display, 1324
make_current
Fl_Gl_Window, 630
Fl_Window, 1563
make_overlay_current
Fl_Gl_Window, 630
make_visible
Fl_Browser, 359
manage_item_destroy
Fl_Tree_Item_Array, 1463
margin_height
Fl_Tooltip, 1393
margin_width
Fl_Tooltip, 1393
marginbottom
  Fl_Tree_Prefs, 1468
margins
  Fl_Paged_Device, 969
  Fl_PostScript_File_Device, 998
  Fl_Printer, 1038
  Fl_System_Printer, 1230
mark
  Fl_Input_, 782
math.h, 1826
max_size
  Fl_RGB_Image, 1099
maximum
  Fl_PROGRESS, 1048
  Fl_VAulator, 1478
maximum_size
  Fl_Input_, 782
maximum
  Fl_Spinner, 1211
maxsize
  Fl_Chart, 418
measure
  Fl_Label, 814
  Fl_Menu_Item, 877
measure_deleted_lines
  Fl_Text_Display, 1324
measure_label
  Fl_Widget, 1532
measure_proportional_character
  Fl_Text_Display, 1324
measure_vline
  Fl_Text_Display, 1325
mediumarrow.h, 1876
menu
  Fl_Menu_, 846
  Fl_Sys_Menu_Bar, 1226
MENU_WINDOW
  Fl_Widget, 1515
menubutton
  Fl_Input_Choice, 800
middleline
  Fl_Browser, 359
Migrating Code from FLTK 1.0 to 1.1, 146
Migrating Code from FLTK 1.1 to 1.3, 147
minimum
  Fl_PROGRESS, 1048, 1049
  Fl_VAulator, 1478
minimum
  Fl_Spinner, 1211
minute
  Fl_Clock_Output, 464
mk_wcwidth.c, 2226
MODAL
  Fl_Widget, 1515
modal
  Fl_Window, 1563
  Windows handling functions, 197
mode
  Fl_Color_Chooser, 475
  Fl_GI_Window, 630, 631
  Fl_Menu_, 846
  Fl_Sys_Menu_Bar, 1226
move
  Fl_Browser, 359
  Fl_Tree_Item, 1454, 1455
  Fl_Tree_Item_Array, 1463
move_above
  Fl_Tree_Item, 1455
move_below
  Fl_Tree_Item, 1455
move_down
  Fl_Text_Display, 1325
move_into
  Fl_Tree_Item, 1455
move_left
  Fl_Text_Display, 1325
move_right
  Fl_Text_Display, 1325
move_up
  Fl_Text_Display, 1326
mTabDist
  Fl_Text_Display, 1326
mulelao.h, 2218
Multithreading support functions, 251
  awake, 251
  lock, 252
  thread_message, 252
  unlock, 252
mvalue
  Fl_Menu_, 846
Name
  Fl_Preferences::Name, 1591
names.h, 1826
nchecked
  Fl_Check_Browser, 430
NEW_FOLDER
  Fl_Native_File_Chooser, 925
new_list
  Fl_Browser_, 383
newUUID
  Fl_Preferences, 1031
next
  Fl_File_Icon, 538
  Fl_Menu_Item, 878
  Fl_Tree, 1429
  Fl_Tree_Item, 1455
  Fl_Wizard, 1576
next_char
  Fl_Text_Buffer, 1291
next_displayed
  Fl_Text_Display, 1325
next_item
  Fl_Tree, 1429
next_selected_item
  Fl_Tree, 1430
next_sibling
  Fl_Tree, 1430
next_visible
  Fl_Tree_Item, 1456
next_visible_item
  Fl_Tree, 1431
next_window
  Windows handling functions, 197
nitems
  Fl_Check_Browser, 430
NO_OPTIONS
  Fl_Native_File_Chooser, 925
NO_OVERLAY
  Fl_Widget, 1515
NOBORDER
  Fl_Widget, 1515
NON_MODAL
  Fl_Widget, 1515
NORMAL_CURSOR
  Fl_Text_Display, 1311
not_clipped
  Fl_Graphics_Driver, 661
  Fl_PostScript_Graphics_Driver, 1013
offset_line_starts
  Fl_Text_Display, 1326
open
  Fl_Tree, 1431, 1432
open_toggle
  Fl_Tree, 1433
opendeficon
  Fl_Tree_Prefs, 1468
openicon
  Fl_Tree, 1433
  Fl_Tree_Prefs, 1468
Operating System Issues, 133
Option
  Fl_Native_File_Chooser, 925
option
  Fl, 306, 307
OPTION_ARROW_FOCUS
  Fl, 299
OPTION_DND_TEXT
  Fl, 299
OPTION_FNFC_USES_GTK
  Fl, 299
OPTION_LAST
  Fl, 299
OPTION_SHOW_TOOLTIP
  Fl, 299
OPTION_VISIBLE_FOCUS
  Fl, 299
options
  Fl_Native_File_Chooser, 927
ORIENTATION
  Fl_Paged_Device, 969
origin
  Fl_Paged_Device, 969, 970
  Fl_PostScript_File_Device, 999
  Fl_Printer, 1038, 1039

Fi_System_Printer, 1230, 1231
original
  Fl_Shared_Image, 1172
ortho
  Fl_Gl_Window, 632
OUTPUT
  Fl_Widget, 1515
output
  Fl_Widget, 1533
outputfile
  Fl_Text_Buffer, 1291
OVERRIDE
  Fl_Widget, 1515
overstrike
  Fl_Text_Display, 1326
own_colormap
  Fl, 307
Page_Format
  Fl_Paged_Device, 968
Page_Layout
  Fl_Paged_Device, 968
parent
  Fl_Tree_Item, 1456
  Fl_Widget, 1533
paste
  Selection & Clipboard functions, 213
picked
  Fl_Menu_, 847
pie
  Fl_Graphics_Driver, 661
  Fl_PostScript_Graphics_Driver, 1013
pixel_h
  Fl_Gl_Window, 632
pixel_w
  Fl_Gl_Window, 632
pixels_per_unit
  Fl_Gl_Window, 632
Pixmap
  Fl_FormsPixmap, 587
platform.H, 1827
point
  Fl_Graphics_Driver, 661
  Fl_PostScript_Graphics_Driver, 1013
pop_clip
  Fl_Graphics_Driver, 662
  Fl_PostScript_Graphics_Driver, 1013
polygon
  Fl_Graphics_Driver, 662
  Fl_PostScript_Graphics_Driver, 1013
popup
  Fl_Menu_Button, 868
  Fl_Menu_Item, 878
POPUP1
  Fl_Menu_Button, 867
POPUP12
  Fl_Menu_Button, 867
POPUP123
  Fl_Menu_Button, 868

Generated by Doxygen
INDEX

POPPUP13
  Fl_Menu_Button, 867
POPPUP2
  Fl_Menu_Button, 867
POPPUP23
  Fl_Menu_Button, 867
POPPUP3
  Fl_Menu_Button, 867
popup_buttons
  Fl_Menu_Button, 867
PORTORAT
  Fl_Paged_Device, 968
position
  Fl_Browser_, 383
  Fl_Input_, 782, 783
  Fl_Text_Selection, 1359
  Fl_Tile, 1371
  Fl_Widget, 1533
position_style
  Fl_Text_Display, 1326
position_to_line
  Fl_Text_Display, 1327
position_to_linecol
  Fl_Text_Display, 1327
position_to_xy
  Fl_Text_Display, 1327
precision
  Fl_Valuator, 1478
Preface, 2
prefs
  Fl_Tree_Item, 1456
preset_file
  Fl_Native_FileChooser, 928
prev
  Fl_Tree, 1433
  Fl_Tree_Item, 1456
prev_char
  Fl_Text_Buffer, 1292
prev_displayed
  Fl_Tree_Item, 1457
prev_sibling
  Fl_Tree_Item, 1457
prev_visible
  Fl_Tree_Item, 1457
PREVIEW
  Fl_Native_FileChooser, 925
preview
  Fl_FileChooser, 533
print
  Fl_DevicePlugin, 491
  Fl_Mac_App_Menu, 830
print_panel.h, 1876
print_widget
  Fl_Paged_Device, 970
  Fl_Printer, 1039
print_window
  Fl_Paged_Device, 970
print_window_part
  Fl_Paged_Device, 971
  Fl_Printer, 1039
printable_rect
  Fl_Paged_Device, 971
  Fl_PostScript_FileDevice, 999
  Fl_Printer, 1040
  Fl_System_Printer, 1231
Programming with FLUID, 75
pulldown
  Fl_Menu_Item, 878
push
  Fl_Tabs, 1280
push_clip
  Fl_GraphicsDriver, 662
  Fl_PostScript_GraphicsDriver, 1014
push_no_clip
  Fl_GraphicsDriver, 662
  Fl_PostScript_GraphicsDriver, 1014
pushed
  Events handling functions, 210
r
  Fl_ColorChooser, 475
radio
  Fl_Menu_Item, 878
range
  Fl_Valuator, 1478
readonly
  Fl_Input_, 783, 784
readqueue
  Fl, 307
ready
  Fl, 307
recalc_scrollbars
  Fl_Scroll, 1132
recalc_tree
  Fl_Tree, 1434
  Fl_Tree_Item, 1457
rect
  Fl_GraphicsDriver, 662
  Fl_PostScript_GraphicsDriver, 1014
rectangle_capture
  Fl_DevicePlugin, 491
rectf
  Fl_GraphicsDriver, 663
  Fl_PostScript_GraphicsDriver, 1014
redisplay_range
  Fl_Text_Display, 1328
redraw
  Fl_Widget, 1534
redraw_label
  Fl_Widget, 1534
redraw_line
  Fl_BROWSER_, 383
redraw_lines
  Fl_Browser_, 384
redraw_overlay
  Fl_GL_Window, 632
  Fl_Overlay_Window, 956

Generated by Doxygen
INDEX 2281

Fl_Graphics_Driver, 663
Fl_PostScript_Graphics_Driver, 1014
Fl_Quartz_Graphics_Driver, 1056
Fl_Xlib_Graphics_Driver, 1586

run
Fl, 308

Safe widget deletion support functions, 252
clear_widget_pointer, 253
delete_widget, 253
do_widget_deletion, 254
release_widget_pointer, 254
watch_widget_pointer, 254

saturation
Fl_Color_Chooser, 476
SAVEAS_CONFIRM
Fl_Native_File_Chooser, 925

savefile
Fl_Text_Buffer, 1293

scale
Fl_Paged_Device, 971
Fl_PostScript_File_Device, 1000
Fl_Printer, 1040
Fl_Shared_Image, 1173
Fl_System_Printer, 1231

scaling_algorithm
Fl_Shared_Image, 1173

scandir_posix.c, 1876

scheme
Fl, 309

Screen functions, 215
screen_dpi, 215
screen_num, 216
screen_work_area, 216, 217
screen_xywh, 217, 218

screen_dpi
Screen functions, 215

screen_num
Screen functions, 216

screen_work_area
Screen functions, 216, 217

screen_xywh
Screen functions, 217, 218

scroll
Fl_Text_Display, 1329

scroll
Fl_Text_Display, 1329
scroll_timer_cb
Fl_Text_Display, 1329
scroll_to
Fl_Scroll, 1133

scrollbar
Fl_Browser_, 387
scrollbar_align
Fl_Text_Display, 1329, 1330
scrollbar_left
Fl_Browser_, 384
scrollbar_right
Fl_Browser_, 384

scrollbar_size
Fl, 309
Fl_Browser_, 384, 385
Fl_Help_View, 706
Fl_Scroll, 1133
Fl_Table, 1253
Fl_Tree, 1435

scrollbar_width
Fl_Browser_, 385
Fl_Text_Display, 1330

scrollvalue
Fl_Slider, 1201

search_backward
Fl_Text_Buffer, 1293

search_forward
Fl_Text_Buffer, 1294

second
Fl_Clock_Output, 464

secondary_selection_text
Fl_Text_Buffer, 1294

select
Fl_Browser, 361
Fl_Browser_, 385
Fl_Tree, 1435, 1436
Fl_Tree_Item, 1459

select_all
Fl_Tree, 1436
Fl_Tree_Item, 1459

select_all_rows
Fl_Table_Row, 1268

select_only
Fl_Browser_, 386
Fl_Tree, 1437

select_row
Fl_Table_Row, 1268

select_toggle
Fl_Tree, 1437

selectbox
Fl_Tree, 1438

selected
Fl_Browser, 361
Fl_Text_Selection, 1361

selection
Fl_Browser_, 386
Selection & Clipboard functions, 214

Selection & Clipboard functions, 212
add_clipboard_notify, 212
clipboard_contains, 213
copy, 213
dnd, 213
paste, 213
selection, 214
selection_owner, 214

selection_color
Fl_Widget, 1534, 1535

selection_owner
Selection & Clipboard functions, 214

selection_text
selectmode
  Fl_Tree, 1438
  Fl_Tree_Prefs, 1469
set
  Fl_Button, 396
  Fl_FormsBitmap, 580
  Fl_FormsPixmap, 587
  Fl_Menu_Item, 879
  Fl_Preferences, 1031–1033
  Fl_Text_Selection, 1361
set_active
  Fl_Widget, 1535
set_atclose
  Windows handling functions, 198
set_box_color
  Fl, 309
set_changed
  Fl_Widget, 1535
set_checked
  Fl_Check_Browser, 430
set_color
  Color & Font functions, 226
set_current
  Fl_Copy_Surface, 479
  Fl_Image_Surface, 758
  Fl_Printer, 1041
  Fl_Surface_Device, 1214
set_draw_cb
  Fl_Cairo_Window, 409
set_font
  Color & Font functions, 227
set_fonts
  Color & Font functions, 227
set_idle
  Fl, 310
set_item_focus
  Fl_Tree, 1438
set_menu_window
  Fl_Widget, 1564
set_modal
  Fl_Widget, 1564
set_non_modal
  Fl_Widget, 1564
set_output
  Fl_Widget, 1535
set_overlay
  Fl_Menu_Window, 891
set_selection
  Fl_Table, 1253
set_tooltip_window
  Fl_Widget, 1564
set_visible
  Fl_Widget, 1535
set_visible_focus
  Fl_Widget, 1535
setonly
  Fl_Menu_Item, 879

shape
  Fl_Window, 1564
shortcut
  Fl_Button, 396
  Fl_Input_, 785
  Fl_Menu_Item, 879
  Fl_Text_Display, 1330
  Fl_Value_Input, 1489
SHORTCUT_LABEL
  Fl_Widget, 1515
show
  Fl_Browser, 362
  Fl_Double_Window, 513
  Fl_Gi_Window, 633
  Fl_Help_Dialog, 692
  Fl_Menu_Window, 891
  Fl_Native_File_Chooser, 928
  Fl_Overlay_Window, 957
  Fl_Single_Window, 1192
  Fl_Widget, 1536
  Fl_Window, 1565, 1566
show_cursor
  Fl_Text_Display, 1331
show_insert_position
  Fl_Text_Display, 1331
show_item
  Fl_Tree, 1438, 1439
  Fl_Tree_Item_bottom
  Fl_Tree, 1439
  Fl_Tree_Item_middle
  Fl_Tree, 1439
  Fl_Tree_Item_top
  Fl_Tree, 1439
show_self
  Fl_Tree, 1439
  Fl_Tree_Item, 1459
show_widgets
  Fl_Tree_Item, 1459
showcollapse
  Fl_Tree, 1440
  Fl_Tree_Prefs, 1469
showHiddenButton
  Fl_File_Chooser, 533
shown
  Fl_Window, 1566
showroot
  Fl_Tree, 1440
  Fl_Tree_Prefs, 1469
SIMPLE_CURSOR
  Fl_Text_Display, 1311
size
  Fl_Browser, 362
  Fl_Input_, 785
  Fl_Menu_, 847
  Fl_Menu_Item, 879
  Fl_Preferences, 1033
  Fl_Tooltip, 1393, 1394
  Fl_Widget, 1536
INDEX

undo
    FI_Input_. 788

Unicode and UTF-8 functions, 257
    ERRORS_TO_CP1252, 259
    ERRORS_TO_ISO8859_1, 259
    fl_access, 259
    fl_chmod, 259
    fl_fopen, 260
    fl_getcwd, 260
    fl.getenv, 260
    fl_make_path, 261
    fl_make_path_for_file, 261
    fl_mkdir, 261
    fl_nonspacing, 261
    fl_open, 261
    fl_rename, 262
    fl_rmdir, 262
    fl_stat, 262
    fl_system, 263
    fl_ucs_to_Utf16, 263
    fl_unlink, 263
    fl_utf8back, 264
    fl_utf8bytes, 264
    fl_utf8decode, 264
    fl_utf8encode, 264
    fl_utf8from_mb, 265
    fl_utf8fromma, 265
    fl_utf8fromwc, 265
    fl_utf8fromw, 266
    fl_utf8len, 266
    fl_utf8len1, 266
    fl_utf8locale, 266
    fl_utf8test, 266
    fl_utf8to_mb, 267
    fl_utf8toa, 267
    fl_utf8toUtf16, 267
    fl_utf8towc, 268
    fl_utf_strcasecmp, 268
    fl_utf_strncasecmp, 268
    fl_utf_tolower, 269
    fl_utf_toupper, 269
    fl_wcwidth, 269
    fl_wcwidth_, 270
    STRICT_RFC3629, 259

Unicode and UTF-8 Support, 111
unlock
    Multithreading support functions, 252

untranslate
    FI_Paged_Device, 972
    FI_PostScript_File_Device, 1001
    FI_Printer, 1042
    FI_System_Printer, 1232

up_down_position
    FI_Input_. 789

update
    FI_Text_Selection, 1361
update_child
    FI_Group, 689
next_window, 197
set_atclose, 198

word_end
 Fl_Input_, 790
 Fl_Text_Buffer, 1295
 Fl_Text_Display, 1333

word_start
 Fl_Input_, 790
 Fl_Text_Buffer, 1296
 Fl_Text_Display, 1334

wrap
 Fl_Input_, 791
 WRAP_AT_BOUNDS
 Fl_Text_Display, 1311
 WRAP_AT_COLUMN
 Fl_Text_Display, 1311
 WRAP_AT_PIXEL
 Fl_Text_Display, 1311

wrap_mode
 Fl_Text_Display, 1334
 WRAP_NONE
 Fl_Text_Display, 1311

wrap_uses_character
 Fl_Text_Display, 1334

wrap_width
 Fl_Tooltip, 1394

wrapped_column
 Fl_Text_Display, 1335

wrapped_line_counter
 Fl_Text_Display, 1335

wrapped_row
 Fl_Text_Display, 1336

x
 Fl_Widget, 1542
 x.H, 1830
 x_to_col
 Fl_Text_Display, 1336

class
 Fl_Widget, 1543

xposition
 Fl_Scroll, 1134

Ximint.h, 2237
Xlibint.h, 2237

xposition
 Fl_Scroll, 1134

Xutf8.h, 1879

xy_to_position
 Fl_Text_Display, 1337

xy_to_rowcol
 Fl_Text_Display, 1337

xyline
 Fl_Graphics_Driver, 664
 Fl_PostScript_Graphics_Driver, 1015

y
 Fl_Widget, 1543

yposition
 Fl_Scroll, 1134

yxline
 Fl_Graphics_Driver, 664, 665