## Contents

1. **FLTK Programming Manual** ........................................... 1

2. **Preface** ......................................................................... 3
   2.1 Organization .................................................................. 3
   2.2 Conventions .................................................................. 4
   2.3 Abbreviations ................................................................. 4
   2.4 Copyrights and Trademarks .............................................. 4

3. **Introduction to FLTK** .................................................... 5
   3.1 History of FLTK ............................................................. 5
   3.2 Features ....................................................................... 6
   3.3 Licensing ...................................................................... 6
   3.4 What Does "FLTK" Mean? ................................................ 7
   3.5 Building and Installing FLTK Under UNIX and Apple macOS ......................................................... 7
   3.6 Building FLTK Under Microsoft Windows ......................... 9
      3.6.1 GNU toolsets (Cygwin or MinGW) hosted on Windows ......................................................... 9
      3.6.2 Using the Visual C++ DLL Library ................................................. 10
   3.7 Internet Resources .......................................................... 10
   3.8 Reporting Bugs ............................................................... 10
## 7 Designing a Simple Text Editor

### 7.1 Determining the Goals of the Text Editor

### 7.2 Designing the Main Window

### 7.3 Variables

### 7.4 Menubars and Menus

### 7.5 Editing the Text

### 7.6 The Replace Dialog

### 7.7 Callbacks

#### 7.7.1 changed_cb()

#### 7.7.2 copy_cb()

#### 7.7.3 cut_cb()

#### 7.7.4 delete_cb()

#### 7.7.5 find_cb()

#### 7.7.6 find2_cb()

#### 7.7.7 new_cb()

#### 7.7.8 open_cb()

#### 7.7.9 paste_cb()

#### 7.7.10 quit_cb()

#### 7.7.11 replace_cb()

#### 7.7.12 replace2_cb()

#### 7.7.13 replall_cb()

#### 7.7.14 replcan_cb()

#### 7.7.15 save_cb()

#### 7.7.16 saveas_cb()

### 7.8 Other Functions

#### 7.8.1 check_save()

#### 7.8.2 load_file()

#### 7.8.3 save_file()

#### 7.8.4 set_title()

### 7.9 The main() Function

### 7.10 Compiling the Editor

### 7.11 The Final Product

### 7.12 Advanced Features

#### 7.12.1 Syntax Highlighting
8 Drawing Things in FLTK

8.1 When Can You Draw Things in FLTK? ............................................ 49

8.1.1 What Drawing Unit do FLTK drawing functions use? ......................... 50

8.2 Drawing Functions ................................................................. 51

8.2.1 Boxes ................................................................. 51

8.2.2 Clipping ................................................................. 52

8.3 Colors ............................................................. 53

8.3.1 Line Dashes and Thickness .................................................. 55

8.3.2 Drawing Fast Shapes ....................................................... 56

8.3.3 Drawing Complex Shapes ................................................... 58

8.3.4 Drawing Text ............................................................ 61

8.3.5 Fonts ................................................................. 63

8.3.6 Character Encoding .......................................................... 63

8.3.7 Drawing Overlays .......................................................... 64

8.4 Drawing Images ................................................................. 64

8.4.1 Direct Image Drawing ....................................................... 64

8.4.2 Direct Image Reading .......................................................... 66

8.4.3 Image Classes ............................................................ 66

8.4.4 Offscreen Drawing .......................................................... 68

9 Handling Events ................................................................. 69

9.1 The FLTK Event Model .......................................................... 69

9.2 Mouse Events ................................................................. 69

9.2.1 FL_PUSH ............................................................. 69

9.2.2 FL_DRAG ............................................................. 69

9.2.3 FL_RELEASE .......................................................... 70

9.2.4 FL_MOVE ............................................................. 70

9.2.5 FL_MOUSEWHEEL ...................................................... 70

9.3 Focus Events ................................................................. 70

9.3.1 FL_ENTER ............................................................. 70

9.3.2 FL_LEAVE ............................................................. 70
## 9.3.3 FL_FOCUS

### 9.3.4 FL_UNFOCUS

## 9.4 Keyboard Events

### 9.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP

### 9.4.2 FL_SHORTCUT

## 9.5 Widget Events

### 9.5.1 FL_DEACTIVATE

### 9.5.2 FL_ACTIVATE

### 9.5.3 FL_HIDE

### 9.5.4 FL_SHOW

## 9.6 Clipboard Events

### 9.6.1 FL_PASTE

### 9.6.2 FL_SELECTIONCLEAR

## 9.7 Drag and Drop Events

### 9.7.1 FL_DND_ENTER

### 9.7.2 FL_DND_DRAG

### 9.7.3 FL_DND_LEAVE

### 9.7.4 FL_DND_RELEASE

## 9.8 Other events

### 9.8.1 FL_SCREEN_CONFIGURATION_CHANGED

### 9.8.2 FL_FULLSCREEN

## 9.9 Fl::event_*() methods

## 9.10 Event Propagation

## 9.11 FLTK Compose-Character Sequences
## 14 Unicode and UTF-8 Support

- 14.1 About Unicode, ISO 10646 and UTF-8 ........................................ 131
- 14.2 Unicode in FLTK ................................................................. 133
- 14.3 Illegal Unicode and UTF-8 Sequences ..................................... 134
- 14.4 FLTK Unicode and UTF-8 Functions ...................................... 134
- 14.5 FLTK Unicode Versions of System Calls ................................. 137

## 15 FLTK Enumerations

- 15.1 Version Numbers ............................................................... 139
- 15.2 Events .............................................................................. 140
- 15.3 Callback "When" Conditions ............................................... 141
- 15.4 Fl::event_button() Values ................................................ 141
- 15.5 Fl::event_key() Values ...................................................... 141
- 15.6 Fl::event_state() Values .................................................. 142
- 15.7 Alignment Values ............................................................. 143
- 15.8 Fonts .............................................................................. 143
- 15.9 Colors ............................................................................. 144
  - 15.9.1 Color Constants .......................................................... 144
- 15.10 Cursors .......................................................................... 145
- 15.11 FD "When" Conditions ..................................................... 146
- 15.12 Damage Masks ............................................................... 146

## 16 GLUT Compatibility

- 16.1 Using the GLUT Compatibility Header File ............................... 147
- 16.2 Known Problems ............................................................... 147
- 16.3 Mixing GLUT and FLTK Code ........................................... 148
- 16.4 class Fl_Glut_Window .......................................................... 149
  - 16.4.1 Class Hierarchy ............................................................. 149
  - 16.4.2 Include Files ............................................................... 149
  - 16.4.3 Description .................................................................. 149
  - 16.4.4 Members .................................................................... 149
  - 16.4.5 Methods ..................................................................... 150
### 17 Forms Compatibility

17.1 Importing Forms Layout Files ........................................ 151
17.2 Using the Compatibility Header File .............................. 151
17.3 Problems You Will Encounter ......................................... 152
17.4 Additional Notes ......................................................... 153

### 18 Operating System Issues

18.1 Accessing the OS Interfaces .......................................... 157
18.2 The UNIX (X11) Interface ............................................. 157
   18.2.1 Handling Other X Events ..................................... 158
   18.2.2 Drawing using Xlib ............................................ 159
   18.2.3 Changing the Display, Screen, or X Visual ................. 160
   18.2.4 Using a Subclass of Fl_Window for Special X Stuff .... 161
   18.2.5 Setting the Icon of a Window ................................ 162
   18.2.6 X Resources ..................................................... 163
   18.2.7 Display Scaling Factor ........................................ 164
18.3 The Windows Interface ............................................... 164
   18.3.1 Using filenames with non-ASCII characters ................ 164
   18.3.2 Responding to WM_QUIT ....................................... 164
   18.3.3 Handling Other Windows API Messages ...................... 164
   18.3.4 Drawing Things Using the Windows GDI ........................ 165
   18.3.5 Display Scaling Factor ...................................... 165
   18.3.6 Setting the Icon of a Window ................................ 165
   18.3.7 How to Not Get a MSDOS Console Window ................... 166
   18.3.8 Known Windows Bugs and Problems .......................... 166
18.4 The Apple OS X Interface ............................................ 166
   18.4.1 Setting the icon of an application .......................... 168
   18.4.2 Drawing Things Using Quartz ................................ 168
   18.4.3 Internationalization ......................................... 169
   18.4.4 OpenGL and ‘retina’ displays ................................ 169
   18.4.5 Fl_Double_Window .............................................. 170
   18.4.6 Mac File System Specifics .................................... 170
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Migrating Code from FLTK 1.3 to 1.4</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>19.1 Migrating from FLTK 1.0 or 1.1 to 1.4</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>19.2 Minor Changes in Header Files</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>19.3 Fl_Preferences</td>
<td>171</td>
</tr>
<tr>
<td>20</td>
<td>Developer Information</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>20.1 Non-ASCII Characters</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>20.2 Document Structure</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>20.3 Creating Links</td>
<td>176</td>
</tr>
<tr>
<td></td>
<td>20.4 Paragraph Layout</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td>20.5 Navigation Elements</td>
<td>178</td>
</tr>
<tr>
<td>21</td>
<td>Software License</td>
<td>179</td>
</tr>
<tr>
<td>22</td>
<td>Example Source Code</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>22.1 Example Applications: Overview</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>22.1.1 adjuster</td>
<td>187</td>
</tr>
<tr>
<td></td>
<td>22.1.2 animated</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.3 arc</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.4 ask</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.5 bitmap</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.6 blocks</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.7 boxtype</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.8 browser</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.9 button</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>22.1.10 buttons</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.11 cairo_test</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.12 checkers</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.13 clock</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.14 colbrowser</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.15 color_chooser</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>22.1.16 cube</td>
<td>189</td>
</tr>
</tbody>
</table>
22.1.17 CubeView .................................................. 189
22.1.18 cursor ..................................................... 190
22.1.19 curve ...................................................... 190
22.1.20 demo ....................................................... 190
22.1.21 device ...................................................... 190
22.1.22 doublebuffer ............................................. 190
22.1.23 editor ....................................................... 190
22.1.24 fast_slow .................................................. 190
22.1.25 file_chooser .............................................. 191
22.1.26 fonts ...................................................... 191
22.1.27 forms ...................................................... 191
22.1.28 fractals ................................................... 191
22.1.29 fullscreen ............................................... 191
22.1.30 gl_overlay ............................................... 191
22.1.31 glpuzzle .................................................. 191
22.1.32 hello ...................................................... 191
22.1.33 help_dialog .............................................. 192
22.1.34 icon ....................................................... 192
22.1.35 iconize .................................................... 192
22.1.36 image ...................................................... 192
22.1.37 inactive ................................................... 192
22.1.38 input ...................................................... 192
22.1.39 input_choice ............................................. 192
22.1.40 keyboard .................................................. 193
22.1.41 label ....................................................... 193
22.1.42 line_style ................................................ 193
22.1.43 list_visuals .............................................. 193
22.1.44 mandelbrot .............................................. 193
22.1.45 menubar .................................................. 193
22.1.46 message ................................................... 193
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1.47</td>
<td>minimum</td>
<td>193</td>
</tr>
<tr>
<td>22.1.48</td>
<td>native-filechooser</td>
<td>194</td>
</tr>
<tr>
<td>22.1.49</td>
<td>navigation</td>
<td>194</td>
</tr>
<tr>
<td>22.1.50</td>
<td>offscreen</td>
<td>194</td>
</tr>
<tr>
<td>22.1.51</td>
<td>output</td>
<td>194</td>
</tr>
<tr>
<td>22.1.52</td>
<td>overlay</td>
<td>194</td>
</tr>
<tr>
<td>22.1.53</td>
<td>pack</td>
<td>194</td>
</tr>
<tr>
<td>22.1.54</td>
<td>pixmap</td>
<td>194</td>
</tr>
<tr>
<td>22.1.55</td>
<td>pixmap_browser</td>
<td>194</td>
</tr>
<tr>
<td>22.1.56</td>
<td>preferences</td>
<td>195</td>
</tr>
<tr>
<td>22.1.57</td>
<td>radio</td>
<td>195</td>
</tr>
<tr>
<td>22.1.58</td>
<td>resizebox</td>
<td>195</td>
</tr>
<tr>
<td>22.1.59</td>
<td>rotated_text</td>
<td>195</td>
</tr>
<tr>
<td>22.1.60</td>
<td>resize</td>
<td>195</td>
</tr>
<tr>
<td>22.1.61</td>
<td>scroll</td>
<td>195</td>
</tr>
<tr>
<td>22.1.62</td>
<td>shape</td>
<td>195</td>
</tr>
<tr>
<td>22.1.63</td>
<td>subwindow</td>
<td>195</td>
</tr>
<tr>
<td>22.1.64</td>
<td>sudoku</td>
<td>196</td>
</tr>
<tr>
<td>22.1.65</td>
<td>symbols</td>
<td>196</td>
</tr>
<tr>
<td>22.1.66</td>
<td>table</td>
<td>196</td>
</tr>
<tr>
<td>22.1.67</td>
<td>tabs</td>
<td>196</td>
</tr>
<tr>
<td>22.1.68</td>
<td>threads</td>
<td>196</td>
</tr>
<tr>
<td>22.1.69</td>
<td>tile</td>
<td>196</td>
</tr>
<tr>
<td>22.1.70</td>
<td>tiled_image</td>
<td>196</td>
</tr>
<tr>
<td>22.1.71</td>
<td>tree</td>
<td>197</td>
</tr>
<tr>
<td>22.1.72</td>
<td>twowin</td>
<td>197</td>
</tr>
<tr>
<td>22.1.73</td>
<td>unittests</td>
<td>197</td>
</tr>
<tr>
<td>22.1.74</td>
<td>utf8</td>
<td>197</td>
</tr>
<tr>
<td>22.1.75</td>
<td>valuators</td>
<td>197</td>
</tr>
<tr>
<td>22.1.76</td>
<td>windowfocus</td>
<td>197</td>
</tr>
<tr>
<td>22.1.77</td>
<td>fluid</td>
<td>197</td>
</tr>
<tr>
<td>22.2</td>
<td>Example Applications: Images</td>
<td>197</td>
</tr>
<tr>
<td>22.2.1</td>
<td>cairo_test</td>
<td>198</td>
</tr>
<tr>
<td>22.2.2</td>
<td>icon</td>
<td>198</td>
</tr>
<tr>
<td>22.2.3</td>
<td>unittests</td>
<td>198</td>
</tr>
</tbody>
</table>
30.7.2.14 event_button3() ...................................................... 244
30.7.2.15 event_buttons() ....................................................... 244
30.7.2.16 event_clicks() [1/2] ................................................... 244
30.7.2.17 event_clicks() [2/2] ................................................... 245
30.7.2.18 event_clipboard() ..................................................... 245
30.7.2.19 event_clipboard_type() .............................................. 245
30.7.2.20 event_command() ...................................................... 245
30.7.2.21 event_ctrl() ........................................................... 246
30.7.2.22 event_dispatch() ...................................................... 246
30.7.2.23 event_dx() ............................................................. 246
30.7.2.24 event_dy() ............................................................. 247
30.7.2.25 event_inside() [1/2] .................................................... 247
30.7.2.26 event_inside() [2/2] .................................................... 247
30.7.2.27 event_is_click() [1/2] ............................................... 248
30.7.2.28 event_is_click() [2/2] ............................................... 248
30.7.2.29 event_key() [1/2] ....................................................... 248
30.7.2.30 event_key() [2/2] ....................................................... 249
30.7.2.31 event_length() ........................................................ 249
30.7.2.32 event_original_key() ............................................... 249
30.7.2.33 event_shift() .......................................................... 250
30.7.2.34 event_state() [1/2] .................................................... 250
30.7.2.35 event_state() [2/2] .................................................... 250
30.7.2.36 event_text() .......................................................... 251
30.7.2.37 event_x_root() ......................................................... 251
30.7.2.38 event_y_root() ......................................................... 251
30.7.2.39 focus() [1/2] ........................................................... 251
30.7.2.40 focus() [2/2] ........................................................... 252
30.7.2.41 get_key() ............................................................... 252
30.7.2.42 get_mouse() ........................................................... 252
30.7.2.43 handle() ............................................................... 252
| 30.11.4.9 | fl_circle | 287 |
| 30.11.4.10 | fl_clip_box | 287 |
| 30.11.4.1 | fl_clip_region [1/2] | 288 |
| 30.11.4.2 | fl_clip_region [2/2] | 289 |
| 30.11.4.13 | fl_copy_offscreen | 289 |
| 30.11.4.14 | fl_create_offscreen | 289 |
| 30.11.4.15 | fl_cursor | 290 |
| 30.11.4.16 | fl_curve | 290 |
| 30.11.4.17 | fl_delete_offscreen | 291 |
| 30.11.4.18 | fl_draw [1/4] | 291 |
| 30.11.4.20 | fl_draw [3/4] | 292 |
| 30.11.4.21 | fl_draw [4/4] | 292 |
| 30.11.4.22 | fl_draw_box | 292 |
| 30.11.4.23 | fl_draw_image [1/2] | 293 |
| 30.11.4.24 | fl_draw_image [2/2] | 294 |
| 30.11.4.25 | fl_draw_image_mono [1/2] | 294 |
| 30.11.4.26 | fl_draw_image_mono [2/2] | 295 |
| 30.11.4.27 | fl_draw_pixmap [1/2] | 295 |
| 30.11.4.28 | fl_draw_pixmap [2/2] | 295 |
| 30.11.4.29 | fl_draw_symbol | 296 |
| 30.11.4.30 | fl_expand_text | 296 |
| 30.11.4.31 | fl_frame | 297 |
| 30.11.4.32 | fl_frame2 | 297 |
| 30.11.4.33 | fl_gap | 297 |
| 30.11.4.34 | fl_line_style | 298 |
| 30.11.4.35 | fl_measure | 298 |
| 30.11.4.36 | fl_measure_pixmap [1/2] | 299 |
| 30.11.4.37 | fl_measure_pixmap [2/2] | 299 |
| 30.11.4.38 | fl_mult_matrix | 300 |
30.12.2 Function Documentation ............................................. 313
  30.12.2.1 awake() [1/2] ................................................... 313
  30.12.2.2 awake() [2/2] ................................................... 314
  30.12.2.3 lock() .......................................................... 314
  30.12.2.4 thread_message() ............................................. 314
  30.12.2.5 unlock() ........................................................ 314

30.13 Safe widget deletion support functions ................................. 315
  30.13.1 Detailed Description ............................................... 315
  30.13.2 Function Documentation ........................................... 316
    30.13.2.1 clear_widget_pointer() .................................... 316
    30.13.2.2 delete_widget() ............................................. 316
    30.13.2.3 do_widget_deletion() ....................................... 317
    30.13.2.4 release_widget_pointer() ................................... 317
    30.13.2.5 watch_widget_pointer() ..................................... 317

30.14 Cairo Support Functions and Classes .................................. 319
  30.14.1 Detailed Description ............................................... 319
  30.14.2 Function Documentation ........................................... 319
    30.14.2.1 cairo_autolink_context() [1/2] ............................. 319
    30.14.2.2 cairo_autolink_context() [2/2] ............................. 319
    30.14.2.3 cairo_cc() [1/2] ........................................... 320
    30.14.2.4 cairo_cc() [2/2] ........................................... 320
    30.14.2.5 cairo_make_current() ...................................... 320

30.15 Unicode and UTF-8 functions ......................................... 321
  30.15.1 Detailed Description ............................................... 323
  30.15.2 Macro Definition Documentation .................................. 323
    30.15.2.1 ERRORS_TO_CP1252 ......................................... 323
    30.15.2.2 ERRORS_TO_ISO8859_1 ..................................... 323
    30.15.2.3 STRICT_RFC3629 ............................................ 323
  30.15.3 Function Documentation ........................................... 324
    30.15.3.1 fl_access() ................................................ 324
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_chdir()</td>
<td>324</td>
</tr>
<tr>
<td>fl_chmod()</td>
<td>325</td>
</tr>
<tr>
<td>fl_fopen()</td>
<td>325</td>
</tr>
<tr>
<td>fl_getcwd()</td>
<td>326</td>
</tr>
<tr>
<td>fl_getenv()</td>
<td>326</td>
</tr>
<tr>
<td>fl_make_path()</td>
<td>327</td>
</tr>
<tr>
<td>fl_make_path_for_file()</td>
<td>327</td>
</tr>
<tr>
<td>fl_mkdir()</td>
<td>328</td>
</tr>
<tr>
<td>fl_nonspacing()</td>
<td>328</td>
</tr>
<tr>
<td>fl_open()</td>
<td>328</td>
</tr>
<tr>
<td>fl_open_ext()</td>
<td>329</td>
</tr>
<tr>
<td>fl_putenv()</td>
<td>329</td>
</tr>
<tr>
<td>fl_rename()</td>
<td>330</td>
</tr>
<tr>
<td>fl_rmdir()</td>
<td>331</td>
</tr>
<tr>
<td>fl_stat()</td>
<td>331</td>
</tr>
<tr>
<td>fl_system()</td>
<td>331</td>
</tr>
<tr>
<td>fl_ucs_to_Utf16()</td>
<td>332</td>
</tr>
<tr>
<td>fl_unlink()</td>
<td>332</td>
</tr>
<tr>
<td>fl_utf8back()</td>
<td>333</td>
</tr>
<tr>
<td>fl_utf8bytes()</td>
<td>333</td>
</tr>
<tr>
<td>fl_utf8decode()</td>
<td>333</td>
</tr>
<tr>
<td>fl_utf8encode()</td>
<td>334</td>
</tr>
<tr>
<td>fl_utf8from_mb()</td>
<td>334</td>
</tr>
<tr>
<td>fl_utf8froma()</td>
<td>335</td>
</tr>
<tr>
<td>fl_utf8fromwc()</td>
<td>335</td>
</tr>
<tr>
<td>fl_utf8fwd()</td>
<td>336</td>
</tr>
<tr>
<td>fl_utf8len()</td>
<td>336</td>
</tr>
<tr>
<td>fl_utf8len1()</td>
<td>336</td>
</tr>
<tr>
<td>fl_utf8locale()</td>
<td>337</td>
</tr>
<tr>
<td>fl_utf8test()</td>
<td>337</td>
</tr>
<tr>
<td>30.17.2.1</td>
<td>fl_input()</td>
</tr>
<tr>
<td>30.17.2.2</td>
<td>fl_message()</td>
</tr>
<tr>
<td>30.17.2.3</td>
<td>fl_message_hotspot() [1/2]</td>
</tr>
<tr>
<td>30.17.2.4</td>
<td>fl_message_hotspot() [2/2]</td>
</tr>
<tr>
<td>30.17.2.5</td>
<td>fl_message_icon()</td>
</tr>
<tr>
<td>30.17.2.6</td>
<td>fl_message_position() [1/3]</td>
</tr>
<tr>
<td>30.17.2.7</td>
<td>fl_message_position() [2/3]</td>
</tr>
<tr>
<td>30.17.2.8</td>
<td>fl_message_position() [3/3]</td>
</tr>
<tr>
<td>30.17.2.9</td>
<td>fl_message_title()</td>
</tr>
<tr>
<td>30.17.2.10</td>
<td>fl_message_title_default()</td>
</tr>
<tr>
<td>30.17.2.11</td>
<td>fl_password()</td>
</tr>
</tbody>
</table>

30.17.3 Variable Documentation

| 30.17.3.1 | error | 358 |
| 30.17.3.2 | fatal | 358 |
| 30.17.3.3 | warning | 358 |

30.18 File names and URI utility functions

| 30.18.1 | Detailed Description | 359 |
| 30.18.2 | Typedef Documentation | 360 |
| 30.18.2.1 | Fl_File_Sort_F | 360 |

30.18.3 Function Documentation

| 30.18.3.1 | fl_decode_uri() | 360 |
| 30.18.3.2 | fl_filename_absolute() | 360 |
| 30.18.3.3 | fl_filename_expand() | 361 |
| 30.18.3.4 | fl_filename_ext() | 361 |
| 30.18.3.5 | fl_filename_free_list() | 362 |
| 30.18.3.6 | fl_filename_isdir() | 362 |
| 30.18.3.7 | fl_filename_list() | 363 |
| 30.18.3.8 | fl_filename_match() | 363 |
| 30.18.3.9 | fl_filename_name() | 364 |
| 30.18.3.10 | fl_filename_relative() | 365 |
| 30.18.3.11 | fl_filename_setext() | 365 |
| 30.18.3.12 | fl_open_uri() | 366 |
31 Class Documentation 367

31.1 Fl_Preferences::Entry Struct Reference 367

31.2 Fl Class Reference 367

31.2.1 Detailed Description 376

31.2.2 Member Enumeration Documentation 377

31.2.2.1 Fl_Option 377

31.2.3 Member Function Documentation 377

31.2.3.1 abi_check() 378

31.2.3.2 abi_version() 378

31.2.3.3 add_awake_handler_() 378

31.2.3.4 add_check() 379

31.2.3.5 add_fd() [1/2] 379

31.2.3.6 add_fd() [2/2] 380

31.2.3.7 add_idle() 380

31.2.3.8 add_timeout() 380

31.2.3.9 api_version() 381

31.2.3.10 arg() 381

31.2.3.11 args() [1/2] 382

31.2.3.12 args() [2/2] 382

31.2.3.13 background() 383

31.2.3.14 background2() 383

31.2.3.15 box_color() 383

31.2.3.16 box_dh() 383

31.2.3.17 box_dw() 384

31.2.3.18 box_dx() 384

31.2.3.19 box_dy() 384

31.2.3.20 check() 385

31.2.3.21 damage() 385

31.2.3.22 display() 385

31.2.3.23 dnd_text_ops() [1/2] 385
31.2.3.24 dnd_text_ops() [2/2] ................................. 385
31.2.3.25 draw_box_active() ................................. 386
31.2.3.26 draw_GL_text_with_textures() [1/2] .................. 386
31.2.3.27 draw_GL_text_with_textures() [2/2] .................. 386
31.2.3.28 flush() ............................................. 387
31.2.3.29 foreground() ....................................... 387
31.2.3.30 get_awake_handler() ............................... 387
31.2.3.31 get_boxtype() ..................................... 387
31.2.3.32 get_system_colors() ............................... 387
31.2.3.33 gl_visual() ......................................... 388
31.2.3.34 insertion_point_location() ......................... 388
31.2.3.35 is_scheme() ........................................ 388
31.2.3.36 menu_linespacing() [1/2] ............................ 389
31.2.3.37 menu_linespacing() [2/2] ............................ 389
31.2.3.38 option() [1/2] ...................................... 389
31.2.3.39 option() [2/2] ...................................... 390
31.2.3.40 own_colormap() ................................... 391
31.2.3.41 program_should_quit() [1/2] ......................... 391
31.2.3.42 program_should_quit() [2/2] ......................... 391
31.2.3.43 readqueue() ...................................... 392
31.2.3.44 ready() ............................................. 392
31.2.3.45 release() .......................................... 392
31.2.3.46 reload_scheme() ................................... 393
31.2.3.47 remove_check() .................................... 393
31.2.3.48 remove_fd() [1/2] .................................. 393
31.2.3.49 remove_fd() [2/2] .................................. 393
31.2.3.50 remove_timeout() .................................. 393
31.2.3.51 repeat_timeout() .................................. 394
31.2.3.52 reset_marked_text() ............................... 394
31.2.3.53 run() ................................................. 394
31.4 Detailed Description ................................................. 404
31.4.2 Constructor & Destructor Documentation ....................... 404
  31.4.2.1 Fl_Bitmap() ............................................. 405
  31.4.2.2 Fl_Bitmap() ............................................. 405
31.4.3 Member Function Documentation .................................. 405
  31.4.3.1 copy() ..................................................................... 405
  31.4.3.2 draw() .................................................................. 405
  31.4.3.3 label() .............................................................. 406
  31.4.3.4 label() .............................................................. 406
  31.4.3.5 uncache() ........................................................... 406
31.5 Fl_BMP_Image Class Reference ........................................ 407
  31.5.1 Detailed Description ................................................. 407
  31.5.2 Constructor & Destructor Documentation ....................... 407
    31.5.2.1 Fl_BMP_Image() .................................................. 407
    31.5.2.2 Fl_BMP_Image() .................................................. 408
31.6 Fl_Box Class Reference .................................................. 408
  31.6.1 Detailed Description ................................................. 409
  31.6.2 Constructor & Destructor Documentation ....................... 409
    31.6.2.1 Fl_Box() ......................................................... 409
  31.6.3 Member Function Documentation .................................. 410
    31.6.3.1 draw() .................................................................. 410
    31.6.3.2 handle() ........................................................... 410
31.7 Fl_Browser Class Reference ............................................. 411
  31.7.1 Detailed Description ................................................. 414
  31.7.2 Constructor & Destructor Documentation ....................... 415
    31.7.2.1 Fl_Browser() ...................................................... 415
  31.7.3 Member Function Documentation .................................. 415
    31.7.3.1 _remove() ......................................................... 415
    31.7.3.2 add() .................................................................. 416
    31.7.3.3 bottomline() ....................................................... 416
31.7.3.34 item_text() ............................................. 429
31.7.3.35 item_width() ......................................... 429
31.7.3.36 lineno() ............................................. 430
31.7.3.37 lineposition() ....................................... 430
31.7.3.38 load() ................................................ 431
31.7.3.39 make_visible() ...................................... 431
31.7.3.40 middleline() ......................................... 432
31.7.3.41 move() ................................................ 432
31.7.3.42 remove() ............................................. 432
31.7.3.43 remove_icon() ....................................... 433
31.7.3.44 replace() ............................................. 433
31.7.3.45 select() ............................................. 433
31.7.3.46 selected() ........................................... 434
31.7.3.47 show() [1/2] .......................................... 434
31.7.3.48 show() [2/2] .......................................... 435
31.7.3.49 size() ................................................ 435
31.7.3.50 swap() [1/2] .......................................... 435
31.7.3.51 swap() [2/2] .......................................... 435
31.7.3.52 text() [1/2] .......................................... 436
31.7.3.53 text() [2/2] .......................................... 436
31.7.3.54 textsize() ........................................... 437
31.7.3.55 topline() [1/2] ....................................... 437
31.7.3.56 topline() [2/2] ....................................... 437
31.7.3.57 value() [1/2] .......................................... 438
31.7.3.58 value() [2/2] .......................................... 438
31.7.3.59 visible() ............................................. 438

31.8 Fl_Browser_ class Reference .................................. 439
31.8.1 Detailed Description ...................................... 442
31.8.2 Member Enumeration Documentation ...................... 442
31.8.2.1 anonymous enum ...................................... 443
## 31.8.3 Constructor & Destructor Documentation

31.8.3.1 `Fl_Browser()` 

## 31.8.4 Member Function Documentation

31.8.4.1 `bbox()` 
31.8.4.2 `deleting()` 
31.8.4.3 `deselect()` 
31.8.4.4 `display()` 
31.8.4.5 `displayed()` 
31.8.4.6 `find_item()` 
31.8.4.7 `full_height()` 
31.8.4.8 `full_width()` 
31.8.4.9 `handle()` 
31.8.4.10 `has_scrollbar()` 
31.8.4.11 `hposition()` [1/2] 
31.8.4.12 `hposition()` [2/2] 
31.8.4.13 `incr_height()` 
31.8.4.14 `inserting()` 
31.8.4.15 `item_at()` 
31.8.4.16 `item_first()` 
31.8.4.17 `item_height()` 
31.8.4.18 `item_last()` 
31.8.4.19 `item_next()` 
31.8.4.20 `item_prev()` 
31.8.4.21 `item_quick_height()` 
31.8.4.22 `item_select()` 
31.8.4.23 `item_selected()` 
31.8.4.24 `item_swap()` 
31.8.4.25 `item_text()` 
31.8.4.26 `item_width()` 
31.8.4.27 `leftedge()`
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.9.3.4</td>
<td><code>draw()</code></td>
</tr>
<tr>
<td>31.9.3.5</td>
<td><code>handle()</code></td>
</tr>
<tr>
<td>31.9.3.6</td>
<td><code>set()</code></td>
</tr>
<tr>
<td>31.9.3.7</td>
<td><code>shortcut()</code> [1/2]</td>
</tr>
<tr>
<td>31.9.3.8</td>
<td><code>shortcut()</code> [2/2]</td>
</tr>
<tr>
<td>31.9.3.9</td>
<td><code>value()</code></td>
</tr>
</tbody>
</table>

31.10 Fl_Cairo_State Class Reference

31.10.1 Detailed Description

31.10.2 Member Function Documentation

31.10.2.1 cc()

31.11 Fl_Cairo_Window Class Reference

31.11.1 Detailed Description

31.11.2 Member Function Documentation

31.11.2.1 set_draw_cb()

31.12 Fl_Chart Class Reference

31.12.1 Detailed Description

31.12.2 Constructor & Destructor Documentation

31.12.2.1 Fl_Chart()

31.12.3 Member Function Documentation

31.12.3.1 add()

31.12.3.2 autosize() [1/2]

31.12.3.3 autosize() [2/2]

31.12.3.4 bounds() [1/2]

31.12.3.5 bounds() [2/2]

31.12.3.6 draw()

31.12.3.7 insert()

31.12.3.8 maxsize()

31.12.3.9 replace()

31.12.3.10 textcolor()

31.12.3.11 textfont()
31.18.2 Constructor & Destructor Documentation ........................................ 495
  31.18.2.1 Fl_Clock_Output() ..................................................... 495

31.18.3 Member Function Documentation ............................................... 496
  31.18.3.1 draw() ...................................................................... 496
  31.18.3.2 hour() ..................................................................... 496
  31.18.3.3 minute() ................................................................. 496
  31.18.3.4 second() ................................................................. 497
  31.18.3.5 shadow() [1/2] ......................................................... 497
  31.18.3.6 shadow() [2/2] ......................................................... 497
  31.18.3.7 value() [1/3] ........................................................... 498
  31.18.3.8 value() [2/3] ........................................................... 498
  31.18.3.9 value() [3/3] ........................................................... 498

31.19 Fl_Color_Chooser Class Reference ............................................... 499
  31.19.1 Detailed Description .......................................................... 500
  31.19.2 Constructor & Destructor Documentation .................................... 501
    31.19.2.1 Fl_Color_Chooser() .................................................. 501
  31.19.3 Member Function Documentation ............................................ 501
    31.19.3.1 b() ...................................................................... 501
    31.19.3.2 g() ...................................................................... 501
    31.19.3.3 handle() ................................................................. 502
    31.19.3.4 hsv() .................................................................... 502
    31.19.3.5 hsv2rgb() ............................................................... 503
    31.19.3.6 hue() .................................................................... 503
    31.19.3.7 mode() [1/2] ........................................................... 503
    31.19.3.8 mode() [2/2] ........................................................... 504
    31.19.3.9 r() ...................................................................... 504
    31.19.3.10rgb() .................................................................... 504
    31.19.3.11 rgb2hsv() ................................................................ 504
    31.19.3.12 saturation() ......................................................... 505
    31.19.3.13 value() .................................................................. 505
31.50.1 Detailed Description .................................................................................. 595
31.50.2 Constructor & Destructor Documentation ............................................. 596
  31.50.2.1 Fl_Help_Dialog() ........................................................................... 596
31.50.3 Member Function Documentation ......................................................... 596
  31.50.3.1 h() ............................................................................................. 596
  31.50.3.2 hide() ......................................................................................... 596
  31.50.3.3 load() ........................................................................................ 596
  31.50.3.4 position() ................................................................................... 597
  31.50.3.5 resize() ....................................................................................... 597
  31.50.3.6 show() ......................................................................................... 597
  31.50.3.7 textsize() [1/2] ............................................................................. 597
  31.50.3.8 textsize() [2/2] ............................................................................. 598
  31.50.3.9 value() [1/2] ............................................................................... 598
  31.50.3.10 value() [2/2] .............................................................................. 598
  31.50.3.11 visible() ..................................................................................... 598
  31.50.3.12 w() .......................................................................................... 598
  31.50.3.13 x() .......................................................................................... 598
  31.50.3.14 y() .......................................................................................... 599
31.51 Fl_Help_Font_Stack Struct Reference ...................................................... 599
  31.51.1 Constructor & Destructor Documentation ........................................... 599
    31.51.1.1 Fl_Help_Font_Stack() ................................................................. 599
31.51.2 Member Function Documentation ......................................................... 600
  31.51.2.1 count() ....................................................................................... 600
  31.51.2.2 top() .......................................................................................... 600
31.52 Fl_Help_Font_Style Struct Reference ...................................................... 600
  31.52.1 Detailed Description ............................................................................ 600
31.53 Fl_Help_Link Struct Reference ................................................................. 600
  31.53.1 Detailed Description ............................................................................ 600
31.54 Fl_Help_Target Struct Reference ............................................................. 600
  31.54.1 Detailed Description ............................................................................ 600
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.55</td>
<td>Fl_Help_View Class Reference</td>
</tr>
<tr>
<td>31.55.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.55.2</td>
<td>Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>31.55.2.1</td>
<td>~Fl_Help_View()</td>
</tr>
<tr>
<td>31.55.3</td>
<td>Member Function Documentation</td>
</tr>
<tr>
<td>31.55.3.1</td>
<td>clear_selection()</td>
</tr>
<tr>
<td>31.55.3.2</td>
<td>directory()</td>
</tr>
<tr>
<td>31.55.3.3</td>
<td>draw()</td>
</tr>
<tr>
<td>31.55.3.4</td>
<td>filename()</td>
</tr>
<tr>
<td>31.55.3.5</td>
<td>find()</td>
</tr>
<tr>
<td>31.55.3.6</td>
<td>handle()</td>
</tr>
<tr>
<td>31.55.3.7</td>
<td>leftline() [1/2]</td>
</tr>
<tr>
<td>31.55.3.8</td>
<td>leftline() [2/2]</td>
</tr>
<tr>
<td>31.55.3.9</td>
<td>link()</td>
</tr>
<tr>
<td>31.55.3.10</td>
<td>load()</td>
</tr>
<tr>
<td>31.55.3.11</td>
<td>resize()</td>
</tr>
<tr>
<td>31.55.3.12</td>
<td>scrollbar_size() [1/2]</td>
</tr>
<tr>
<td>31.55.3.13</td>
<td>scrollbar_size() [2/2]</td>
</tr>
<tr>
<td>31.55.3.14</td>
<td>select_all()</td>
</tr>
<tr>
<td>31.55.3.15</td>
<td>size()</td>
</tr>
<tr>
<td>31.55.3.16</td>
<td>textcolor() [1/2]</td>
</tr>
<tr>
<td>31.55.3.17</td>
<td>textcolor() [2/2]</td>
</tr>
<tr>
<td>31.55.3.18</td>
<td>textfont() [1/2]</td>
</tr>
<tr>
<td>31.55.3.19</td>
<td>textfont() [2/2]</td>
</tr>
<tr>
<td>31.55.3.20</td>
<td>textsize() [1/2]</td>
</tr>
<tr>
<td>31.55.3.21</td>
<td>textsize() [2/2]</td>
</tr>
<tr>
<td>31.55.3.22</td>
<td>title()</td>
</tr>
<tr>
<td>31.55.3.23</td>
<td>topline() [1/3]</td>
</tr>
<tr>
<td>31.55.3.24</td>
<td>topline() [2/3]</td>
</tr>
<tr>
<td>31.55.3.25</td>
<td>topline() [3/3]</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>31.55.3.2</td>
<td>value() [1/2]</td>
</tr>
<tr>
<td>31.55.3.2</td>
<td>value() [2/2]</td>
</tr>
<tr>
<td>31.56</td>
<td>Fl_Hold_Browser Class Reference</td>
</tr>
<tr>
<td>31.56.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.56.2</td>
<td>Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>31.56.2.1</td>
<td>Fl_Hold_Browser()</td>
</tr>
<tr>
<td>31.57</td>
<td>Fl_Hor_Fill_Slider Class Reference</td>
</tr>
<tr>
<td>31.58</td>
<td>Fl_Hor_Nice_Slider Class Reference</td>
</tr>
<tr>
<td>31.59</td>
<td>Fl_Hor_Slider Class Reference</td>
</tr>
<tr>
<td>31.59.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.60</td>
<td>Fl_Hor_Value_Slider Class Reference</td>
</tr>
<tr>
<td>31.61</td>
<td>Fl_Image Class Reference</td>
</tr>
<tr>
<td>31.61.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.61.2</td>
<td>Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>31.61.2.1</td>
<td>Fl_Image()</td>
</tr>
<tr>
<td>31.61.3</td>
<td>Member Function Documentation</td>
</tr>
<tr>
<td>31.61.3.1</td>
<td>as_shared_image()</td>
</tr>
<tr>
<td>31.61.3.2</td>
<td>color_average()</td>
</tr>
<tr>
<td>31.61.3.3</td>
<td>copy() [1/2]</td>
</tr>
<tr>
<td>31.61.3.4</td>
<td>copy() [2/2]</td>
</tr>
<tr>
<td>31.61.3.5</td>
<td>count()</td>
</tr>
<tr>
<td>31.61.3.6</td>
<td>d()</td>
</tr>
<tr>
<td>31.61.3.7</td>
<td>data()</td>
</tr>
<tr>
<td>31.61.3.8</td>
<td>desaturate()</td>
</tr>
<tr>
<td>31.61.3.9</td>
<td>draw() [1/2]</td>
</tr>
<tr>
<td>31.61.3.10</td>
<td>draw() [2/2]</td>
</tr>
<tr>
<td>31.61.3.11</td>
<td>draw_empty()</td>
</tr>
<tr>
<td>31.61.3.12</td>
<td>draw_scaled()</td>
</tr>
<tr>
<td>31.61.3.13</td>
<td>fail()</td>
</tr>
<tr>
<td>31.61.3.14</td>
<td>h() [1/2]</td>
</tr>
</tbody>
</table>
31.63.3.9 translate() ................................................. 633
31.64 Fl_Input Class Reference ....................................... 634
  31.64.1 Detailed Description ........................................ 634
  31.64.2 Constructor & Destructor Documentation .................... 636
    31.64.2.1 Fl_Input() .............................................. 636
  31.64.3 Member Function Documentation ................................ 636
    31.64.3.1 draw() ................................................ 636
    31.64.3.2 handle() .............................................. 636
31.65 Fl_Input_ Class Reference ...................................... 637
  31.65.1 Detailed Description ........................................ 640
  31.65.2 Constructor & Destructor Documentation .................... 640
    31.65.2.1 Fl_Input__() ........................................... 640
    31.65.2.2 ~Fl_Input_() ......................................... 641
  31.65.3 Member Function Documentation ................................ 641
    31.65.3.1 append() .............................................. 641
    31.65.3.2 copy() ............................................... 641
    31.65.3.3 copy_cuts() .......................................... 642
    31.65.3.4 cursor_color() [1/2] .................................. 642
    31.65.3.5 cursor_color() [2/2] .................................. 642
    31.65.3.6 cut() [1/3] ............................................ 643
    31.65.3.7 cut() [2/3] ............................................ 643
    31.65.3.8 cut() [3/3] ............................................ 643
    31.65.3.9 drawtext() ............................................ 644
    31.65.3.10 handle_mouse() ....................................... 644
    31.65.3.11 handletext() ......................................... 645
    31.65.3.12 index() .............................................. 645
    31.65.3.13 input_type() [1/2] .................................... 645
    31.65.3.14 input_type() [2/2] .................................... 645
    31.65.3.15 insert() .............................................. 646
    31.65.3.16 line_end() ........................................... 646

31.73.3.6 down_box() ................................................................. 682
31.73.3.7 find_index() [1/3] ...................................................... 682
31.73.3.8 find_index() [2/3] ...................................................... 683
31.73.3.9 find_index() [3/3] ...................................................... 683
31.73.3.10 find_item() [1/2] ...................................................... 684
31.73.3.11 find_item() [2/2] ...................................................... 685
31.73.3.12 global() ................................................................. 685
31.73.3.13 insert() ................................................................. 685
31.73.3.14 item_pathname() .................................................... 686
31.73.3.15 menu() [1/2] ............................................................ 687
31.73.3.16 menu() [2/2] ............................................................ 687
31.73.3.17 menu_end() ............................................................ 688
31.73.3.18 mode() [1/2] ........................................................... 688
31.73.3.19 mode() [2/2] ........................................................... 688
31.73.3.20 mvalue() ............................................................... 689
31.73.3.21 picked() ............................................................... 689
31.73.3.22 remove() ............................................................... 689
31.73.3.23 replace() ............................................................... 689
31.73.3.24 setonly() ............................................................... 690
31.73.3.25 shortcut() ............................................................. 690
31.73.3.26 size() ................................................................. 690
31.73.3.27 test_shortcut() ....................................................... 690
31.73.3.28 text() [1/2] ............................................................ 691
31.73.3.29 text() [2/2] ............................................................ 691
31.73.3.30 textcolor() [1/2] ...................................................... 691
31.73.3.31 textcolor() [2/2] ...................................................... 691
31.73.3.32 textfont() [1/2] ....................................................... 691
31.73.3.33 textfont() [2/2] ....................................................... 691
31.73.3.34 textsize() [1/2] ....................................................... 692
31.73.3.35 textsize() [2/2] ....................................................... 692
<table>
<thead>
<tr>
<th>31.73.3.3 &amp;value() [1/3]</th>
<th>692</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.73.3.3 &amp;value() [2/3]</td>
<td>692</td>
</tr>
<tr>
<td>31.73.3.3 &amp;value() [3/3]</td>
<td>692</td>
</tr>
</tbody>
</table>

31.74 Fl_Menu_Bar Class Reference

### 31.74.1 Detailed Description

### 31.74.2 Constructor & Destructor Documentation

#### 31.74.2.1 Fl_Menu_Bar()

### 31.74.3 Member Function Documentation

#### 31.74.3.1 draw()

#### 31.74.3.2 handle()

#### 31.74.3.3 update()

31.75 Fl_Menu_Button Class Reference

### 31.75.1 Detailed Description

### 31.75.2 Member Enumeration Documentation

#### 31.75.2.1 popup_buttons

### 31.75.3 Constructor & Destructor Documentation

#### 31.75.3.1 Fl_Menu_Button()

### 31.75.4 Member Function Documentation

#### 31.75.4.1 draw()

#### 31.75.4.2 handle()

#### 31.75.4.3 popup()

31.76 Fl_Menu_Item Struct Reference

### 31.76.1 Detailed Description

### 31.76.2 Member Function Documentation

#### 31.76.2.1 activate()

#### 31.76.2.2 active()

#### 31.76.2.3 activeVisible()

#### 31.76.2.4 add()

#### 31.76.2.5 argument() [1/2]

#### 31.76.2.6 argument() [2/2]
31.76.2.7 callback() ........................................ 706
31.76.2.8 callback() ........................................ 706
31.76.2.9 callback() ........................................ 706
31.76.2.10 callback() ...................................... 707
31.76.2.11 callback() ...................................... 707
31.76.2.12 check() .......................................... 707
31.76.2.13 checkbox() ...................................... 707
31.76.2.14 checked() ........................................ 708
31.76.2.15 clear() ........................................... 708
31.76.2.16 deactivate() ..................................... 708
31.76.2.17 do_callback() ................................... 708
31.76.2.18 do_callback() ................................... 708
31.76.2.19 do_callback() ................................... 709
31.76.2.20 draw() ............................................ 709
31.76.2.21 find_shortcut() .................................. 709
31.76.2.22 first() ............................................ 709
31.76.2.23 first() ............................................ 710
31.76.2.24 hide() ............................................ 710
31.76.2.25 image() .......................................... 710
31.76.2.26 image() .......................................... 710
31.76.2.27 insert() .......................................... 710
31.76.2.28 label() ............................................ 711
31.76.2.29 labelcolor() .................................... 711
31.76.2.30 labelcolor() .................................... 711
31.76.2.31 labelfont() ...................................... 712
31.76.2.32 labelfont() ...................................... 712
31.76.2.33 labelsize() ....................................... 712
31.76.2.34 labelsize() ....................................... 712
31.76.2.35 labeltype() ...................................... 713
31.76.2.36 labeltype() ...................................... 713
31.96.2.10 scale() ........................................ 798
31.96.2.11 set_current() .................................. 799
31.96.2.12 translate() ..................................... 799
31.97 Fl_Progress Class Reference ................................ 799
31.97.1 Detailed Description .................................. 800
31.97.2 Constructor & Destructor Documentation .................... 800
31.97.2.1 Fl_Progress() ..................................... 800
31.97.3 Member Function Documentation .......................... 801
31.97.3.1 draw() ........................................ 801
31.97.3.2 maximum() [1/2] .................................. 801
31.97.3.3 maximum() [2/2] .................................. 801
31.97.3.4 minimum() [1/2] .................................. 801
31.97.3.5 minimum() [2/2] .................................. 801
31.97.3.6 value() [1/2] ...................................... 802
31.97.3.7 value() [2/2] ...................................... 802
31.98 Fl_Radio_Button Class Reference ............................... 802
31.98.1 Constructor & Destructor Documentation .................... 802
31.98.1.1 Fl_Radio_Button() .................................. 802
31.99 Fl_Radio_Light_Button Class Reference ....................... 803
31.100 Fl_Radio_Round_Button Class Reference ..................... 803
31.100.1 Constructor & Destructor Documentation .................. 804
31.100.1.1 Fl_Radio_Round_Button() .......................... 804
31.101 Fl_Rect Class Reference ................................ 804
31.101.1 Detailed Description .................................. 805
31.101.2 Constructor & Destructor Documentation .................. 805
31.101.2.1 Fl_Rect() [1/5] .................................... 805
31.101.2.2 Fl_Rect() [2/5] .................................... 806
31.101.2.3 Fl_Rect() [3/5] .................................... 806
31.101.2.4 Fl_Rect() [4/5] .................................... 806
31.101.2.5 Fl_Rect() [5/5] .................................... 806

Generated by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.110.3 clear()</td>
<td>Page 827</td>
</tr>
<tr>
<td>31.110.3 draw()</td>
<td>Page 828</td>
</tr>
<tr>
<td>31.110.3 handle()</td>
<td>Page 828</td>
</tr>
<tr>
<td>31.110.3 recalc_scrollbars()</td>
<td>Page 829</td>
</tr>
<tr>
<td>31.110.3 resize()</td>
<td>Page 829</td>
</tr>
<tr>
<td>31.110.3 scroll_to()</td>
<td>Page 830</td>
</tr>
<tr>
<td>31.110.3 scrollbar_size() [1/2]</td>
<td>Page 830</td>
</tr>
<tr>
<td>31.110.3 scrollbar_size() [2/2]</td>
<td>Page 830</td>
</tr>
<tr>
<td>31.110.3 xposition()</td>
<td>Page 831</td>
</tr>
<tr>
<td>31.110.3 yposition()</td>
<td>Page 831</td>
</tr>
<tr>
<td>31.111 Fl_Scrollbar Class Reference</td>
<td>Page 831</td>
</tr>
<tr>
<td>31.111.1 Detailed Description</td>
<td>Page 832</td>
</tr>
<tr>
<td>31.111.2 Constructor &amp; Destructor Documentation</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.2.1 Fl_Scrollbar()</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.2.2 Fl_Scrollbar()</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.3 Member Function Documentation</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.3.1 draw()</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.3.2 handle()</td>
<td>Page 833</td>
</tr>
<tr>
<td>31.111.3.3 linesize()</td>
<td>Page 834</td>
</tr>
<tr>
<td>31.111.3.4 value() [1/3]</td>
<td>Page 834</td>
</tr>
<tr>
<td>31.111.3.5 value() [2/3]</td>
<td>Page 834</td>
</tr>
<tr>
<td>31.111.3.6 value() [3/3]</td>
<td>Page 835</td>
</tr>
<tr>
<td>31.112 Fl::ScrollInfo::Fl_Scrollbar_Data Struct Reference</td>
<td>Page 835</td>
</tr>
<tr>
<td>31.112.1 Detailed Description</td>
<td>Page 836</td>
</tr>
<tr>
<td>31.113 Fl_Secret_Input Class Reference</td>
<td>Page 836</td>
</tr>
<tr>
<td>31.113.1 Detailed Description</td>
<td>Page 836</td>
</tr>
<tr>
<td>31.113.2 Constructor &amp; Destructor Documentation</td>
<td>Page 836</td>
</tr>
<tr>
<td>31.113.2.1 Fl_Secret_Input()</td>
<td>Page 837</td>
</tr>
<tr>
<td>31.113.3 Member Function Documentation</td>
<td>Page 837</td>
</tr>
<tr>
<td>31.113.3.1 handle()</td>
<td>Page 837</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>31.11 Fl_Simple_Terminal Class Reference</td>
<td>849</td>
</tr>
<tr>
<td>Detailed Description</td>
<td>850</td>
</tr>
<tr>
<td>Member Function Documentation</td>
<td>852</td>
</tr>
<tr>
<td>31.117.1 ansi()</td>
<td>852</td>
</tr>
<tr>
<td>31.117.2 append()</td>
<td>854</td>
</tr>
<tr>
<td>clear()</td>
<td>854</td>
</tr>
<tr>
<td>current_style_index()</td>
<td>854</td>
</tr>
<tr>
<td>31.117.2.1 draw()</td>
<td>855</td>
</tr>
<tr>
<td>31.117.2.2 enforce_history_lines()</td>
<td>855</td>
</tr>
<tr>
<td>enforce_stay_at_bottom()</td>
<td>855</td>
</tr>
<tr>
<td>history_lines()</td>
<td>855</td>
</tr>
<tr>
<td>31.117.2.3 normal_style_index()</td>
<td>856</td>
</tr>
<tr>
<td>31.117.2.4 printf()</td>
<td>857</td>
</tr>
<tr>
<td>remove_lines()</td>
<td>857</td>
</tr>
<tr>
<td>stay_at_bottom()</td>
<td>857</td>
</tr>
<tr>
<td>style_table()</td>
<td>858</td>
</tr>
<tr>
<td>style_table_size()</td>
<td>858</td>
</tr>
<tr>
<td>text()</td>
<td>859</td>
</tr>
<tr>
<td>vprintf()</td>
<td>860</td>
</tr>
<tr>
<td>31.11 Fl_Single_Window Class Reference</td>
<td>861</td>
</tr>
<tr>
<td>Detailed Description</td>
<td>861</td>
</tr>
<tr>
<td>Member Function Documentation</td>
<td>862</td>
</tr>
<tr>
<td>31.118.1 show()</td>
<td>862</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>31.120.3.14</td>
<td>textfont()</td>
</tr>
<tr>
<td>31.120.3.15</td>
<td>textfont()</td>
</tr>
<tr>
<td>31.120.3.16</td>
<td>textsize()</td>
</tr>
<tr>
<td>31.120.3.17</td>
<td>textsize()</td>
</tr>
<tr>
<td>31.120.3.18</td>
<td>type()</td>
</tr>
<tr>
<td>31.120.3.19</td>
<td>type()</td>
</tr>
<tr>
<td>31.120.3.20</td>
<td>value()</td>
</tr>
<tr>
<td>31.120.3.21</td>
<td>value()</td>
</tr>
<tr>
<td>31.120.3.22</td>
<td>wrap()</td>
</tr>
<tr>
<td>31.120.3.23</td>
<td>wrap()</td>
</tr>
<tr>
<td>31.121</td>
<td>Fl_Spinner::Fl_Spinner_Input Class Reference</td>
</tr>
<tr>
<td>31.121.1</td>
<td>Member Function Documentation</td>
</tr>
<tr>
<td>31.121.1.1</td>
<td>handle()</td>
</tr>
<tr>
<td>31.122</td>
<td>Fl_Surface_Device Class Reference</td>
</tr>
<tr>
<td>31.122.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.122.2</td>
<td>Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>31.122.2.1</td>
<td>Fl_Surface_Device()</td>
</tr>
<tr>
<td>31.122.2.2</td>
<td>~Fl_Surface_Device()</td>
</tr>
<tr>
<td>31.122.3</td>
<td>Member Function Documentation</td>
</tr>
<tr>
<td>31.122.3.1</td>
<td>driver()</td>
</tr>
<tr>
<td>31.122.3.2</td>
<td>driver()</td>
</tr>
<tr>
<td>31.122.3.3</td>
<td>pop_current()</td>
</tr>
<tr>
<td>31.122.3.4</td>
<td>push_current()</td>
</tr>
<tr>
<td>31.122.3.5</td>
<td>set_current()</td>
</tr>
<tr>
<td>31.122.3.6</td>
<td>surface()</td>
</tr>
<tr>
<td>31.123</td>
<td>Fl_SVG_File_Surface Class Reference</td>
</tr>
<tr>
<td>31.123.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>31.123.2</td>
<td>Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>31.123.2.1</td>
<td>Fl_SVG_File_Surface()</td>
</tr>
<tr>
<td>31.123.2.2</td>
<td>~Fl_SVG_File_Surface()</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>col_header()</td>
<td>912</td>
</tr>
<tr>
<td>draw()</td>
<td>913</td>
</tr>
<tr>
<td>row_header()</td>
<td>918</td>
</tr>
<tr>
<td>recalc_dimensions()</td>
<td>917</td>
</tr>
<tr>
<td>set_selection()</td>
<td>920</td>
</tr>
<tr>
<td>tab_cell-nav()</td>
<td>921</td>
</tr>
<tr>
<td>insert()</td>
<td>916</td>
</tr>
<tr>
<td>is_selected()</td>
<td>917</td>
</tr>
<tr>
<td>find_cell()</td>
<td>916</td>
</tr>
<tr>
<td>do_callback()</td>
<td>913</td>
</tr>
<tr>
<td>get_selection()</td>
<td>916</td>
</tr>
<tr>
<td>init_sizes()</td>
<td>916</td>
</tr>
<tr>
<td>row_resize()</td>
<td>919</td>
</tr>
<tr>
<td>row_resize_min()</td>
<td>919</td>
</tr>
<tr>
<td>scrollbar_size()</td>
<td>919</td>
</tr>
<tr>
<td>is_interactive_resize()</td>
<td>916</td>
</tr>
<tr>
<td>redraw_range()</td>
<td>918</td>
</tr>
<tr>
<td>row_height()</td>
<td>919</td>
</tr>
<tr>
<td>row_height_all()</td>
<td>919</td>
</tr>
<tr>
<td>col_resize()</td>
<td>912</td>
</tr>
<tr>
<td>col_resize_min()</td>
<td>912</td>
</tr>
<tr>
<td>col_width()</td>
<td>912</td>
</tr>
<tr>
<td>col_width_all()</td>
<td>912</td>
</tr>
<tr>
<td>move_cursor()</td>
<td>917</td>
</tr>
<tr>
<td>damage_rowcol()</td>
<td>913</td>
</tr>
<tr>
<td>damage_zone()</td>
<td>913</td>
</tr>
<tr>
<td>draw_cell()</td>
<td>914</td>
</tr>
<tr>
<td>scrollbar_size()</td>
<td>919</td>
</tr>
<tr>
<td>scrollbar_size()</td>
<td>920</td>
</tr>
<tr>
<td>damage_rowcol()</td>
<td>913</td>
</tr>
<tr>
<td>damage_zone()</td>
<td>913</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>loadfile()</td>
<td>948</td>
</tr>
<tr>
<td>next_char()</td>
<td>948</td>
</tr>
<tr>
<td>outputfile()</td>
<td>949</td>
</tr>
<tr>
<td>prev_char()</td>
<td>949</td>
</tr>
<tr>
<td>printf()</td>
<td>949</td>
</tr>
<tr>
<td>remove()</td>
<td>950</td>
</tr>
<tr>
<td>remove_()</td>
<td>950</td>
</tr>
<tr>
<td>replace()</td>
<td>951</td>
</tr>
<tr>
<td>rewind_lines()</td>
<td>952</td>
</tr>
<tr>
<td>savefile()</td>
<td>952</td>
</tr>
<tr>
<td>search_backward()</td>
<td>952</td>
</tr>
<tr>
<td>search_forward()</td>
<td>953</td>
</tr>
<tr>
<td>secondary_selection_text()</td>
<td>953</td>
</tr>
<tr>
<td>selection_text()</td>
<td>954</td>
</tr>
<tr>
<td>skip_displayed_characters()</td>
<td>954</td>
</tr>
<tr>
<td>tab_distance()</td>
<td>954</td>
</tr>
<tr>
<td>text() [1/2]</td>
<td>954</td>
</tr>
<tr>
<td>text() [2/2]</td>
<td>955</td>
</tr>
<tr>
<td>text_range()</td>
<td>956</td>
</tr>
<tr>
<td>vprintf()</td>
<td>956</td>
</tr>
<tr>
<td>word_end()</td>
<td>957</td>
</tr>
<tr>
<td>word_start()</td>
<td>957</td>
</tr>
</tbody>
</table>

31.129.4 Member Data Documentation .................................................. 957

31.129.4.1 file_encoding_warning_message ............................................. 957
31.129.4.2 mPredeleteProcs .................................................................... 958
31.129.4.3 mTabDist ............................................................................. 958
31.129.4.4 transcoding_warning_action ................................................... 958

31.130 Fl_Text_Display Class Reference .................................................... 958

31.130.1 Detailed Description .................................................................... 965

31.130.2 Member Enumeration Documentation ............................................. 966
31.130.4.85 update_h_scrollbar() .................................................. 1000
31.130.4.86 update_line_starts() .................................................. 1000
31.130.4.87 update_v_scrollbar() .................................................. 1000
31.130.4.88 line_length() .......................................................... 1001
31.130.4.89 word_end() ............................................................ 1001
31.130.4.90 word_start() ........................................................... 1001
31.130.4.91 wrap_mode() ............................................................ 1002
31.130.4.92 wrap_uses_character() ................................................ 1002
31.130.4.93 wrapped_column() ...................................................... 1003
31.130.4.94 wrapped_line_counter() .............................................. 1003
31.130.4.95 wrapped_row() .......................................................... 1004
31.130.4.96 to_col() ................................................................ 1005
31.130.4.97 to_rowcol() ................................................................ 1005
31.131 Fl_Text_Editor Class Reference .............................................. 1006
31.131.1 Detailed Description .......................................................... 1009
31.131.2 Member Typedef Documentation ........................................... 1009
   31.131.2.1 Key_Func .................................................................. 1009
31.131.3 Constructor & Destructor Documentation .................................. 1009
   31.131.3.1 Fl_Text_Editor() ....................................................... 1009
31.131.4 Member Function Documentation ......................................... 1009
   31.131.4.1 add_default_key_bindings() ....................................... 1010
   31.131.4.2 add_key_binding() .................................................. 1010
   31.131.4.3 bound_key_function() .............................................. 1010
   31.131.4.4 default_key_function() ........................................... 1011
   31.131.4.5 insert_mode() ......................................................... 1011
   31.131.4.6 kf_backspace() ....................................................... 1011
31.131. Member Data Documentation ........................................ 1019
    31.131.5. global_key_bindings ........................................ 1020

31.132. Fl_Text_Selection Class Reference .................................... 1020
    31.132. Detailed Description ........................................... 1021
    31.132. Member Function Documentation .................................... 1021
    31.132.2.1end() .................................................. 1022
    31.132.2.includes() .................................................. 1022
    31.132.2.3length() .................................................. 1022
    31.132.2.4position() ............................................... 1022
    31.132.2.5selected() [1/2] ........................................... 1023
    31.132.2.6selected() [2/2] ........................................... 1023
    31.132.2.7set() .................................................... 1024
    31.132.2.8start() ................................................... 1024
    31.132.2.9update() .................................................. 1024

31.133. Fl_Tile Class Reference ........................................... 1025
    31.133. Detailed Description ........................................... 1025
    31.133. Constructor & Destructor Documentation ............................ 1027
    31.133.2. Fl_Tile() ................................................ 1027
    31.133. Member Function Documentation .................................... 1027
    31.133.3. handle() .................................................. 1027
    31.133.3. position() ................................................ 1028
    31.133.3. resize() ................................................... 1028

31.134. Fl_Tiled_Image Class Reference ..................................... 1029
    31.134. Detailed Description .......................................... 1029
    31.134. Constructor & Destructor Documentation ............................ 1030
    31.134.2. Fl_Tiled_Image() ........................................... 1030
    31.134. Member Function Documentation .................................... 1030
    31.134.3. color_average() .......................................... 1030
    31.134.3. copy() .................................................... 1031
    31.134.3. desaturate() .............................................. 1031
31.137.2.137.2.12 hidedelay() [1/2] .............................................. 1041
31.137.2.137.2.13 hidedelay() [2/2] .............................................. 1041
31.137.2.137.2.14 hoverdelay() [1/2] .............................................. 1041
31.137.2.137.2.15 hoverdelay() [2/2] .............................................. 1041
31.137.2.137.2.16 margin_height() [1/2] .............................................. 1041
31.137.2.137.2.17 margin_height() [2/2] .............................................. 1041
31.137.2.137.2.18 margin_width() [1/2] .............................................. 1042
31.137.2.137.2.19 margin_width() [2/2] .............................................. 1042
31.137.2.138 Fl_Tree Class Reference .............................................. 1043
31.138.1 Detailed Description ...................................................... 1050
31.138.2 Member Function Documentation .............................................. 1054
  31.138.2.1 add() [1/2] ......................................................... 1055
  31.138.2.2 add() [2/2] ......................................................... 1055
  31.138.2.3 calc_dimensions() ..................................................... 1056
  31.138.2.4 calc_tree() .......................................................... 1056
  31.138.2.5 callback_item() [1/2] ................................................. 1057
  31.138.2.6 callback_item() [2/2] ................................................. 1057
  31.138.2.7 callback_reason() [1/2] .............................................. 1057
  31.138.2.8 callback_reason() [2/2] .............................................. 1057
  31.138.2.9 clear() ............................................................... 1058
  31.138.2.10 clear_children() ................................................... 1058
  31.138.2.11 close() [1/2] ....................................................... 1059
  31.138.2.12 close() [2/2] ....................................................... 1059
  31.138.2.13 closeicon() [1/2] ................................................... 1059
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>closeicon()</td>
<td>1059</td>
</tr>
<tr>
<td>connectorstyle()</td>
<td>1060</td>
</tr>
<tr>
<td>deselect()</td>
<td>1060</td>
</tr>
<tr>
<td>deselect_all()</td>
<td>1061</td>
</tr>
<tr>
<td>displayed()</td>
<td>1062</td>
</tr>
<tr>
<td>extend_selection()</td>
<td>1062</td>
</tr>
<tr>
<td>extend_selection_dir()</td>
<td>1063</td>
</tr>
<tr>
<td>get_selected_items()</td>
<td>1066</td>
</tr>
<tr>
<td>handle()</td>
<td>1067</td>
</tr>
<tr>
<td>insert()</td>
<td>1068</td>
</tr>
<tr>
<td>insert_above()</td>
<td>1069</td>
</tr>
<tr>
<td>is_close()</td>
<td>1069</td>
</tr>
<tr>
<td>is_hscroll_visible()</td>
<td>1070</td>
</tr>
<tr>
<td>is_open()</td>
<td>1071</td>
</tr>
<tr>
<td>is_scrollbar()</td>
<td>1072</td>
</tr>
<tr>
<td>is_selected()</td>
<td>1072</td>
</tr>
<tr>
<td>is_vscroll_visible()</td>
<td>1073</td>
</tr>
</tbody>
</table>
31.138.2.70 calc_tree() ..................................................... 1085
31.138.2.70 move() ....................................................... 1085
31.138.2.70 size() ........................................................ 1086
31.138.2.70 bot() .......................................................... 1086
31.138.2.70 top_label() ................................................... 1086
31.138.2.70 scrollbar_size() [1/2] ....................................... 1087
31.138.2.70 scrollbar_size() [2/2] ....................................... 1087
31.138.2.70 select() [1/2] .................................................. 1087
31.138.2.70 select() [2/2] .................................................. 1088
31.138.2.70 select_all() ................................................... 1089
31.138.2.70 select_only() .................................................. 1089
31.138.2.70 select_toggle() ............................................... 1090
31.138.2.70 selectbox() [1/2] .............................................. 1090
31.138.2.70 selectbox() [2/2] .............................................. 1091
31.138.2.70 selectmode() [1/2] .......................................... 1091
31.138.2.70 selectmode() [2/2] .......................................... 1091
31.138.2.70 set_item_focus() ............................................ 1091
31.138.2.70 show_item() [1/2] ............................................ 1091
31.138.2.70 show_item() [2/2] ............................................ 1092
31.138.2.70 show_item_bottom() ........................................ 1092
31.138.2.70 show_item_middle() ......................................... 1093
31.138.2.70 show_item_top() ............................................. 1093
31.138.2.70 show_self() ................................................... 1093
31.138.2.70 showcollapse() [1/2] ....................................... 1093
31.138.2.70 showcollapse() [2/2] ....................................... 1094
31.138.2.70 showroot() .................................................... 1094
31.138.2.70 sortorder() ................................................... 1094
31.138.2.70 usericon() [1/2] .............................................. 1094
31.138.2.70 usericon() [2/2] .............................................. 1094
31.138.2.70 usericonmarginleft() [1/2] ............................... 1095
31.138.2.104 usericonmarginleft() [2/2] .................................................. 1095
31.138.2.105 vposition() [1/2] .......................................................... 1095
31.138.2.106 vposition() [2/2] .......................................................... 1095
31.138.2.107 widgetmarginleft() [1/2] .................................................. 1096
31.138.2.108 widgetmarginleft() [2/2] .................................................. 1096

31.139 Fl_Tree_Item Class Reference ......................................................... 1096

31.139.1 Detailed Description ................................................................. 1101
31.139.2 Constructor & Destructor Documentation ..................................... 1102
31.139.2.1 Fl_Tree_Item() [1/2] ......................................................... 1102
31.139.2.2 Fl_Tree_Item() [2/2] ......................................................... 1102

31.139.3 Member Function Documentation ............................................. 1103
31.139.3.1 activate() ..................................................................... 1103
31.139.3.2 add() [1/4] ..................................................................... 1103
31.139.3.2 add() [2/4] ..................................................................... 1103
31.139.3.2 add() [3/4] ..................................................................... 1104
31.139.3.2 add() [4/4] ..................................................................... 1104
31.139.3.6 calc_item_height() .............................................................. 1104
31.139.3.7 child() ........................................................................... 1105
31.139.3.8 deactivate() ..................................................................... 1105
31.139.3.9 deparent() ...................................................................... 1105
31.139.3.10 depth() .......................................................................... 1105
31.139.3.11 deselect_all() .................................................................. 1105
31.139.3.12 draw() ........................................................................... 1106
31.139.3.13 draw_horizontal_connector() ........................................... 1106
31.139.3.14 draw_item_content() ....................................................... 1107
31.139.3.15 draw_vertical_connector() ................................................. 1108
31.139.3.16 drawbgcolor() ................................................................ 1108
31.139.3.17 drawfgcolor() .................................................................. 1109
31.139.3.18 find_child() [1/2] .............................................................. 1109
31.139.3.19 find_child() [2/2] .............................................................. 1109

Generated by Doxygen
31.139.3.30nd_child_item() [1/2] ................................................. 1110
31.139.3.31nd_child_item() [2/2] ................................................. 1110
31.139.3.32nd_clicked() ............................................................ 1110
31.139.3.33nd_item() ............................................................... 1111
31.139.3.34de_widgets() ......................................................... 1111
31.139.3.35sort() ................................................................. 1111
31.139.3.36sort_above() .......................................................... 1112
31.139.3.37label() ................................................................. 1112
31.139.3.38label_h() ............................................................... 1112
31.139.3.39label_w() ............................................................... 1112
31.139.3.40label_x() ............................................................... 1113
31.139.3.41label_y() ............................................................... 1113
31.139.3.42labelbgcolor() [1/2] ................................................. 1113
31.139.3.43labelbgcolor() [2/2] ................................................. 1113
31.139.3.44move() [1/2] ............................................................. 1114
31.139.3.45move() [2/2] ............................................................. 1114
31.139.3.46move_above() .......................................................... 1115
31.139.3.47move_below() .......................................................... 1115
31.139.3.48move_into() ............................................................ 1115
31.139.3.49next() ................................................................. 1116
31.139.3.50next_displayed() .................................................... 1116
31.139.3.51next_sibling() ........................................................ 1116
31.139.3.52next_visible() ........................................................ 1116
31.139.3.53parent() ............................................................... 1117
31.139.3.54prefs() ............................................................... 1117
31.139.3.55rev() ................................................................. 1117
31.139.3.56rev_displayed() .................................................... 1117
31.139.3.57rev_sibling() ........................................................ 1118
31.139.3.58rev_visible() ........................................................ 1118
31.139.3.59calc_tree() ........................................................... 1118
31.14 Fl_Tree_Prefs Class Reference ........................................... 1128

31.14.1 Detailed Description .................................................... 1131

31.14.2 Member Function Documentation ...................................... 1131

31.14.2.1 closedeicon() ..................................................... 1132

31.14.2.2 closeicon() ...................................................... 1132

31.14.2.3 item_draw_mode() ............................................... 1132

31.14.2.4 item_labelbgcolor() ............................................. 1132

31.14.2.5 item_labelbgcolor() ............................................. 1132

31.14.2.6 marginbottom() .................................................. 1133

31.14.2.7 marginbottom() .................................................. 1133

31.14.2.8 opendeicon() .................................................... 1133

31.14.2.9 openicon() ....................................................... 1133

31.14.3 Fl_Valuator Class Reference .......................................... 1135

31.14.2 Detailed Description .................................................... 1137

31.14.2 Constructor & Destructor Documentation .......................... 1137

31.14.2.1 Fl_Valuator() ................................................... 1137

31.14.3 Member Function Documentation ...................................... 1138

31.14.3.1 bounds() .......................................................... 1138

31.14.3.2 clamp() .......................................................... 1138

31.14.3.3 format() .......................................................... 1138
<table>
<thead>
<tr>
<th>Member Function Documentation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_window()</td>
<td>1215</td>
</tr>
<tr>
<td>border()</td>
<td>1215</td>
</tr>
<tr>
<td>clear_border()</td>
<td>1215</td>
</tr>
<tr>
<td>clear_modal_states()</td>
<td>1216</td>
</tr>
<tr>
<td>current()</td>
<td>1216</td>
</tr>
<tr>
<td>cursor() [1/3]</td>
<td>1217</td>
</tr>
<tr>
<td>cursor() [2/3]</td>
<td>1217</td>
</tr>
<tr>
<td>cursor() [3/3]</td>
<td>1217</td>
</tr>
<tr>
<td>decorated_h()</td>
<td>1218</td>
</tr>
<tr>
<td>decorated_w()</td>
<td>1218</td>
</tr>
<tr>
<td>default_cursor() [1/2]</td>
<td>1218</td>
</tr>
<tr>
<td>default_cursor() [2/2]</td>
<td>1218</td>
</tr>
<tr>
<td>default_icon()</td>
<td>1218</td>
</tr>
<tr>
<td>default_icons()</td>
<td>1219</td>
</tr>
<tr>
<td>default_xclass() [1/2]</td>
<td>1219</td>
</tr>
<tr>
<td>default_xclass() [2/2]</td>
<td>1220</td>
</tr>
<tr>
<td>draw()</td>
<td>1220</td>
</tr>
<tr>
<td>flush()</td>
<td>1221</td>
</tr>
<tr>
<td>force_position() [1/2]</td>
<td>1221</td>
</tr>
<tr>
<td>force_position() [2/2]</td>
<td>1221</td>
</tr>
<tr>
<td>free_icons()</td>
<td>1221</td>
</tr>
<tr>
<td>free_position()</td>
<td>1222</td>
</tr>
<tr>
<td>fullscreen()</td>
<td>1222</td>
</tr>
<tr>
<td>fullscreen_screens()</td>
<td>1222</td>
</tr>
<tr>
<td>handle()</td>
<td>1223</td>
</tr>
<tr>
<td>hide()</td>
<td>1224</td>
</tr>
<tr>
<td>hotspot()</td>
<td>1224</td>
</tr>
<tr>
<td>icon() [1/3]</td>
<td>1225</td>
</tr>
<tr>
<td>icon() [2/3]</td>
<td>1226</td>
</tr>
</tbody>
</table>
32 File Documentation

32.1 Enumerations.H File Reference

32.1.1 Detailed Description

32.1.2 Macro Definition Documentation

32.1.2.1 FL_ABI_VERSION

32.1.2.2 FL_API_VERSION

32.1.2.3 FL_IMAGE_LABEL

32.1.2.4 FL_MAJOR_VERSION

32.1.2.5 FL_MINOR_VERSION

32.1.2.6 FL_MULTI_LABEL

32.1.2.7 FL_PATCH_VERSION

32.1.2.8 FL_SYMBOL_LABEL

32.1.2.9 FL_VERSION

32.1.3 Typedef Documentation

32.1.3.1 Fl_Align

32.1.3.2 Fl_Font

32.1.3.3 Fl_Fontsize

32.1.4 Enumeration Type Documentation

32.1.4.1 anonymous enum

32.1.4.2 Fl_Boxtype

32.1.4.3 Fl_Cursor

32.1.4.4 Fl_DAMAGE

32.1.4.5 Fl_Event

32.1.4.6 Fl_Labeltype

32.1.4.7 Fl_When

32.1.5 Function Documentation

32.1.5.1 fl_box()

32.1.5.2 fl_color_cube()

32.1.5.3 fl_darker()

32.1.5.4 fl_define_FL_EMBOSSED_LABEL()
32.1.5.5  fl_define_FL_ENGRAVED_LABEL()  ................................................. 1271
32.1.5.6  fl_define_FL_ICON_LABEL()  .................................................. 1271
32.1.5.7  fl_define_FL_IMAGE_LABEL()  ................................................ 1271
32.1.5.8  fl_define_FL_MULTI_LABEL() .................................................. 1271
32.1.5.9  fl_define_FL_SHADOW_LABEL() ................................................. 1271
32.1.5.10 fl_down() ............................................................................. 1271
32.1.5.11 fl_frame() ............................................................................. 1272
32.1.5.12 fl_gray_ramp() .................................................................. 1272
32.1.5.13 fl_lighter() .......................................................................... 1272
32.1.5.14 fl_rgb_color() [1/2] .............................................................. 1272
32.1.5.15 fl_rgb_color() [2/2] .............................................................. 1272
32.1.6  Variable Documentation ............................................................... 1273
    32.1.6.1 FL_ALIGN_BOTTOM .............................................................. 1273
    32.1.6.2 FL_ALIGN_CENTER ............................................................... 1273
    32.1.6.3 FL_ALIGN_CLIP ................................................................. 1273
    32.1.6.4 FL_ALIGN_IMAGE_BACKDROP ............................................ 1273
    32.1.6.5 FL_ALIGN_IMAGE_MASK ...................................................... 1273
    32.1.6.6 FL_ALIGN_IMAGE_NEXT_TO_TEXT ...................................... 1274
    32.1.6.7 FL_ALIGN_IMAGE_OVER_TEXT .......................................... 1274
    32.1.6.8 FL_ALIGN_INSIDE ............................................................... 1274
    32.1.6.9 FL_ALIGN_LEFT ................................................................. 1274
    32.1.6.10 FL_ALIGN_LEFT_BOTTOM .................................................. 1274
    32.1.6.11 FL_ALIGN_LEFT_TOP ......................................................... 1274
    32.1.6.12 FL_ALIGN_NOWRAP ............................................................ 1275
    32.1.6.13 FL_ALIGN_POSITION_MASK ............................................ 1275
    32.1.6.14 FL_ALIGN_RIGHT ............................................................. 1275
    32.1.6.15 FL_ALIGN_RIGHT_BOTTOM .............................................. 1275
    32.1.6.16 FL_ALIGN_RIGHT_TOP ....................................................... 1275
    32.1.6.17 FL_ALIGN_TEXT_NEXT_TO_IMAGE .................................... 1275
    32.1.6.18 FL_ALIGN_TEXT_OVER_IMAGE .......................................... 1276
<table>
<thead>
<tr>
<th>Preface</th>
<th>Advanced FLTK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to FLTK</td>
<td>Unicode and UTF-8 Support</td>
</tr>
<tr>
<td>FLTK Basics</td>
<td></td>
</tr>
<tr>
<td>Common Widgets and Attributes</td>
<td></td>
</tr>
<tr>
<td>• Colors</td>
<td>Appendices:</td>
</tr>
<tr>
<td>• Box Types</td>
<td>• FLTK Enumerations</td>
</tr>
<tr>
<td>• Labels and Label Types</td>
<td>• GLUT Compatibility</td>
</tr>
<tr>
<td>• Drawing Images</td>
<td>– class Fl_Glut_Window</td>
</tr>
<tr>
<td>How does resizing work?</td>
<td>• Forms Compatibility</td>
</tr>
<tr>
<td>Designing a Simple Text Editor</td>
<td></td>
</tr>
<tr>
<td>Drawing Things in FLTK</td>
<td>• Operating System Issues</td>
</tr>
<tr>
<td>Handling Events</td>
<td>• Migrating Code from FLTK 1.3 to 1.4</td>
</tr>
<tr>
<td>• Fl::event_«() methods</td>
<td>• Developer Information</td>
</tr>
<tr>
<td>• Event Propagation</td>
<td>• Software License</td>
</tr>
<tr>
<td>Adding and Extending Widgets</td>
<td></td>
</tr>
<tr>
<td>Using OpenGL</td>
<td>• Example Source Code</td>
</tr>
<tr>
<td>Programming with FLUID</td>
<td>• FAQ (Frequently Asked Questions)</td>
</tr>
<tr>
<td>• GUI Attributes</td>
<td></td>
</tr>
<tr>
<td>• Selecting and Moving Widgets</td>
<td></td>
</tr>
<tr>
<td>• Image Labels</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2

Preface

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

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.

2.1 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
- 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 1.3 to 1.4
- Developer Information
- Software License
- Example Source Code
2.2 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 p, 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.

2.3 Abbreviations

The following abbreviations are used in this manual:

X11
The X Window System version 11.

Xlib
The X Window System interface library.

MS Windows, WIN32
The Microsoft Windows Application Programmer's Interface for Windows 2000, Windows XP, Windows Vista, Windows 7 and later Windows versions. FLTK uses the preprocessor definition _WIN32 for the 32 bit and 64 bit MS Windows API.

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

2.4 Copyrights and Trademarks

FLTK is Copyright 1998-2020 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.
Chapter 3

Introduction to FLTK

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

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

3.1 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.
3.2 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, Windows or Cocoa) for maximum speed, and carefully optimized for code size and performance.

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

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

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

- Very small & fast portable 2-D drawing library to hide Xlib, Windows, or QuickDraw.

- OpenGL/Mesa drawing area widget.

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

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

- Compatibility header file for the GLUT library.

- Compatibility header file for the XForms library.

3.3 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!
3.4 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".

3.5 Building and Installing FLTK Under UNIX and Apple macOS

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

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

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

For C shell and tcsh, use:

setenv CFLAGS "-Iincludedir"
s.getenv CXXFLAGS "-Iincludedir"
s.getenv LDFLAGS "-Llibdir"

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

CXX=xlc; export CXX
setenv CXX "xlc"

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

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

- enable-cygwin
  
  Enable the Cygwin libraries under Windows

- enable-debug
  
  Enable debugging code & symbols

- disable-gl
  
  Disable OpenGL support
–disable-print
    Disable print support for an X11 platform

–enable-shared
    Enable generation of shared libraries

–enable-threads
    Enable multithreading support

–enable-xdbe
    Enable the X double-buffer extension

–enable-xf
    Enable the Xft library for anti-aliased fonts under X11

–enable-pango
    Enable the pango library for drawing any text in any script under X11.

–enable-x11
    When targeting cygwin, build with X11 GUI instead of windows GDI. Also applicable to macOS platforms supplemented with XQuartz.

–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".
3.6 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.Windows.txt" and you should read that file to determine if there are changes that may be applicable to your build environment.

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

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

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

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

Since FLTK 1.4 the project files MUST be generated with CMake. Please read "README.CMake.txt" for more information about this.

3.6.1 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 macOS build and the notes above in the section on Building and Installing FLTK Under UNIX and Apple macOS apply, in particular the descriptions of using the "configure" script and its related options.

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

.
configure
make

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

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

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.
3.6.2 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.

3.7 Internet Resources

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

WWW

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

NNTP Newsgroups

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

3.8 Reporting Bugs

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

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

For general support and questions, please use the fltk.general newsgroup (see above, "NNTP Newsgroups") or the web interface to the newsgroups at https://www.fltk.org/newsgroups.php.
Chapter 4

FLTK Basics

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

4.1 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"

```plaintext
#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.

```
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.

```
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:

```
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.

```
window->end();
```

Finally, we show the window and enter the FLTK event loop:

```
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.

**Figure 4.1 The Hello, World! Window**

### 4.1.1 Creating the Widgets

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

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

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

### 4.1.2 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 `my←Group→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(...)`.
4.1 Writing Your First FLTK Program

4.1.3 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".

4.1.4 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.
4.2 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:

```
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:

```
CC 'fltk-config --cxxflags' ...
```

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

```
CC ... -L/usr/local/lib -lfltk -lXext -lX11 -lm
gcc ... -L/usr/local/lib -lfltk -lXext -lX11 -lm
```

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.

**Note**

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:

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

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

```
CC ... 'fltk-config --use-forms --ldflags'
CC ... 'fltk-config --use-gl --ldflags'
CC ... 'fltk-config --use-images --ldflags'
CC ... 'fltk-config --use-forms --use-gl --use-images --ldflags'
```

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

```
fltk-config --compile filename.cpp
fltk-config --use-forms --compile filename.cpp
fltk-config --use-gl --compile filename.cpp
fltk-config --use-images --compile filename.cpp
fltk-config --use-forms --use-gl --use-images --compile filename.cpp
```

Any of these will create an executable named `filename`. 
4.3 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:

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

.SUFFIXES: .o .cxx
%.o: %.cxx
   $(CXX) $(CXXFLAGS) $(DEBUG) -c $<
all: $(TARGET)
   $(LINK) -o $(TARGET) $(OBJS) $(LDSTATIC)

$(TARGET): $(OBJS)
CubeMain.o: CubeMain.cxx CubeViewUI.h
CubeView.o: CubeView.cxx 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
```

4.4 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 if the compiler doesn’t do this. Currently all known Windows compilers define _WIN32 unless you use Cygwin. You must not define _WIN32 if you use Cygwin.

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

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

4.5 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/...>`.
4.6 Header Files

The proper way to include FLTK header files is:

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

**Note**

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

```c
#include <FL\Fl_xyz.H>
#include <fl/fl_xyz.h>
#include <Fl/fl_xyz.h>
```
Chapter 5

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.

5.1 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.

All of these buttons just need the corresponding `<FL/Fl_xyz_Button.H>` header file. The constructor takes the bounding box of the button and optionally a label string:
Each button has an associated `type()` which allows it to behave as a push button, toggle button, or radio button:

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

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

### 5.2 Text

FLTK provides several text widgets for displaying and receiving text:

- **FL_Input** - A one-line text input field.
- **FL_Output** - A one-line text output field.
- **FL_Multiline_Input** - A multi-line text input field.
- **FL_Multiline_Output** - A multi-line text output field.
- **FL_Text_Display** - A multi-line text display widget.
- **FL_Text_Editor** - A multi-line text editing widget.
- **FL_Help_View** - A HTML text display widget.

The `FL_Output` and `FL_Multiline_Output` widgets allow the user to copy text from the output field but not change it. The `value()` method is used to get or set the string that is displayed:

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

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

The `FL_Text_Display` and `FL_Text_Editor` widgets use an associated `FL_Text_Buffer` class for the value, instead of a simple string.
5.3 Valuators

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

- **FL_Counter** - A widget with arrow buttons that shows the current value.
- **FL.Dial** - A round knob.
- **FL_Roller** - An SGI-like dolly widget.
- **FL_Scrollbar** - A standard scrollbar widget.
- **FL_Slider** - A scrollbar with a knob.
- **FL_Value_Slider** - A slider that shows the current value.

The `value()` method gets and sets the current value of the widget. The `minimum()` and `maximum()` methods set the range of values that are reported by the widget.
5.4 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.

5.5 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.
5.6 Colors

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

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

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

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

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

- FL_FOREGROUND_COLOR
- FL_BACKGROUND_COLOR
- FL_INACTIVE_COLOR
- FL_SELECTION_COLOR

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

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

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

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

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

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

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

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

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

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

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

Note

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

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

Figure 5.3 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 5.3 is the area that is not drawn by the frame types.

5.7.1 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:
5.7 Box Types

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

**Fl_Boxtype fl_down(Fl_Boxtype b)**

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

**Fl_Boxtype fl_frame(Fl_Boxtype b)**

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

**Fl_Boxtype fl_box(Fl_Boxtype b)**

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

Adding Your Box Type

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

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

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

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

The `label()`, `align()`, `labelfont()`, `labelsizel()`, `labelltype()`, `image()`, and `deimage()` methods control the labeling of widgets.

**label()**

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 5.4 shows the available symbols.

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

- '#' forces square scaling, rather than distortion to the widget's shape.
- +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
- '$' flips the symbol horizontally, '%' flips it vertically.
- [0-9] - rotates by a multiple of 45 degrees. '5' and '6' do no rotation while the others point in the direction of that key on a numeric keypad. '0', followed by four more digits rotates the symbol by that amount in degrees.
Thus, to show a very large arrow pointing downward you would use the label string 
"@+92-">".

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

![Figure 5.5 FLTK symbols and text](image)

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

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

**labeltype()**

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

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

**Note**

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

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

**See also**

    class Fl_Multi_Label, Fl_Widget::align()

**image()** and **deimage()**

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

To make an image you use a subclass of Fl_Image.

**Making Your Own Label Types**

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

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

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

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

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

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

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

Adding Your Label Type

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

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

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

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

Making your own symbols

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

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

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

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

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

This function draws a named symbol fitting the given rectangle.
5.9 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); } ... } ...

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

5.10 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.
Chapter 6

How does resizing work?

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

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

6.1 Resizing can be disabled

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

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

6.2 Resizing can be simple

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

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

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

6.3 Resizing can be complex

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

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

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

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

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

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

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

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

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

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

---

**Figure 6.2 Complex resizing example**

---

**Figure 6.3 Resizing dialog example (a)**
The message text area would be the logical choice so that the user can expand the dialog to see if there is more of an explanation below the short error message. This results in the behaviour shown in the diagram below.

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

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

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

```cpp
dialog = new FL_Window(300, 100);
icon = new FL_Box(0, 0, 50, 50, "!");
text = new FL_Box(50, 0, 250, 40, "Out of Memory Error");
btns = new FL_Group(50, 50, 250, 50); // parent group
darn = new FL_Button(200, 50, 100, 50, "Darn!");
R = new FL_Box(50, 50, 150, 50); // "invisible" box "R"
R->hide(); // make sure it's invisible
btns->resizable(R); // make "R" parent group resizable
btns->end();
dialog->resizable(text);
dialog->end();
```
Imagine instead that you have a group that has a button, an input field, another button and a second input field, all next to each other, and you want the input fields to resize equally, but not the buttons. How could you achieve this?

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

The answer is to leave the resizable of the group set to itself, and to create two equal size subgroups, each of which will resize equally. Add a button and input field to each subgroup, and set each subgroup’s resizable to the input field, as shown below. Tada!

In FLTK it is possible to solve almost any layout and resizing problem by introducing an invisible box into a group, or an extra group into the widget hierarchy. It might take some thought to achieve exactly what you want and sometimes it is necessary to introduce parallel hierarchies in order to get widgets in different groups to resize together.
How does resizing work?
Chapter 7

Designing a Simple Text Editor

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

7.1 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.

7.2 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];
};
```
7.3 Variables

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

```c
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.

7.4 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:

```c
FL_Menu_Item menuitems[] = {
    { "File", 0, 0, 0, FL_SUBMENU },
    { "New File", 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 },
    { "Exit", FL_COMMAND + 'q', (Fl_Callback *)quit_cb, 0 },

    { "Edit", 0, 0, 0, FL_SUBMENU },
    { "Undo", FL_COMMAND + 'z', (Fl_Callback *)undo_cb, 0, FL_MENU_DIVIDER },
    { "Cut", FL_COMMAND + 'x', (Fl_Callback *)cut_cb },
    { "Copy", FL_COMMAND + 'c', (Fl_Callback *)copy_cb },
    { "Paste", FL_COMMAND + 'v', (Fl_Callback *)paste_cb },
    { "Delete", 0, (Fl_Callback *)delete_cb },

    { "Search", 0, 0, 0, FL_SUBMENU },
    { "Find...", FL_COMMAND + 'f', (Fl_Callback *)find_cb },
    { "Find Again", FL_COMMAND + 'g', find2_cb },
    { "Replace...", FL_COMMAND + 'r', replace_cb },
    { "Replace Again", FL_COMMAND + 't', replace2_cb },

    { 0 },
};
```

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

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

We'll define the callback functions later.
7.5 Editing the Text

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

\[ w->editor = new \text{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:

\[ \text{textbuf}\rightarrow add\_modify\_callback(changed\_cb, w); \]
\[ \text{textbuf}\rightarrow call\_modify\_callbacks(); \]

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

\[ w->editor->textfont(FL\_COURIER); \]

7.6 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 7.1 The search and replace dialog](image)

Fl_Window *replace_dlg = new Fl_Window(300, 105, "Replace");
Fl_Input *replace_find = new Fl_Input(70, 10, 200, 25, "Find:");
Fl_Input *replace_with = new Fl_Input(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");

7.7 Callbacks

Now that we've defined the GUI components of our editor, we need to define our callback functions.

Generated by Doxygen
7.7.1 changed_cb()

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

```c
void changed_cb(int, int nInserted, int nDeleted, int, const char*, void* v) { 
  if ((nInserted || nDeleted) && !loading) 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.

7.7.2 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);
}
```

7.7.3 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);
}
```

7.7.4 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();
}
```

7.7.5 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);
  }
```
7.7 Callbacks

7.7.6 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*, void*, v) {
  EditorWindow* e = (EditorWindow*)v;
  if (e->search[0] == '\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... */
    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.

7.7.7 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();
}
```

7.7.8 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.

7.7.9 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;
  if (!check_save()) return;
  Fl_Text_Editor::kf_paste(0, e->editor);
}
```
### 7.7.10 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);
}
```

### 7.7.11 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();
}
```

### 7.7.12 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);
}
```

### 7.7.13 replall_cb()

This callback will replace all occurrences of the search string in the file:
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);
    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));
            e->editor->show_insert_position();
            times++;
        }
    }
    if (times) fl_message("Replaced %d occurrences.", times);
    else fl_alert("No occurrences of \"%s\" found!", find);
}

7.7.14 replcan_cb()

This callback just hides the replace dialog:

void replcan_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    e->replace_dlg->hide();
}

7.7.15 save_cb()

This callback saves the current file. If the current filename is blank it calls the "save as" callback:

void save_cb(void) {
    if (filename[0] == '\0') {
        // No filename - get one!
        saveas_cb();
        return;
    } else save_file(filename);
}

The save_file() function saves the current file to the specified filename.

7.7.16 saveas_cb()

This callback asks the user for a filename and saves the current file:

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.
7.8  Other Functions

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

7.8.1  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;
}
```

7.8.2  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;
    int r;
    if (!insert) strcpy(filename, "");
    int 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.

7.8.3  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();
}
```
7.9 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();
    if (argc > 1) load_file(argv[1], -1);
    return Fl::run();
}
```

7.10 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!
### 7.11 The Final Product

The final editor window should look like the image in Figure 7.2.

![Figure 7.2 The completed editor window](image)

```c
// "Id: editor.cxx,v 1.2.2.3.2.5 2001/12/09 12:52:13
// A simple text editor program for the Fast Light Tool
// This program is described in Chapter 4 of the FLTK
// Copyright 1996-2001 by Bill Spitzak and others.
// This library is free software; you can redistribute
// modify it under the terms of the GNU Library General
// License as published by the Free Software Foundation.
// This library is distributed in the hope that it will
// but WITHOUT ANY WARRANTY; without even the implied
// MERCHANTABILITY or FITNESS FOR ANY PURPOSE.
// Library General Public License for more details.
// You should have received a copy of the GNU Library
// License along with this library; if not, write to the
// Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA.
```

### 7.12 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.

#### 7.12.1 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 [https://sourceforge.net/projects/nedit/](https://sourceforge.net/projects/nedit/), which uses a parallel "style" buffer which tracks the font, color, and size of the text that is drawn.

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

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

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

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

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:

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:

//
// 'style_update()' - Update the style buffer...
//
void
style_update(int pos, // I - Position of update
int nInserted, // I - Number of inserted chars
int nDeleted, // I - Number of deleted chars
int nRestyled, // I - Number of restyled chars
const char *deletedText, // I - Text that was deleted
void *cbArg) { // I - Callback data

int start, // Start of text
end; // End of text
char last, // Last style on line
*style, // 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
46

Designing a Simple Text Editor

// 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);
if (last != style[end - start - 1]) {
// The last character on the line changed styles, so reparse the
// remainder of the buffer...
free(text);
free(style);
end
= textbuf->length();
text = textbuf->text_range(start, end);
style = stylebuf->text_range(start, end);
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:

//
// ’style_parse()’ - Parse text and produce style data.
//
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) {
// Quoted quote...
*style++ = current;
*style++ = current;
text ++;
length --;
col += 2;
continue;
} else if (*text == ’\"’) {
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;

Generated by Doxygen


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 == '\n') {
        // End quote...
        *style++ = current;
        col ++;
        current = 'A';
        continue;
    }
}
// Copy style info...
if (current == 'A' && (*text == '{' || *text == '}')) *style++ = 'G';
else *style++ = current;
col ++;
last = isalnum(*text) || *text == '.';
if (*text == '\n') {
    // Reset column and possibly reset the style
    col = 0;
    if (current == 'B' || current == 'S') current = 'A';
}
}
Chapter 8

Drawing Things in FLTK

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

8.1 When Can You Draw Things in FLTK?

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

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

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

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

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

1. Make your surface the new current drawing surface calling the `Fl_Surface_Device::push_current(Fl_Surface_Device*)` function.

2. Make a series of calls to any of the drawing functions described below; these will operate on the new current drawing surface;

3. Set the current drawing surface back to its previous state calling `Fl_Surface_Device::pop_current()`.
8.1.1 What Drawing Unit do FLTK drawing functions use?

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

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

FLTK uses several units to measure graphical elements:

- All data visible by the public API (e.g., window widths, line lengths, font sizes, clipping regions) are in FLTK units which are both system- and DPI-independent.
- Just before drawing to a screen, the library internally multiplies all quantities expressed in FLTK units by the current value of the scale factor for the screen in use and obtains quantities in drawing units. The current scale factor value, for an Fl_Window named \( \text{window} \), is given by

\[
\text{int nscreen = window->screen_num();} // the screen where window is mapped \\
\text{float s = Fl::screen_scale(nscreen);} // this screen's scale factor
\]

One drawing unit generally corresponds to one screen pixel...
- ...but not on Mac OS X and for retina displays, where one drawing unit corresponds to two pixels.

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

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

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

When drawing to \( \text{Fl\_Printer} \) or \( \text{Fl\_PostScript\_File\_Device} \), the drawing unit is initially one point, that is, 1/72 of an inch. This unit is changed by calls to \( \text{Fl\_Paged\_Device::scale()} \).
8.2 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

8.2.1 Boxes

FLTK provides three functions that can be used to draw boxes for buttons and other UI controls. Each function uses the supplied upper-lefthand 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.
8.2.2 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 FLTK units and is unaffected by the current transformation matrix.

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

```c
void fl_push_clip(int x, int y, int w, int h)
void fl_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.
8.3 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:

![Figure 8.1 FLTK default colormap (Fl_Color 0x00 - 0xff)](image)

Color values greater than 255 are treated as 24-bit RGB values. These are mapped to the closest color supported by the screen, either from one of the 256 colors in the FLTK 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()}
\texttt{Fl::foreground()}
\texttt{Fl::background()}
\texttt{Fl::background2()}

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

\texttt{Fl::own\_ colormap()}
\texttt{Fl::free\_color(Fl\_Color i, int overlay)}
\texttt{Fl::set\_color(Fl\_Color i, unsigned c)}

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

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

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

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

// COLOR MAP INDEX
color(0x0000000111)
        ------- |
                | |
                | 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.

8.3.1 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 FLTK units thick to draw the lines. Zero results in the system-defined default, which on both X and Windows is somewhat different and nicer than 1.

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

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

8.3.2 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).

void fl_point(int x, int y)

Draw a single pixel at the given coordinates.

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

Color a rectangle that exactly fills the given bounding box.

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

Color a rectangle with "exactly" the passed r, g, b color. On screens with less than 24 bits of color this is done by drawing a solid-colored block using fl_draw_image() so that the correct color shade is produced.

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)
8.3 Colors

Draw a 1-pixel border *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 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.

```c
void fl_scroll(int X, int Y, int W, int H, int dx, int dy, void (draw_area)(void, int, int, int, int), void* data)
```

Scroll a rectangle and draw the newly exposed portions. The contents of the rectangular area is first shifted by dx and dy FLTK units. The callback is then called for every newly exposed rectangular area.

### 8.3.3 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.
double fl_transform_x(double x, double y)
double fl_transform_y(double x, double y)
double fl_transform_dx(double x, double y)
double fl_transform_dy(double x, double y)
void fl_transformed_vertex(double xf, double yf)

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

void fl_begin_points()
void fl_end_points()

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

void fl_begin_line()
void fl_end_line()

Start and end drawing lines.

void fl_begin_loop()
void fl_end_loop()

Start and end drawing a closed sequence of lines.

void fl_begin_polygon()
void fl_end_polygon()

Start and end drawing a convex filled polygon.

void fl_begin_complex_polygon()
void fl_gap()
void fl_end_complex_polygon()

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

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

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

Add a single vertex to the current path.

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

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

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.

![Figure 8.3 fl_arc(x,y,r,a1,a2)](image)

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

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

**Note:** fl_circle() draws incorrectly if the transformation is both rotated and non-square scaled.
### 8.3.4 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` bytes starting at the given location. In both cases, the text must be UTF-8 encoded. Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract `fl_descent()` from `y`. To align to the top, subtract `fl_descent()` and add `fl_height()`. This version of `fl_draw()` provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

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

Draw a UTF-8 string of length `n` bytes right to left starting at the given `x`, `y` location.

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

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

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

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

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

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

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

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

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

```c
int fl_height()
```

Recommended minimum line spacing for the current font. You can also just use the value of `size` passed to `fl_font()`.

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 width of a null-terminated string, a sequence of `n` characters, or a single character in the current font.

See also

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

```c
void fl_text_extents(const char *txt, int& dx, int& dy, int& w, int& h)
```

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

Given a string "txt" drawn using `fl_draw(txt, x, y)` you would determine its extents in FLTK units on the display using `fl_text_extents(txt, dx, dy, w, h)` such that a bounding box that exactly fits around the inked area of the text could be drawn with `fl_rect(x+dx, y+dy, w, h)`.

Refer to the full documentation for `fl_text_extents()` for details on usage.

See also

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

```c
const char* fl_shortcut_label(int shortcut)
```

Unparse a shortcut value as used by `Fl_Button` or `Fl_Menu_Item` into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero an empty string is returned. The return value points at a static buffer that is overwritten with each call.
8.3.5 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: FL_HELVETICA, FL_TIMES, FL_COURIER, and modifier values FL_BOLD and FL_ITALIC which can be added to these, and FL_SYMBOL and FL_ZAPF_DINGBATS. Faces greater than 255 cannot be used in Fl_Widget labels, since Fl_Widget stores the index as a byte.

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

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

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

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

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

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

8.3.6 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.

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

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

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

---

Generated by Doxygen
8.3.7 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.

```cpp
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.

8.4 Drawing Images

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

8.4.1 Direct Image Drawing

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

```cpp
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:

```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.

```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*H` 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.

```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.
8.4.2 Direct Image Reading

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

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

Read a RGB(A) image from the current window or off-screen buffer. The \( p \) argument points to a buffer that can hold the image and must be at least \( W \times H \times 3 \) 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.

8.4.3 Image Classes

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

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

- \texttt{Fl_Bitmap}
- \texttt{Fl_Pixmap}
- \texttt{Fl_RGB_Image}

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

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

The \texttt{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 \texttt{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()

\texttt{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:

- \texttt{Fl_GIF_Image}
- \texttt{Fl_JPEG_image}
Each of these image classes loads a named file of the corresponding format. The Fl_Shared_Image class can be used to load any type of image file - the class examines the file and constructs an image of the appropriate type.

Finally, FLTK provides a special image class called Fl_Tiled_Image to tile another image object in the specified area. This class can be used to tile a background image in a Fl_Group widget, for example.

virtual 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(Fl_RGB_Scaling method)

virtual void Fl_Image::draw(int x, int y, int w, int h, int ox, int oy)

The draw() method draws the image object. x, y, w, h indicates the destination rectangle. ox, oy, w, h is the source rectangle. This source rectangle is copied to the destination. The source rectangle may extend outside the image, i.e. ox and oy may be negative and w and h may be bigger than the image, and this area is left unchanged.

Note
See exceptions for Fl_Tiled_Image::draw() regarding arguments ox, oy, w, and h.

virtual void Fl_Image::draw(int x, int y)

Draws the image with the upper-left corner at x, y. This is the same as doing img->draw(x, y, img->w(), img->h(), 0, 0) where img is a pointer to any Fl_Image type.
8.4.4 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/fl_draw.H>`.

**Fl_Double_Window** uses offscreen rendering to avoid flickering on systems that don't support double-buffering natively.

**Fl_Offscreen fl_create_offscreen(int w, int h)**

Create an RGB offscreen buffer containing as many pixels as in a screen area of size \(w, h\) FLTK units.

**void fl_delete_offscreen(Fl_Offscreen)**

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

**void fl_begin_offscreen(Fl_Offscreen)**

Send all subsequent drawing commands to this offscreen buffer. FLTK can draw into a buffer at any time. There is no need to wait for an Fl_Widget::draw() to occur.

**void fl_end_offscreen()**

Quit sending drawing commands to this offscreen buffer.

**void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen osrc, int srcx, int srcy)**

Copy a rectangular area of the size \(w \times h\) from \(srcx, srcy\) in the offscreen buffer into the current drawing surface at \(x, y\).

**void fl_rescale_offscreen(Fl_Offscreen &osrc)**

Adapts the offscreen's size in pixels to a changed value of the scale factor while keeping the offscreen's graphical content.
Chapter 9

Handling Events

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

9.1 The FLTK Event Model

Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application. Events can also come from other programs like the window manager.

Events are identified by the integer argument passed to a `handle()` method that overrides the `Fl_Widget::handle()` virtual method. Other information about the most recent event is stored in static locations and acquired by calling the `Fl::event_∗()` methods. This static information remains valid until the next event is read from the window system, so it is ok to look at it outside of the `handle()` method.

Event numbers can be converted to their actual names using the `fl_eventnames[]` array defined in `#include <FL/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.

9.2 Mouse Events

9.2.1 FL_PUSH

A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling `Fl::event_button()`. You find out the mouse position by calling `Fl::event_x()` and `Fl::event_y()`.

A widget indicates that it "wants" the mouse click by returning non-zero from its `handle()` method, as in the `MyClass::handle()` example. It will then become the `Fl::pushed()` widget and will get `FL_DRAG` and the matching `FL_RELEASE` events. If `handle()` returns zero then FLTK will try sending the `FL_PUSH` to another widget.

9.2.2 FL_DRAG

The mouse has moved with a button held down. The current button state is in `Fl::event_state()`. The mouse position is in `Fl::event_x()` and `Fl::event_y()`.

In order to receive `FL_DRAG` events, the widget must return non-zero when handling `FL_PUSH`. 
9.2.3 FL_RELEASE

A mouse button has been released. You can find out what button by calling Fl::event_button().

In order to receive the FL_RELEASE event, the widget must return non-zero when handling FL_PUSH.

9.2.4 FL_MOVE

The mouse has moved without any mouse buttons held down. This event is sent to the Fl::belowmouse() widget.

In order to receive FL_MOVE events, the widget must return non-zero when handling FL_ENTER.

9.2.5 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.

9.3 Focus Events

9.3.1 FL_ENTER

The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.

9.3.2 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.

9.3.3 FL_FOCUS

This indicates an attempt to give a widget the keyboard focus.

If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its 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.

9.3.4 FL_UNFOCUS

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

9.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP

A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN, and both names are used interchangeably in this documentation.

The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length().

If you use the key, then handle() should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a FL_SHORTCUT event. FL_KEYBOARD events are also generated by the character palette/map.

To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events by returning 1. This way FLTK knows whether to bother sending your widget keyboard events. (Some widgets don’t need them, e.g. Fl_Box.)

If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into characters.

FL_KEYUP events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding FL_KEYDOWN event because focus may have changed between events.

Todo  Add details on how to detect repeating keys, since on some X servers a repeating key will generate both FL_KEYUP and FL_KEYDOWN, such that to tell if a key is held, you need Fl::event_key(int) to detect if the key is being held down during FL_KEYUP or not.

9.4.2 FL_SHORTCUT

If the Fl::focus() widget is zero or ignores an FL_KEYBOARD event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. FL_SHORTCUT is first sent to the Fl::belowmouse() widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes!

You can also make “global” shortcuts by using Fl::add_handler(). A global shortcut will work no matter what windows are displayed or which one has the focus.

9.5 Widget Events

9.5.1 FL_DEACTIVATE

This widget is no longer active, due to deactivate() being called on it or one of its parents. Please note that although active() may still return true for this widget after receiving this event, it is only truly active if active() is true for both it and all of its parents. (You can use active_r() to check this).

9.5.2 FL_ACTIVATE

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

This widget is no longer visible, due to hide() being called on it or one of its parents, or due to a parent window being minimized. Please note that although visible() may still return true for this widget after receiving this event, it is only truly visible if visible() is true for both it and all of its parents. (You can use visible_r() to check this).

9.5.4 FL_SHOW

This widget is visible again, due to show() being called on it or one of its parents, or due to a parent window being restored. A child Fl_Window will respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass FL_SHOW to the base class handle() method!

Note

The events in this chapter ("Widget Events"), i.e. FL_ACTIVATE, FL_DEACTIVATE, FL_SHOW, and FL_HIDE, are the only events deactivated and invisible widgets can usually get, depending on their states. Under certain circumstances, there may also be FL_LEAVE or FL_UNFOCUS events delivered to deactivated or hidden widgets.

9.6 Clipboard Events

9.6.1 FL_PASTE

You should get this event some time after you call Fl::paste(). The contents of Fl::event_text() is the text to insert and the number of characters is in Fl::event_length().

9.6.2 FL_SELECTIONCLEAR

The Fl::selection_owner() will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.

9.7 Drag and Drop Events

FLTK supports drag and drop of text and files from any application on the desktop to an FLTK widget. Text is transferred using UTF-8 encoding. 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.
9.7.1 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.

9.7.2 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.

9.7.3 FL_DND_LEAVE

The mouse has moved out of the widget.

9.7.4 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.

9.8 Other events

9.8.1 FL_SCREEN_CONFIGURATION_CHANGED

Sent whenever the screen configuration changes (a screen is added/removed, a screen resolution is changed, screens are moved). Use Fl::add_handler() to be notified of this event.

9.8.2 FL_FULLSCREEN

The application window has been changed from normal to fullscreen, or from fullscreen to normal. If you are using a X window manager which supports Extended Window Manager Hints, this event will not be delivered until the change has actually happened.
9.9 Fl::event_∗() methods

FLTK keeps the information about the most recent event in static storage. This information is good until the next event is processed. Thus it is valid inside handle() and callback() methods.

These are all trivial inline functions and thus very fast and small:

- Fl::event_button()
- Fl::event_clicks()
- Fl::event_dx()
- Fl::event_dy()
- Fl::event_inside()
- Fl::event_is_click()
- Fl::event_key()
- Fl::event_length()
- Fl::event_state()
- Fl::event_text()
- Fl::event_x()
- Fl::event_x_root()
- Fl::event_y()
- Fl::event_y_root()
- Fl::get_key()
- Fl::get_mouse()
- Fl::testShortcut()

9.10 Event Propagation

Widgets receive events via the virtual handle() function. The argument indicates the type of event that can be handled. The widget must indicate if it handled the event by returning 1. FLTK will then remove the event and wait for further events from the host. If the widget's handle function returns 0, FLTK may redistribute the event based on a few rules.

Most events are sent directly to the handle() method of the Fl_Window that the window system says they belong to. The window (actually the Fl_Group that Fl_Window is a subclass of) is responsible for sending the events on to any child widgets. To make the Fl_Group code somewhat easier, FLTK sends some events (FL_DRAG, FL_RELEASE, FL_KEYBOARD, FL_SHORTCUT, FL_UNFOCUS, and FL_LEAVE) directly to leaf widgets. These procedures control those leaf widgets:

- Fl::add_handler()
- Fl::belowmouse()
- Fl::focus()
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.

### 9.11 FLTK Compose-Character Sequences

The character composition done by FL_Input widget requires that you call the Fl::compose() function if you are writing your own text editor widget.

Currently, all characters made by single key strokes with or without modifier keys, or by system-defined character compose sequences (that can involve dead keys or a compose key) can be input. You should call Fl::compose() in case any enhancements to this processing are done in the future. The interface has been designed to handle arbitrary UTF-8 encoded text.

The following methods are provided for character composition:

- Fl::compose()
- Fl::compose_reset()

Under Mac OS X, FLTK “previews” partially composed sequences.
Chapter 10

Adding and Extending Widgets

This chapter describes how to add your own widgets or extend existing widgets in FLTK.

10.1 Subclassing

New widgets are created by subclassing an existing FLTK widget, typically Fl_Widget for controls and Fl_Group for composite widgets.

A control widget typically interacts with the user to receive and/or display a value of some sort.

A composite widget holds a list of child widgets and handles moving, sizing, showing, or hiding them as needed. Fl_Group is the main composite widget class in FLTK, and all of the other composite widgets (Fl_Pack, Fl_Scroll, Fl_Tabs, Fl_Tile, 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.

10.2 Making a Subclass of Fl_Widget

Your subclasses can directly descend from Fl_Widget or any subclass of Fl_Widget. Fl_Widget has only four virtual methods, and overriding some or all of these may be necessary.

10.3 The Constructor

The constructor should have the following arguments:

MyClass(int x, int y, int w, int h, const char *label = 0);

This will allow the class to be used in FLUID without problems.

The constructor must call the constructor for the base class and pass the same arguments:
Adding and Extending Widgets

```cpp
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:

```cpp
type(0);
box(FL_NO_BOX);
color(FL_BACKGROUND_COLOR);
selection_color(FL_BACKGROUND_COLOR);
labeltype(FL_NORMAL_LABEL);
labelstyle(FL_NORMAL_STYLE);
labelcolor(FL_FOREGROUND_COLOR);
align(FL_ALIGN_CENTER);
callback(default_callback, 0);
flags(ACTIVE|VISIBLE);
image(0);
deimage(0);
```

### 10.4 Protected Methods of Fl_Widget

The following methods are provided for subclasses to use:

- `clear_visible()`
- `damage()`
- `draw_box()`
- `draw_focus()`
- `draw_label()`
- `set_flag()`
- `set_visible()`
- `test_shortcut()`
- `type()`

```cpp
void Fl_Widget::damage(uchar mask)
void Fl_Widget::damage(uchar mask, int x, int y, int w, int h)
uchar Fl_Widget::damage()
```

The first form indicates that a partial update of the object is needed. The bits in mask are OR'd into `damage()`. Your `draw()` routine can examine these bits to limit what it is drawing. The public method `Fl_Widget::redraw()` simply does `Fl_Widget::damage(FL_DAMAGE_ALL)`, but the implementation of your widget can call the public `damage(n)`.

The second form indicates that a region is damaged. If only these calls are done in a window (no calls to `damage(n)`) then FLTK will clip to the union of all these calls before drawing anything. This can greatly speed up incremental displays. The mask bits are OR'd into `damage()` unless this is a `Fl_Window` widget.
The third form returns the bitwise-OR of all damage(n) calls done since the last draw().

When redrawing your widgets you should look at the damage bits to see what parts of your widget need redrawing. The handle() method can then set individual damage bits to limit the amount of drawing that needs to be done:

```c
MyClass::handle(int event) {
    ...
    if (change_to_part1) damage(1);
    if (change_to_part2) damage(2);
    if (change_to_part3) damage(4);
}
```

```c
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!

```c
void Fl_Widget::draw_box() const
void Fl_Widget::draw_box(Fl_Boxtype t, Fl_Color c) const
```

The first form draws this widget's box(), using the dimensions of the widget. The second form uses t as the box type and c as the color for the box.

```c
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.

```c
void Fl_Widget::draw_label() const
void Fl_Widget::draw_label(Fl_Boxtype t, int x, int y, int w, int h) const
void Fl_Widget::draw_label(int x, int y, int w, int h, Fl_Align align) const
```

The first form is the usual function for a draw() method to call to draw the widget's label. It does not draw the label if it is supposed to be outside the box (on the assumption that the enclosing group will draw those labels).

The second form uses the passed bounding box instead of the widget's bounding box. This is useful so "centered" labels are aligned with some feature, like a moving slider.

The third form draws the label anywhere. It acts as though FL_ALIGN_INSIDE has been forced on so the label will appear inside the passed bounding box. This is designed for parent groups to draw labels with.
void Fl_Widget::set_flag(int c)

Calling `set_flag(SHORTCUT_LABEL)` modifies the behavior of `draw_label()` so that `&` characters cause an underscore to be printed under the next letter.

void Fl_Widget::set_visible()
void Fl_Widget::clear_visible()

Fast inline versions of `Fl_Widget::hide()` and `Fl_Widget::show()`. These do not send the `FL_HIDE` and `FL_SHOW` events to the widget.

int Fl_Widget::testShortcut()
static int Fl_Widget::testShortcut(const char *s)

The first version tests `Fl_Widget::label()` against the current event (which should be a `FL_SHORTCUT` event). If the label contains a `&` character and the character after it matches the keypress, this returns true. This returns false if the `SHORTCUT_LABEL` flag is off, if the label is `NULL`, or does not have a `&` character in it, or if the keypress does not match the character.

The second version lets you do this test against an arbitrary string.

**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).
10.5 Handling Events

The virtual method Fl_Widget::handle(int event) is called to handle each event passed to the widget. It can:

- Change the state of the widget.
- Call Fl_Widget::redraw() if the widget needs to be redisplayed.
- Call Fl_Widget::damage(uchar c) if the widget needs a partial-update (assuming you provide support for this in your draw() method).
- Call Fl_Widget::do_callback() if a callback should be generated.
- Call Fl_Widget::handle() on child widgets.

Events are identified by the integer argument. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_*() methods. This information remains valid until another event is handled.

Here is a sample handle() method for a widget that acts as a pushbutton and also accepts the keystroke 'x' to cause the callback:

```c
int MyClass::handle(int event) {
  switch(event) {
  case FL_PUSH:
    highlight = 1;
    redraw();
    return 1;
  case FL_DRAG:
    int t = Fl::event_inside(this);
    if (t != highlight) {
      highlight = t;
      redraw();
    }
    return 1;
  case FL_RELEASE:
    if (highlight) {
      highlight = 0;
      redraw();
      do_callback();
      // never do anything after a callback, as the callback
      // may delete the widget!
    }
    return 1;
  case FL_SHORTCUT:
    if (Fl::event_key() == 'x') {
      do_callback();
      return 1;
    }
    return 0;
  default:
    return Fl_Widget::handle(event);
  }
}
```

You must return non-zero if your handle() method uses the event. If you return zero, the parent widget will try sending the event to another widget.

For debugging purposes, event numbers can be printed as their actual event names using the fl_eventnames[] array, e.g.:

```c
#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)"
  // ...
}
```

Generated by Doxygen
10.6 Drawing the Widget

The `draw()` virtual method is called when FLTK wants you to redraw your widget. It will be called if and only if `damage()` is non-zero, and `damage()` will be cleared to zero after it returns. The `draw()` method should be declared protected so that it can't be called from non-drawing code.

The `damage()` value contains the bitwise-OR of all the `damage(n)` calls to this widget since it was last drawn. This can be used for minimal update, by only redrawing the parts whose bits are set. FLTK will turn on the `FL_DAMAGE_ALL` bit if it thinks the entire widget must be redrawn, e.g. for an expose event.

Expose events (and the `damage(mask,x,y,w,h)` function described above) will cause `draw()` to be called with FLTK's clipping turned on. You can greatly speed up redrawing in some cases by testing `fl_not_clipped(x,y,w,h)` or `fl_clip_box()` and skipping invisible parts.

Besides the protected methods described above, FLTK provides a large number of basic drawing functions, which are described in the chapter Drawing Things in FLTK.

10.7 Resizing the Widget

The `resize(x,y,w,h)` method is called when the widget is being resized or moved. The arguments are the new position, width, and height. `x()`, `y()`, `w()`, and `h()` still remain the old size. You must call `resize()` on your base class with the same arguments to get the widget size to actually change.

This should not call `redraw()`, at least if only the `x()` and `y()` change. This is because composite widgets like `Fl_Scroll` may have a more efficient way of drawing the new position.

10.8 Making a Composite Widget

A "composite" widget contains one or more "child" widgets. To make a composite widget you should subclass `Fl_Group`. It is possible to make a composite object that is not a subclass of `Fl_Group`, but you'll have to duplicate the code in `Fl_Group` anyways.

Instances of the child widgets may be included in the parent:

```cpp
class MyClass : public Fl_Group {
    Fl_Button the_button;
    Fl_Slider the_slider;
    ...
};
```

The constructor has to initialize these instances. They are automatically added to the group, since the `Fl_Group` constructor does `Fl_Group::begin()`. Don't forget to call `Fl_Group::end()` or use the `Fl_End` pseudo-class:

```cpp
MyClass::MyClass(int x, int y, int w, int h) :
    Fl_Group(x, y, w, h),
    the_button(x + 5, y + 5, 100, 20),
    the_slider(x, y + 50, w, 20) {
    ...
    } // 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:
10.8 Making a Composite Widget

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_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()

void Fl_Group::draw_child(Fl_Widget &widget) const

This will force the child's damage() bits all to one and call draw() on it, then clear the damage(). You should call this on all children if a total redraw of your widget is requested, or if you draw something (like a background box) that damages the child. Nothing is done if the child is not visible() or if it is clipped.

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.

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).

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.
10.9 Cut and Paste Support

FLTK provides routines to cut and paste UTF-8 encoded text between applications:

- Fl::copy()
- Fl::paste()
- Fl::selection()
- Fl::selection_owner()

It is also possible to copy and paste image data between applications:

- Fl_Copy_Surface
- Fl::clipboard_contains()
- Fl::paste()

It may be possible to cut/paste other kinds of data by using Fl::add_handler(). Note that handling events beyond those provided by FLTK may be operating system specific. See Operating System Issues for more details.

10.10 Drag And Drop Support

FLTK provides routines to drag and drop UTF-8 encoded text between applications:

Drag'n'drop operations are initiated by copying data to the clipboard and calling the function Fl::dnd().

Drop attempts are handled via the following events, already described under Drag and Drop Events in a previous chapter:

- FL_DND_ENTER
- FL_DND_DRAG
- FL_DND_LEAVE
- FL_DND_RELEASE
- FL_PASTE

10.11 Making a subclass of Fl_Window

You may want your widget to be a subclass of Fl_Window, Fl_Double_Window, or Fl_Gl_Window. This can be useful if your widget wants to occupy an entire window, and can also be used to take advantage of system-provided clipping, or to work with a library that expects a system window ID to indicate where to draw.

Subclassing Fl_Window is almost exactly like subclassing Fl_Group, and in fact you can easily switch a subclass back and forth. Watch out for the following differences:

1. Fl_Window is a subclass of Fl_Group so make sure your constructor calls end() unless you actually want children added to your window.

2. When handling events and drawing, the upper-left corner is at 0,0, not x(), y() as in other Fl_Widget's. For instance, to draw a box around the widget, call draw_box(0, 0, w(), h()), rather than draw←box(x(), y(), w(), h()).

You may also want to subclass Fl_Window in order to get access to different visuals or to change other attributes of the windows. See the Operating System Issues chapter for more information.
Chapter 11

Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

11.1 Using OpenGL in FLTK

The easiest way to make an OpenGL display is to subclass Fl_Gl_Window. Your subclass must implement a `draw()` method which uses OpenGL calls to draw the display. Your main program should call `redraw()` when the display needs to change, and (somewhat later) FLTK will call `draw()`.

With a bit of care you can also use OpenGL to draw into normal FLTK windows (see Using OpenGL in Normal FLTK Windows below). This allows you to use Gouraud shading for drawing your widgets. To do this you use the `gl_start()` and `gl_finish()` functions around your OpenGL code.

You must include FLTK's `<FL/gl.h>` header file. It will include the file `<GL/gl.h>` (on macOS: `<OpenGL/include/GL/gl.h>`), define some extra drawing functions provided by FLTK, and include the `<windows.h>` header file needed by Windows applications.

Some simple coding rules (see OpenGL and support of HighDPI displays) allow to write cross-platform code that will support OpenGL run on HighDPI displays (including the 'retina' displays of Apple hardware).

11.2 Making a Subclass of Fl_Gl_Window

To make a subclass of Fl_Gl_Window, you must provide:

- A class definition.
- A `draw()` method.
- A `handle()` method if you need to receive input from the user.

If your subclass provides static controls in the window, they must be redrawn whenever the FL_DAMAGE_ALL bit is set in the value returned by `damage()`. For double-buffered windows you will need to surround the drawing code with the following code to make sure that both buffers are redrawn:

```c
#ifndef MESA
glDrawBuffer(GL_FRONT_AND_BACK);
#endif // !MESA

... draw stuff here ...

#ifndef MESA
glDrawBuffer(GL_BACK);
#endif // !MESA
```
11.2.1 Defining the Subclass

To define the subclass you just subclass the Fl_Gl_Window class:

```cpp
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.)

11.2.2 The draw() Method

The `draw()` method is where you actually do your OpenGL drawing:

```cpp
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 ...
}
```

11.2.3 The handle() Method

The `handle()` method handles mouse and keyboard events for the window:

```cpp
int MyWindow::handle(int event) {
  switch(event) {
    case FL_PUSH:
      ... mouse down event ...
      ... 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 0;
}
```
When \texttt{handle()} is called, the OpenGL context is not set up! If your display changes, you should call \texttt{redraw()} and let \texttt{draw()} do the work. Don't call any OpenGL drawing functions from inside \texttt{handle()}!

You can call some OpenGL stuff like hit detection and texture loading functions by doing:

\begin{verbatim}
    case FL_PUSH:
        make_current(); // make OpenGL context current
        if (!valid()) {
            ... set up projection exactly the same as draw ...
            valid(1); // stop it from doing this next time
        }
        ... ok to call NON-DRAWING OpenGL code here, such as hit
detection, loading textures, etc...
\end{verbatim}

Your main program can now create one of your windows by doing \texttt{new MyWindow(...)}.

You can also use your new window class in FLUID by:

1. Putting your class definition in a \texttt{MyWindow.H} file.
2. Creating a \texttt{Fl_Box} widget in FLUID.
3. In the widget panel fill in the "class" field with \texttt{MyWindow}. This will make FLUID produce constructors for your new class.
4. In the "Extra Code" field put \texttt{"#include "MyWindow.H"}, so that the FLUID output file will compile.

You must put \texttt{glwindow->show()} in your main code after calling \texttt{show()} on the window containing the OpenGL window.

11.3 OpenGL and support of HighDPI displays

HighDPI displays (including the so-called 'retina' displays of Apple hardware) are supported by FLTK in such a way that 1 unit of an FLTK quantity (say, the value given by \texttt{Fl_Gl_Window::w()}) corresponds to more than 1 pixel on the display. Conversely, when a program specifies the width and height of the OpenGL viewport, it is necessary to use an API that returns quantities expressed in pixels. That can be done as follows:

\begin{verbatim}
    Fl_Gl_Window *glw = ...;
    glViewport(0, 0, glw->pixel_w(), glw->pixel_h());
\end{verbatim}

which makes use of the \texttt{Fl_Gl_Window::pixel_w()} and \texttt{Fl_Gl_Window::pixel_h()} methods giving the size in pixels of an \texttt{Fl_Gl_Window} that is potentially mapped to a HighDPI display. Method \texttt{Fl_Gl_Window::pixels_per_unit()} can also be useful in this context.

\begin{itemize}
    \item A further coding rule is necessary to properly support retina displays and OpenGL under macOS (see OpenGL and 'retina' displays)
\end{itemize}
11.4 Using OpenGL in Normal FLTK Windows

Note

Drawing both with OpenGL and Quartz in a normal FLTK window is not possible with the macOS platform. This technique is therefore not useful under macOS because it permits nothing more than what is possible with class Fl_Gl_Window.

You can put OpenGL code into the `draw()` method, as described in Drawing the Widget in the previous chapter, or into the code for a boxtype or other places with some care.

Most importantly, before you show any windows, including those that don’t have OpenGL drawing, you must initialize FLTK so that it knows it is going to use OpenGL. You may use any of the symbols described for Fl_Gl_Window::mode() to describe how you intend to use OpenGL:

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

You can then put OpenGL drawing code anywhere you can draw normally by surrounding it with `gl_start()` and `gl_finish()` to set up, and later release, an OpenGL context with an orthographic projection so that 0,0 is the lower-left corner of the window and each pixel is one unit. The current clipping is reproduced with OpenGL `glScissor()` commands. These functions also synchronize the OpenGL graphics stream with the drawing done by other X, Windows, or FLTK functions.

```c
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`!
11.5 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>`.

`void gl_color(Fl_Color)`

Sets the current OpenGL color to a FLTK color. For color-index modes it will use `fl_xpixel(c)`, which is only right if this window uses the default colormap!

`void gl_rect(int x, int y, int w, int h)`

`void gl_rectf(int x, int y, int w, int h)`

Outlines or fills a rectangle with the current color. If `Fl_Gl_Window::ortho()` has been called, then the rectangle will exactly fill the pixel rectangle passed.

`void gl_font(Fl_Font fontid, int size)`

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

`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.

`void gl_draw(const char *s)`

`void gl_draw(const char *s, int n)`

`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)`

`void gl_draw(const char *s, int x, int y, int w, int h, Fl_Align)`

`void gl_draw(const char *s, int x, int y, int w, int h, Fl_Align)`

Draws a nul-terminated string or an array of `n` characters in the current OpenGL font at the current raster position.

`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.

`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 \( \wedge X \), and aligned with the edges or center. Exactly the same output as `fl_draw()`.
11.6 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.

11.7 Using OpenGL Optimizer with FLTK

OpenGL Optimizer is a scene graph toolkit for OpenGL available from Silicon Graphics for IRIX and Microsoft Windows. It allows you to view large scenes without writing a lot of OpenGL code.

OptimizerWindow Class Definition

to use OpenGL Optimizer with FLTK you'll need to create a subclass of Fl_Gl_Widget that includes several state variables:

class OptimizerWindow : public Fl_Gl_Widget {
    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_->setCamera(camera_);
            camera_->draw(draw_action_);
            redraw();
        }
    }
};
The camera() Method

The camera() method sets the camera (projection and viewpoint) to use when drawing the scene. The scene is redrawn after this call.

The draw() Method

The draw() method performs the needed initialization and does the actual drawing:

```cpp
void OptimizerWindow::draw() {
  if (!context_) {
    // This is the first time we've been asked to draw; create the
    // Optimizer context for the scene...
    #ifdef _WIN32
      context_ = new csContext((HDC)fl_getHDC());
      context_->ref();
      context_->makeCurrent((HDC)fl_getHDC());
    #else
      context_ = new csContext(fl_display, fl_visual);
      context_->ref();
      context_->makeCurrent(fl_display, fl_window);
    #endif // _WIN32
    ... perform other context setup as desired ...
    // Then create the draw action to handle drawing things...
    draw_action_ = new csDrawAction;
    if (camera_) { draw_action_->setCamera(camera_); camera_)->draw(draw_action_);
    } else {
      #ifdef _WIN32
        context_->makeCurrent((HDC)fl_getHDC());
      #else
        context_->makeCurrent(fl_display, fl_window);
      #endif // _WIN32
      if (!valid()) {
        // Update the viewport for this context...
        context_->setViewport(0, 0, w(), h());
      }
      // Clear the window...
      context_->clear(csContext::COLOR_CLEAR | csContext::DEPTH_CLEAR,
                      0.0f, // Red
                      0.0f, // Green
                      0.0f, // Blue
                      1.0f); // Alpha
      // Then draw the scene (if any)...
      if (scene_)
        draw_action_->apply(scene_);
    }
  }
}
```

The scene() Method

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.
11.8 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. FLTK recommends to use the GLEW library to perform this. It is therefore necessary to install the GLEW library (see below).

On the macOS platform, MacOS 10.7 or above is required; GLEW is possible but not necessary. FLTK creates contexts for OpenGL versions 1 and 2 without the FL_OPENGL3 flag and for OpenGL versions 3.2 and above with it.

GLEW installation (Unix/Linux and Windows platforms)

GLEW is available as a package for most Linux distributions and in source form at http://glew.sourceforge.net/. For the Windows platform, a Visual Studio static library (glew32.lib) can be downloaded from the same web site; a MinGW-style static library (libglew32.a) can be built from source with the make command.

Source-level changes for OpenGL 3:

- Put this in all OpenGL-using source files (instead of #include <FL/gl.h>, and before #include <FL/glut.h> if you use GLUT):
  ```c
  #if defined(__APPLE__)
  # include <OpenGL/gl3.h> // defines OpenGL 3.0+ functions
  #else
  # if defined(_WIN32)
  # define GLEW_STATIC 1
  # endif
  # include <GL/glew.h>
  #endif
  ```

- Add the FL_OPENGL3 flag when calling Fl_Gl_Window::mode(int a) or glutInitDisplayMode().

- Put this in the handle(int event) member function of the first to be created among your Fl_Gl_Window-derived classes:
  ```c
  #ifndef __APPLE__
  static int first = 1;
  if ((first && event == FL_SHOW && shown()) ||
    first == 0)
    make_current();
    glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  }
  #endif
  ```

- Alternatively, if you use GLUT, put
  ```c
  #ifdef __APPLE__
  glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  #endif
  ```

  after the first glutCreateWindow() call.

If GLEW is installed on the Mac OS development platform, it is possible to use the same code for all platforms, with one exception: put

```c
#ifdef __APPLE__
  glewExperimental = GL_TRUE;
#endif
``` 

before the glewInit() call.

Changes in the build process

Link with libGLEW.so (on Unix/Linux), libglew32.a (with MinGW) or glew32.lib (with MS Visual Studio); no change is needed on the Mac OS platform.
Chapter 12

Programming with FLUID

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

Subchapters:

- What is FLUID?
- Running FLUID Under UNIX
- Running FLUID Under Microsoft Windows
- Compiling .fl Files
- A Short Tutorial
- FLUID Reference
- FLUID Templates
- Internationalization with FLUID
- Known Limitations

12.1 What is FLUID?

The Fast Light User Interface Designer, or FLUID, is a graphical editor that is used to produce FLTK source code. FLUID edits and saves its state in .fl files. These files are text, and you can (with care) edit them in a text editor, perhaps to get some special effects.

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.
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.

12.2 Running FLUID Under UNIX

To run FLUID under UNIX, type:

```
fluid filename.fl &
```

to edit the .fl file filename.fl. If the file does not exist you will get an error pop-up, but if you dismiss it you will be editing a blank file of that name. You can run FLUID without any name, in which case you will be editing an unnamed blank setup (but you can use save-as to write it to a file).

You can provide any of the standard FLTK switches before the filename:

```
-display host:n.n
-geometry WxH+X+Y
-title window:title
-name classname
-ic
-fg color
-bg color
-bgi2 color
-scheme schemename
```

Changing the colors may be useful to see what your interface will look at if the user calls it with the same switches. Similarly, using "-scheme plastic" will show how the interface will look using the "plastic" scheme.

In the current version, if you don’t put FLUID into the background with '&', then you will be able to abort FLUID by typing CTRL-C on the terminal. It will exit immediately, losing any changes.
12.3 Running FLUID Under Microsoft Windows

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

12.4 Compiling .fl Files

FLUID can also be called as a command-line "compiler" to create the `.cxx` and `.h` file from a `.fl` file. To do this type:

```
fluid -c filename.fl
```

This is the same as the menu ‘File/Write Code...’. It will read the `filename.fl` file and write `filename.cxx` and `filename.h`. Any leading directory on `filename.fl` will be stripped, so they are always written to the current directory. If there are any errors reading or writing the files, FLUID will print the error and exit with a non-zero code. You can use the following lines in a makefile to automate the creation of the source and header files:

```
my_panels.h my_panels.cxx: my_panels.fl
   fluid -c my_panels.fl
```

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

```
.SUFFIXES: .fl .cxx .h
.fl.h .fl.cxx:
   fluid -c $<
```

If you use

```
fluid -cs filename.fl
```

FLUID will also write the "strings" for internationalization in file 'filename.txt' (menu: ‘File/Write Strings...’).

Finally there is another option which is useful for program developers who have many .fl files and want to upgrade them to the current FLUID version. FLUID will read the `filename.fl` file, save it, and exit immediately. This writes the file with current syntax and options and the current FLTK version in the header of the file. Use

```
fluid -u filename.fl
```

to 'upgrade' `filename.fl`. You may combine this with `-c` or `-cs`.

Note

All these commands overwrite existing files w/o warning. You should particularly take care when running ‘fluid -u’ since this overwrites the original .fl source file.
12.5 A Short Tutorial

FLUID is an amazingly powerful little program. However, this power comes at a price as it is not always obvious how to accomplish seemingly simple tasks with it. This tutorial will show you how to generate a complete user interface class with FLUID that is used for the CubeView program provided with FLTK.

The window is of class CubeViewUI, and is completely generated by FLUID, including class member functions. The central display of the cube is a separate subclass of Fl_Gl_Window called CubeView. CubeViewUI manages CubeView using callbacks from the various sliders and rollers to manipulate the viewing angle and zoom of CubeView.

At the completion of this tutorial you will (hopefully) understand how to:

1. Use FLUID to create a complete user interface class, including constructor and any member functions necessary.
2. Use FLUID to set callback member functions of custom widget classes.
3. Subclass an Fl_Gl_Window to suit your purposes.

12.5.1 The CubeView Class

The CubeView class is a subclass of Fl_Gl_Window. It has methods for setting the zoom, the $x$ and $y$ pan, and the rotation angle about the $x$ and $y$ axes.

You can safely skip this section as long as you realize that CubeView is a subclass of Fl_Gl_Window and will respond to calls from CubeViewUI, generated by FLUID.
The CubeView Class Definition

Here is the CubeView class definition, as given by its header file "test/CubeView.h":

```cpp
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.
    void 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;};
    // 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 drawCube();
    float vAng,hAng; float xshift,yshift;
    float boxv0[3];float boxv1[3]; float boxv2[3];float boxv3[3];
    float boxv4[3];float boxv5[3]; float boxv6[3];float boxv7[3];
};
```

The CubeView Class Implementation

Here is the CubeView implementation. It is very similar to the "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;
}
```
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);
    glEnd();

glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex3fv(boxv0);
    glVertex3fv(boxv1);
    glVertex3fv(boxv1);
    glVertex3fv(boxv2);
    glVertex3fv(box2);
    glVertex3fv(box3);
    glVertex3fv(box3);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv5);
    glVertex3fv(boxv5);
    glVertex3fv(boxv6);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glVertex3fv(boxv4);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv1);
    glVertex3fv(boxv5);
    glVertex3fv(boxv2);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glEnd();
**12.5 A Short Tutorial**

```cpp
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();
}
```

### 12.5.2 The CubeViewUI Class

We will completely construct a window to display and control the CubeView defined in the previous section using FLUID.

**Defining the CubeViewUI Class**

Once you have started FLUID, the first step in defining a class is to create a new class within FLUID using the **New->Code->Class** menu item. Name the class "CubeViewUI" and leave the subclass blank. We do not need any inheritance for this window. You should see the new class declaration in the FLUID browser window.

![Figure 12.3 FLUID file for CubeView](image)

**Adding the Class Constructor**

Generated by Doxygen
Click on the CubeViewUI class in the FLUID window and add a new method by selecting `New->Code->Function/Method`. The name of the function will also be CubeViewUI. FLUID will understand that this will be the constructor for the class and will generate the appropriate code. Make sure you declare the constructor `public`.

Then add a window to the CubeViewUI class. Highlight the name of the constructor in the FLUID browser window and click on `New->Group->Window`. In a similar manner add the following to the CubeViewUI constructor:

- A horizontal roller named `hrot`
- A vertical roller named `vrot`
- A horizontal slider named `xpan`
- A vertical slider named `ypan`
- A horizontal value slider named `zoom`

None of these additions need be public. And they shouldn’t be unless you plan to expose them as part of the interface for CubeViewUI.

When you are finished you should have something like this:

We will talk about the `show()` method that is highlighted shortly.

Adding the CubeView Widget

What we have is nice, but does little to show our cube. We have already defined the CubeView class and we would like to show it within the CubeViewUI.

The CubeView class inherits the `Fl_Gl_Window` class, which is created in the same way as an `Fl_Box` widget. Use `New->Other->Box` to add a square box to the main window. This will be no ordinary box, however.

The Box properties window will appear. The key to letting CubeViewUI display CubeView is to enter CubeView in the `Class` text entry box. This tells FLUID that it is not an `Fl_Box`, but a similar widget with the same constructor.

In the `Extra Code: field` enter `#include "CubeView.h"`
This `#include` is important, as we have just included CubeView as a member of CubeViewUI, so any public CubeView methods are now available to CubeViewUI.

![Figure 12.5 CubeView methods](image)

**Figure 12.5 CubeView methods**

### Defining the Callbacks

Each of the widgets we defined before adding CubeView can have callbacks that call CubeView methods. You can call an external function or put a short amount of code in the **Callback** field of the widget panel. For example, the callback for the `ypan` slider is:

```c
cube->pany(((Fl_Slider *)o)->value());
cube->redraw();
```

We call `cube->redraw()` after changing the value to update the CubeView window. CubeView could easily be modified to do this, but it is nice to keep this exposed. In the case where you may want to do more than one view change only redrawing once saves a lot of time.

There is no reason to wait until after you have added CubeView to enter these callbacks. FLUID assumes you are smart enough not to refer to members or functions that don’t exist.

### Adding a Class Method

You can add class methods within FLUID that have nothing to do with the GUI. As an example add a show function so that CubeViewUI can actually appear on the screen.

Make sure the top level CubeViewUI is selected and select **New->Code->Function/Method**. Just use the name `show()`. We don’t need a return value here, and since we will not be adding any widgets to this method FLUID will assign it a return type of `void`.

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

12.5.3 Adding Constructor Initialization Code

If you need to add code to initialize a class, for example setting initial values of the horizontal and vertical angles in the CubeView, you can simply highlight the constructor and select New->Code->Code. Add any required code.

12.5.4 Generating the Code

Now that we have completely defined the CubeViewUI, we have to generate the code. There is one last trick to ensure this all works. Open the preferences dialog from Edit->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.

12.6 FLUID Reference

The following sections describe each of the windows in FLUID.

12.6.1 The Widget Browser

The main window shows a menu bar and a scrolling browser of all the defined widgets. The name of the .fl file being edited is shown in the window title.

The widgets are stored in a hierarchy. You can open and close a level by clicking the "triangle" at the left of a widget. The leftmost widgets are the parents, and all the widgets listed below them are their children. Parents don't have to have any children.
The top level of the hierarchy is composed of functions and classes. Each of these will produce a single C++ public function or class in the output .cxx file. Calling the function or instantiating the class will create all of the child widgets.

The second level of the hierarchy contains the windows. Each of these produces an instance of class Fl_Window.

Below that are either widgets (subclasses of Fl_Widget) or groups of widgets (including other groups). Plain groups are for layout, navigation, and resize purposes. Tab groups provide the well-known file-card tab interface.

Widgets are shown in the browser by either their name (such as "main_panel" in the example), or by their type and label (such as "Button "the green").

You select widgets by clicking on their names, which highlights them (you can also select widgets from any displayed window). You can select many widgets by dragging the mouse across them, or by using Shift+Click to toggle them on and off. To select no widgets, click in the blank area under the last widget. Note that hidden children may be selected even when there is no visual indication of this.

You open widgets by double-clicking on them, or (to open several widgets you have picked) by typing the F1 key. A control panel will appear so you can change the widget(s).

### 12.6.2 Menu Items

The menu bar at the top is duplicated as a pop-up menu on any displayed window. The shortcuts for all the menu items work in any window. The menu items are:

- File/Open... (Ctrl+o)

  Discards the current editing session and reads in a different .fl file. You are asked for confirmation if you have changed the current file.

  FLUID can also read .fd files produced by the Forms and XForms "fdesign" programs. It is best to File/Merge them instead of opening them. FLUID does not understand everything in a .fd file, and will print a warning message on the controlling terminal for all data it does not understand. You will probably need to edit the resulting setup to fix these errors. Be careful not to save the file without changing the name, as FLUID will write over the .fd file with its own format, which fdesign cannot read!

- File/Insert... (Ctrl+i)

  Inserts the contents of another .fl file, without changing the name of the current .fl file. All the functions (even if they have the same names as the current ones) are added, and you will have to use cut/paste to put the widgets where you want.

- File/Save (Ctrl+s)

  Writes the current data to the .fl file. If the file is unnamed then FLUID will ask for a filename.
File/Save As... (Ctrl+Shift+S)

Asks for a new filename and saves the file.

File/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+v)

Pastes the widgets from the clipboard file.

If the widget is a window, it is added to whatever function is selected, or contained in the current selection.

If the widget is a normal widget, it is added to whatever window or group is selected. If none is, it is added to the window or group that is the parent of the current selection.

To avoid confusion, it is best to select exactly one widget before doing a paste.

Cut/paste is the only way to change the parent of a widget.

Edit/Select All (Ctrl+a)

Selects all widgets in the same group as the current selection.

If they are all selected already then this selects all widgets in that group's parent. Repeatedly typing Ctrl+a will select larger and larger groups of widgets until everything is selected.

Edit/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)**

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.
Edit/GUI Settings... (Shift+Alt+p)

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

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](image)

**Figure 12.9 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 an `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.

Layout/Align/...

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

Layout/Space Evenly/...

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

Layout/Make Same Size/...

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

Layout/Center in Group/...

Center all selected widgets relative to their parent widget

Layout/Grid and Size Settings... (Ctrl+g)

Displays the grid settings panel.

This panel controls 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.
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.
12.7 GUI Attributes

12.6.3 The Widget Panel

When you double-click on a widget or a set of widgets you will get the "widget attribute panel".

When you change attributes using this panel, the changes are reflected immediately in the window. It is useful to hit the "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.

![](image)

Figure 12.11 The FLUID widget GUI attributes

12.7 GUI Attributes

Not all fields in the Widget attributes dialog will be visible for all types of widgets.

Label (text field)
String to print next to or inside the button. You can put newlines into the string to make multiple lines. The easiest way is by typing Ctrl+j.

*Symbols* can be added to the label using the at sign ("@").

Label (pull down menu)

How to draw the label. Normal, shadowed, engraved, and embossed change the appearance of the text.

Image

The active image for the widget. Click on the Browse... button to pick an image file using the file chooser.

Inactive

The inactive image for the widget. Click on the Browse... button to pick an image file using the file chooser.

Alignment (buttons)

Where to draw the label. The arrows put it on that side of the widget, you can combine them to put it in the corner. The "box" button puts the label inside the widget, rather than outside.

The *clip* button clips the label to the widget box, the *wrap* button wraps any text in the label, and the *text image* button puts the text over the image instead of under the image.

Position (text fields)

The position fields show the current position and size of the widget box. Enter new values to move and/or resize a widget.

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.
12.7.1 Style Attributes

Label Font (pulldown menu)

Font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. Your program can change the actual font used by these “slots” in case you want some font other than the 16 provided.

Label Size (pulldown menu)

Pixel size (height) for the font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. To see the result without dismissing the panel, type the new number and then Tab.

Label Color (button)

Color to draw the label. Ignored by pixmaps (bitmaps, however, do use this color as the foreground color).

Box (pulldown menu)
The boxtype to draw as a background for the widget.

Many widgets will work, and draw faster, with a "frame" instead of a "box". A frame does not draw the colored interior, leaving whatever was already there visible. Be careful, as FLUID may draw this ok but the real program may leave unwanted stuff inside the widget.

If a window is filled with child widgets, you can speed up redrawing by changing the window's box type to "← NO_BOX". FLUID will display a checkerboard for any areas that are not colored in by boxes. Note that this checkerboard is not drawn by the resulting program. Instead random garbage will be displayed.

**Down Box (pulldown menu)**

The boxtype to draw when a button is pressed or for some parts of other widgets like scrollbars and valuators.

**Color (button)**

The color to draw the box with.

**Select Color (button)**

Some widgets will use this color for certain parts. FLUID does not always show the result of this: this is the color buttons draw in when pushed down, and the color of input fields when they have the focus.

**Text Font, Size, and Color**

Some widgets display text, such as input fields, pull-down menus, and browsers.
12.7.2 C++ Attributes

**Class**

This is how you use your own subclasses of `Fl_Widget`. Whatever identifier you type in here will be the class that is instantiated.

In addition, no `#include` header file is put in the `.h` file. You must provide a `#include` line as the first line of the "Extra Code" which declares your subclass.

The class must be similar to the class you are spoofing. It does not have to be a subclass. It is sometimes useful to change this to another FLTK class. For windows you can select either `Single` or `Double` in the drop-down box right to the "Class:" field to get a normal window (`Fl_Window`) or a double-buffered window (`Fl_Double_Window`), respectively.

**Type (upper-right pulldown menu)**

Some classes have subtypes that modify their appearance or behavior. You pick the subtype off of this menu.

**Name (text field)**
Name of a variable to declare, and to store a pointer to this widget into. This variable will be of type "]class >". If the name is blank then no variable is created.

You can name several widgets with "name[0]", "name[1]", "name[2]", etc. This will cause FLUID to declare an array of pointers. The array is big enough that the highest number found can be stored. All widgets in the array must be the same type.

**Public (button)**

Controls whether the widget is publicly accessible. When embedding widgets in a C++ class, this controls whether the widget is public or private in the class. Otherwise it controls whether the widget is declared static or global (extern).

**Extra Code (text fields)**

These four fields let you type in literal lines of code to dump into the .h or .cxx files.

If the text starts with a # or the word extern then FLUID thinks this is an "include" line, and it is written to the .h file. If the same include line occurs several times then only one copy is written.

All other lines are "code" lines. The current widget is pointed to by the local variable o. The window being constructed is pointed to by the local variable w. You can also access any arguments passed to the function here, and any named widgets that are before this one.

FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler. If you need more than four lines you probably should call a function in your own .cxx code.

**Callback (text field)**

This can either be the name of a function, or a small snippet of code. If you enter anything other than letters, numbers, and the underscore then FLUID treats it as code.

A name refers to a function in your own code. It must be declared as void name(<class>*, void*).

A code snippet is inserted into a static function in the .cxx output file. The function prototype is void name(class *o, void *v) so that you can refer to the widget as o and the user_data() as v. FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler.
If the callback is blank then no callback is set.

User Data (text field)

This is a value for the user_data() of the widget. If blank the default value of zero is used. This can be any piece of C code that can be cast to a void pointer.

Type (text field)

The void* in the callback function prototypes is replaced with this. You may want to use long for old XForms code. Be warned that anything other than void* is not guaranteed to work! However on most architectures other pointer types are ok, and long is usually ok, too.

When (pulldown menu)

When to do the callback. This can be Never, Changed, Release, or Enter Key. The value of Enter Key is only useful for text input fields.

There are other rare but useful values for the when() field that are not in the menu. You should use the extra code fields to put these values in.

No Change (button)

The No Change button means the callback is done on the matching event even if the data is not changed.

12.8 Selecting and Moving Widgets

Double-clicking a window name in the browser will display it, if not displayed yet. From this display you can select widgets, sets of widgets, and move or resize them. To close a window either double-click it or type ESC.

To select a widget, click it. To select several widgets drag a rectangle around them. Holding down shift will toggle the selection of the widgets instead.

You cannot pick hidden widgets. You also cannot choose some widgets if they are completely overlapped by later widgets. Use the browser to select these widgets.

The selected widgets are shown with a red "overlay" line around them. You can move the widgets by dragging this box. Or you can resize them by dragging the outer edges and corners. Hold down the Alt key while dragging the mouse to defeat the snap-to-grid effect for fine positioning.
If there is a tab box displayed you can change which child is visible by clicking on the file tabs. The child you pick is selected.

The arrow, tab, and shift+tab keys "navigate" the selection. Left, right, tab, or shift+tab move to the next or previous widgets in the hierarchy. Hit the right arrow enough and you will select every widget in the window. Up/down widgets move to the previous/next widgets that overlap horizontally. If the navigation does not seem to work you probably need to "Sort" the widgets. This is important if you have input fields, as FLTK uses the same rules when using arrow keys to move between input fields.

To "open" a widget, double click it. To open several widgets select them and then type F1 or pick "Edit/Open" off the pop-up menu.

Type Ctrl+o to temporarily toggle the overlay off without changing the selection, so you can see the widget borders.

You can resize the window by using the window manager border controls. FLTK will attempt to round the window size to the nearest multiple of the grid size and makes it big enough to contain all the widgets (it does this using illegal X methods, so it is possible it will barf with some window managers!). Notice that the actual window in your program may not be resizable, and if it is, the effect on child widgets may be different.

The panel for the window (which you get by double-clicking it) is almost identical to the panel for any other Fl_Widget. There are three extra items:

### 12.9 Image Labels

The contents of the image files in the Image and Inactive text fields are written to the .cxx file. If many widgets share the same image then only one copy is written. Since the image data is embedded in the generated source code, you need only distribute the C++ code and not the image files themselves.

However, the filenames are stored in the .fl file so you will need the image files as well to read the .fl file. Filenames are relative to the location of the .fl file and not necessarily the current directory. We recommend you either put the images in the same directory as the .fl file, or use absolute path names.

Notes for All Image Types

FLUID runs using the default visual of your X server. This may be 8 bits, which will give you dithered images. You may get better results in your actual program by adding the code "Fl::visual(FL_RGB)" to your code right before the first window is displayed.

All widgets with the same image on them share the same code and source X pixmap. Thus once you have put an image on a widget, it is nearly free to put the same image on many other widgets.

If you edit an image at the same time you are using it in FLUID, the only way to convince FLUID to read the image file again is to remove the image from all widgets that are using it or re-load the .fl file.

Don't rely on how FLTK crops images that are outside the widget, as this may change in future versions! The cropping of inside labels will probably be unchanged.
To more accurately place images, make a new "box" widget and put the image in that as the label.

XBM (X Bitmap) Files

FLUID reads X bitmap files which use C source code to define a bitmap. Sometimes they are stored with the ".h" or ".bm" extension rather than the standard ".xbm" extension.

FLUID writes code to construct an Fl_Bitmap image and use it to label the widget. The '1' bits in the bitmap are drawn using the label color of the widget. You can change this color in the FLUID widget attributes panel. The '0' bits are transparent.

The program "bitmap" on the X distribution does an adequate job of editing bitmaps.

XPM (X Pixmap) Files

FLUID reads X pixmap files as used by the libxpm library. These files use C source code to define a pixmap. The filenames usually have the ".xpm" extension.

FLUID writes code to construct an Fl_Pixmap image and use it to label the widget. The label color of the widget is ignored, even for 2-color images that could be a bitmap. XPM files can mark a single color as being transparent, and FLTK uses this information to generate a transparency mask for the image.

We have not found any good editors for small iconic pictures. For pixmaps we have used XPaint and the KDE icon editor.

BMP Files

FLUID reads Windows BMP image files which are often used in Windows applications for icons. FLUID converts BMP files into (modified) XPM format and uses an Fl_BMP_Image image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .bmp file.

GIF Files

FLUID reads GIF image files which are often used in HTML documents to make icons. FLUID converts GIF files into (modified) XPM format and uses an Fl_GIF_Image image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .gif file. Only the first image of an animated GIF file is used.
12.10 FLUID Templates

JPEG Files

If FLTK is compiled with JPEG support, FLUID can read JPEG image files which are often used for digital photos. FLUID uses an Fl_JPEG_Image image to label the widget, and writes uncompressed RGB or grayscale data to the source file.

PNG (Portable Network Graphics) Files

If FLTK is compiled with PNG support, FLUID can read PNG image files which are often used in HTML documents. FLUID uses a Fl_PNG_Image image to label the widget, and writes uncompressed RGB or grayscale data to the source file. PNG images can provide a full alpha channel for partial transparency, and FLTK supports this as best as possible on each platform.

12.10 FLUID Templates

Fluid can store a number of project templates. Project templates are great for storing often used boilerplate code for fast access. A common use would be projects with readily prepared copyright messages.

A sample template for FLTK projects is included with Fluid.

Choose "File > New From Template..." to create a new project based on a template file. In the template dialog, select one of the existing templates. All occurrences of the word "@INSTANCE@" in the template are replaced with the text in the "Instance" field. To create the new project click "New".

To add your current project as a new template, choose "File > Save As Template...", fill in a name, and click "Save".

To delete a template, open the template dialog using "New from Template" or "Save As Template", then select any existing template, and click "Delete Template".

12.11 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.
12.11.1 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.

12.11.2 Using GNU gettext for I18N

FLUID's code support for GNU gettext is limited to calling a function or macro to retrieve the localized label; you still need to call setlocale() and textdomain() or bindtextdomain() to select the appropriate language and message file.

To use GNU gettext for I18N, open the preferences window and choose "GNU gettext" from the Use: chooser. 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 12.14 Internationalization using GNU gettext](image-url)

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.
12.11.3 Using POSIX catgets for I18N

FLUID's code support for POSIX catgets allows you to use a global message file for all interfaces or a file specific to each .fl file; you still need to call `setlocale()` to select the appropriate language.

To use POSIX catgets for I18N, open the preferences window and choose "POSIX catgets" from the Use: chooser. Three new input fields will then appear to control the include file, catalog file, and set number for retrieving the localized label strings.

![Preferences Window]

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.

12.12 Known Limitations

Declaration Blocks can be used to temporarily block out already designed code using #if 0 and #endif type construction. This will effectively avoid compilation of blocks of code. However, static code and data generated by this segment (menu items, images, include statements, etc.) will still be generated and likely cause compile-time warnings.
Chapter 13

Advanced FLTK

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

13.1 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.

13.2 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.
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.

### 13.3 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;
```
13.3 Simple multithreaded examples using Fl::lock

You can start as many threads as you like. From within a thread (other than the main() thread) FLTK calls must be wrapped with calls to Fl::lock() and Fl::unlock():

```c
void my_thread(void) {
    while (thread_still_running) {
        /* do thread work */
        ...
        /* compute new values for widgets */
        ...
        Fl::lock(); // acquire the lock
        my_widget->update(values);
        Fl::unlock(); // release the lock; allow other threads to access FLTK again
        Fl::awake(); // use Fl::awake() to signal main thread to refresh the GUI
    }
}
```

Note

To trigger a refresh of the GUI from a worker thread, the worker code should call Fl::awake()

Using Fl::awake thread messages

You can send messages from worker threads to the main() thread using Fl::awake(void* message). If using this thread message interface, your main() might look like this:

```c
int main(int argc, char **argv) {
    /* Create your windows and widgets here */
    Fl::lock(); /* "start" the FLTK lock mechanism */
    /* show your window */
    main_win->show(argc, argv);
    /* start your worker threads */
    ... start threads ...
    /* Run the FLTK loop and process thread messages */
    while (Fl::wait() > 0) {
        if ((next_message = Fl::thread_message()) != NULL) {
            /* process your data, update widgets, etc. */
            ...
        }
    }
    /* terminate any pending worker threads */
    ... stop threads ...
    return 0;
}
```

Your worker threads can send messages to the main() thread using Fl::awake(void* message):

```c
void *msg; // "msg" is a pointer to your message
Fl::awake(msg); // send "msg" to main thread
```

A message can be anything you like. The main() thread can retrieve the message by calling Fl::thread_←message().

Generated by Doxygen
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
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 use or update `userdata`.

**Warning**

The mechanisms used to deliver `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` events to the `main()` thread can interact in unexpected ways on some platforms. Therefore, for reliable operation, it is advised that a program use either `Fl::awake(Fl_Awake_Handler cb, void* userdata)` or `Fl::awake(void* message)`, but that they never be intermixed. Calling `Fl::awake()` with no parameters should be safe in either case.

If you have to choose between using the `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` mechanisms and don't know which to choose, then try the `Fl::awake(Fl_Awake_Handler cb, void* userdata)` method first as it tends to be more powerful in general.

### 13.4 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?
13.5 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 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().
Chapter 14

Unicode and UTF-8 Support

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

Unicode support was added to FLTK starting with version 1.3.0 and is still incomplete but mostly functional. This chapter is Work in Progress, reflecting the current state of Unicode support.

14.1 About Unicode, ISO 10646 and UTF-8

The summary of Unicode, ISO 10646 and UTF-8 given below is deliberately brief and provides just enough information for the rest of this chapter.

For further information, please see:

- https://unicode.org
- https://iso.org
- https://www.cl.cam.ac.uk/~mgk25/unicode.html

The Unicode Standard

The Unicode Standard was originally developed by a consortium of mainly US computer manufacturers and developers of multi-lingual software. It has now become a de facto standard for character encoding and is supported by most of the major computing companies in the world.

Before Unicode, many different systems, on different platforms, had been developed for encoding characters for different languages, but no single encoding could satisfy all languages. Unicode provides access to over 130,000 characters used in all the major languages written today, and is independent of platform and language.

Unicode also provides higher-level concepts needed for text processing and typographic publishing systems, such as algorithms for sorting and comparing text, composite character and text rendering, right-to-left and bi-directional text handling.
Note

There are currently no plans to add this extra functionality to FLTK.

ISO 10646

The International Organisation for Standardization (ISO) had also been trying to develop a single unified character set. Although both ISO and the Unicode Consortium continue to publish their own standards, they have agreed to coordinate their work so that specific versions of the Unicode and ISO 10646 standards are compatible with each other.

The international standard ISO 10646 defines the Universal Character Set (UCS) which contains the characters required for almost all known languages. The standard also defines three different implementation levels specifying how these characters can be combined.

Note

There are currently no plans for handling the different implementation levels or the combining characters in FLTK.

In UCS, characters have a unique numerical code and an official name, and are usually shown using 'U+' and the code in hexadecimal, e.g. U+0041 is the "Latin capital letter A". The UCS characters U+0000 to U+007F correspond to US-ASCII, and U+0000 to U+00FF correspond to ISO 8859-1 (Latin1).

ISO 10646 was originally designed to handle a 31-bit character set from U+00000000 to U+7FFFFFFF, but the current idea is that 21 bits will be sufficient for all future needs, giving characters up to U+10FFFF. The complete character set is sub-divided into planes. Plane 0, also known as the Basic Multilingual Plane (BMP), ranges from U+0000 to U+FFFF and consists of the most commonly used characters from previous encoding standards. Other planes contain characters for specialist applications.

Todo

FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus (Fl_Shortcut) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.

The UCS also defines various methods of encoding characters as a sequence of bytes. UCS-2 encodes Unicode characters into two bytes, which is wasteful if you are only dealing with ASCII or Latin1 text, and insufficient if you need characters above U+00FFFF. UCS-4 uses four bytes, which lets it handle higher characters, but this is even more wasteful for ASCII or Latin1.

UTF-8

The Unicode standard defines various UCS Transformation Formats (UTF). UTF-16 and UTF-32 are based on units of two and four bytes. UCS characters requiring more than 16 bits are encoded using "surrogate pairs" in UTF-16.

UTF-8 encodes all Unicode characters into variable length sequences of bytes. Unicode characters in the 7-bit ASCII range map to the same value and are represented as a single byte, making the transformation to Unicode quick and easy.

All UCS characters above U+007F are encoded as a sequence of several bytes. The top bits of the first byte are set to show the length of the byte sequence, and subsequent bytes are always in the range 0x80 to 0xBF. This combination provides some level of synchronisation and error detection.
14.2 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().

---

<table>
<thead>
<tr>
<th>Unicode range</th>
<th>Byte sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+00000000 – U+0000007F</td>
<td>0xxxxxxxx</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+03FFFFFF</td>
<td>111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+04000000 – U+7FFFFFFF</td>
<td>11111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
</tbody>
</table>

Note

This table contains theoretical values outside the valid Unicode range (U+000000 – U+10FFFF). Such values can only be returned by conversion functions for illegal input values (see Illegal Unicode and UTF-8 Sequences).

Moving from ASCII encoding to Unicode will allow all new FLTK applications to be easily internationalized and used all over the world. By choosing UTF-8 encoding, FLTK remains largely source-code compatible to previous iterations of the library.
14.3 Illegal Unicode and UTF-8 Sequences

Three pre-processor variables are defined in the source code [1] that determine how fl_utf8decode() handles illegal UTF-8 sequences:

- if ERRORS_TO_CP1252 is set to 1 (the default), fl_utf8decode() will assume that a byte sequence starting with a byte in the range 0x80 to 0x9f represents a Microsoft CP1252 character, and will return the value of an equivalent UCS character. Otherwise, it will be processed as an illegal byte value as described below.

- if STRICT_RFC3629 is set to 1 (not the default!) then UTF-8 sequences that correspond to illegal UCS values are treated as errors. Illegal UCS values include those above U+10FFFF, or corresponding to UTF-16 surrogate pairs. Illegal byte values are handled as described below.

- if ERRORS_TO_ISO8859_1 is set to 1 (the default), the illegal byte value is returned unchanged, otherwise 0xFFFD, the Unicode REPLACEMENT CHARACTER, is returned instead.

[1] Since FLTK 1.3.4 you may set these three pre-processor variables on your compile command line with -D"variable=value" (value: 0 or 1) to avoid editing the source code.

fl_utf8encode() is less strict, and only generates the UTF-8 sequence for 0xFFFD, the Unicode REPLACEMENT CHARACTER, if it is asked to encode a UCS value above U+10FFFF.

Many of the [ftlk2] 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.

14.4 FLTK Unicode and UTF-8 Functions

This section provides a brief overview of the functions. For more details, consult the main text for each function via its link.

int fl_utf8locale() **FLTK2**

fl_utf8locale() returns true if the "locale" seems to indicate that UTF-8 encoding is used.

*It is highly recommended that you change your system so this does return true!*
int fl_utf8test(const char *src, unsigned len) FLTK2

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.

int fl_unichar_to_utf8_size(Fl_Unichar)
int fl_utf8bytes(unsigned ucs)

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

int fl_utf8len(char c) OksiD

If c is a valid first byte of a UTF-8 encoded character sequence, fl_utf8len() will return the number of bytes in that sequence. It returns -1 if c is not a valid first byte.

unsigned int fl_nonspacing(unsigned int ucs) OksiD

Returns true if ucs is a non-spacing character.

const char* fl_utf8back(const char *p, const char *start, const char *end) FLTK2
const char* fl_utf8fwd(const char *p, const char *start, const char *end) FLTK2

If p already points to the start of a UTF-8 character sequence, these functions will return p. Otherwise fl_utf8back() searches backwards from p and fl_utf8fwd() searches forwards from p, within the start and end limits, looking for the start of a UTF-8 character.

unsigned int fl_utf8decode(const char *p, const char *end, int *len) FLTK2
int fl_utf8encode(unsigned ucs, char *buf) FLTK2

fl_utf8decode() attempts to decode the UTF-8 character that starts at p and may not extend past end. It returns the Unicode value, and the length of the UTF-8 character sequence is returned via the len argument. fl_utf8encode() writes the UTF-8 encoding of ucs into buf and returns the number of bytes in the sequence. See the main documentation for the treatment of illegal Unicode and UTF-8 sequences.
unsigned int fl_utf8froma(char *dst, unsigned dstlen, const char *src, unsigned srclen) FLTK2
unsigned int fl_utf8toa(const char *src, unsigned srclen, char *dst, unsigned dstlen) FLTK2

fl_utf8froma() converts a character string containing single bytes per character (i.e. ASCII or ISO-8859-1) into UTF-8. If the src string contains only ASCII characters, the return value will be the same as srclen.

fl_utf8toa() converts a string containing UTF-8 characters into single byte characters. UTF-8 characters that do not correspond to ASCII or ISO-8859-1 characters below 0xFF are replaced with '?'.

Both functions return the number of bytes that would be written, not counting the null terminator. dstlen provides a means of limiting the number of bytes written, so setting dstlen to zero is a means of measuring how much storage would be needed before doing the real conversion.

char* fl_utf2mbcs(const char *src) OksiD

converts a UTF-8 string to a local multi-byte character string. [More info required here!]

unsigned int fl_utf8fromwc(char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen) FLTK2
unsigned int fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen) FLTK2
unsigned int fl_utf8toUtf16(const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen) FLTK2

These routines convert between UTF-8 and wchar_t or "wide character" strings. The difficulty lies in the fact that sizeof(wchar_t) is 2 on Windows and 4 on Linux and most other systems. Therefore some "wide characters" on Windows may be represented as "surrogate pairs" of more than one wchar_t.

fl_utf8fromwc() converts from a "wide character" string to UTF-8. Note that srclen is the number of wchar_t elements in the source string and on Windows this might be larger than the number of characters. dstlen specifies the maximum number of bytes to copy, including the null terminator.

fl_utf8towc() converts a UTF-8 string into a "wide character" string. Note that on Windows, some "wide characters" might result in "surrogate pairs" and therefore the return value might be more than the number of characters. dstlen specifies the maximum number of wchar_t elements to copy, including a zero terminating element. [Is this all worded correctly?]

fl_utf8toUtf16() converts a UTF-8 string into a "wide character" string using UTF-16 encoding to handle the "surrogate pairs" on Windows. dstlen specifies the maximum number of wchar_t elements to copy, including a zero terminating element. [Is this all worded correctly?]

These routines all return the number of elements that would be required for a full conversion of the 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.
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. `fl_utf_strncasecmp()` only compares the first n characters [bytes?]

### 14.5 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

TODO:

• more doc on unicode, add links
• write something about filename encoding on OS X...
• explain the fl_utf8___ commands
• explain issues with Fl_Preferences
Chapter 15

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.

15.1 Version Numbers

The FLTK version number is stored in a number of compile-time constants:

- FL_MAJOR_VERSION - The major release number, currently 1
- FL_MINOR_VERSION - The minor release number, currently 4
- FL_PATCH_VERSION - The patch release number, currently 0
- FL_VERSION - [Deprecated] A combined floating-point version number for the major, minor, and patch release numbers, currently 1.0400
- FL_API_VERSION - A combined integer version number for the major, minor, and patch release numbers, currently 10400 (use this instead of FL_VERSION, if possible)
- FL_ABI_VERSION - A combined integer version number for the application binary interface (ABI) major, minor, and patch release numbers, currently 10400 (default)

Note
The ABI version (FL_ABI_VERSION) is usually constant throughout one major/minor release version, for instance 10300 if FL_API_VERSION is 10304. Hence the ABI is constant if only the patch version is changed. You can change this with configure or CMake though if you want the latest enhancements (called "ABI features", see CHANGES).
15.2 Events

Events are identified by an `Fl_Event` enumeration value. The following events are currently defined:

- `FL_NO_EVENT` - No event (or an event fltk does not understand) occurred (0).
- `FL_PUSH` - A mouse button was pushed.
- `FL_RELEASE` - A mouse button was released.
- `FL_ENTER` - The mouse pointer entered a widget.
- `FL_LEAVE` - The mouse pointer left a widget.
- `FL_DRAG` - The mouse pointer was moved with a button pressed.
- `FL_FOCUS` - A widget should receive keyboard focus.
- `FL_UNFOCUS` - A widget loses keyboard focus.
- `FL_KEYBOARD` - A key was pressed.
- `FL_CLOSE` - A window was closed.
- `FL_MOVE` - The mouse pointer was moved with no buttons pressed.
- `FL_SHORTCUT` - The user pressed a shortcut key.
- `FL_DEACTIVATE` - The widget has been deactivated.
- `FL_ACTIVATE` - The widget has been activated.
- `FL_HIDE` - The widget has been hidden.
- `FL_SHOW` - The widget has been shown.
- `FL_PASTE` - The widget should paste the contents of the clipboard.
- `FL_SELECTIONCLEAR` - The widget should clear any selections made for the clipboard.
- `FL_MOUSEWHEEL` - The horizontal or vertical mousewheel was turned.
- `FL_DND_ENTER` - The mouse pointer entered a widget dragging data.
- `FL_DND_DRAG` - The mouse pointer was moved dragging data.
- `FL_DND_LEAVE` - The mouse pointer left a widget still dragging data.
- `FL_DND_RELEASE` - Dragged data is about to be dropped.
- `FL_SCREEN_CONFIGURATION_CHANGED` - The screen configuration (number, positions) was changed.
- `FL_FULLSCREEN` - The fullscreen state of the window has changed.
15.3 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.

15.4 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

15.5 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.
FLTK Enumerations

- 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.

15.6 Fl::event_state() Values

The following constants define bits in the Fl::event_state() value:

- FL_SHIFT - One of the shift keys is down.
- FL_CAPS_LOCK - The caps lock is on.
- FL_CTRL - One of the ctrl keys is down.
- FL_ALT - One of the alt keys is down.
- FL_NUM_LOCK - The num lock is on.
- FL_META - One of the meta/Windows keys is down.
- FL_COMMAND - An alias for FL_CTRL on Windows and X11, or FL_META on MacOS X.
- FL_CONTROL - An alias for FL_META on Windows and X11, or FL_CTRL on MacOS X.
- FL_SCROLL_LOCK - The scroll lock is on.
- FL_BUTTON1 - Mouse button 1 is pushed.
- FL_BUTTON2 - Mouse button 2 is pushed.
- FL_BUTTON3 - Mouse button 3 is pushed.
- FL_BUTTONS - Any mouse button is pushed.
- FL_BUTTON(n) - Mouse button n (where n > 0) is pushed.
15.7 Alignment Values

The following constants define bits that can be used with `Fl_Widget::align()` to control the positioning of the label:

- `FL_ALIGN_CENTER` - The label is centered (0).
- `FL_ALIGN_TOP` - The label is top-aligned.
- `FL_ALIGN_BOTTOM` - The label is bottom-aligned.
- `FL_ALIGN_LEFT` - The label is left-aligned.
- `FL_ALIGN_RIGHT` - The label is right-aligned.
- `FL_ALIGN_CLIP` - The label is clipped to the widget.
- `FL_ALIGN_WRAP` - The label text is wrapped as needed.
- `FL_ALIGN_TOP_LEFT` - The label appears at the top of the widget, aligned to the left.
- `FL_ALIGN_TOP_RIGHT` - The label appears at the top of the widget, aligned to the right.
- `FL_ALIGN_BOTTOM_LEFT` - The label appears at the bottom of the widget, aligned to the left.
- `FL_ALIGN_BOTTOM_RIGHT` - The label appears at the bottom of the widget, aligned to the right.
- `FL_ALIGN_LEFT_TOP` - The label appears to the left of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_RIGHT_TOP` - The label appears to the right of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_LEFT_BOTTOM` - The label appears to the left of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_RIGHT_BOTTOM` - The label appears to the right of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_INSIDE` - 'or' this with other values to put label inside the widget.
- `FL_ALIGN_TEXT_OVER_IMAGE` - Label text will appear above the image.
- `FL_ALIGN_IMAGE_OVER_TEXT` - Label text will be below the image.
- `FL_ALIGN_IMAGE_NEXT_TO_TEXT` - The image will appear to the left of the text.
- `FL_ALIGN_TEXT_NEXT_TO_IMAGE` - The image will appear to the right of the text.
- `FL_ALIGN_IMAGE_BACKDROP` - The image will be used as a background for the widget.

15.8 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.

15.9 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.

15.9.1 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
• 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.

15.10 Cursors

The following constants define the mouse cursors that are available in FLTK. The double-headed arrows are bitmaps provided by FLTK on X, the others are provided by system-defined cursors.

• FL_CURSOR_DEFAULT - the default cursor, usually an arrow (0)
• FL_CURSOR_ARROW - an arrow pointer
• FL_CURSOR_CROSS - crosshair
• FL_CURSOR_WAIT - watch or hourglass
• FL_CURSOR_INSERT - I-beam
• FL_CURSOR_HAND - hand (uparrow on Windows)
• FL_CURSOR_HELP - question mark
• FL_CURSOR_MOVE - 4-pointed arrow
• FL_CURSOR_NS - up/down arrow
• FL_CURSOR_WE - left/right arrow
• FL_CURSOR_NWSE - diagonal arrow
• FL_CURSOR_NESW - diagonal arrow
• FL_CURSOR_NONE - invisible
15.11 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.

15.12 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.
Chapter 16

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.

16.1 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.

16.2 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 key, int x, int y))`
• 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").

16.3 Mixing GLUT and FLTK Code

You can make your GLUT window a child of a Fl_Window with the following scheme. The biggest trick is that GLUT insists on a call to show() the window at the point it is created, which means the Fl_Window parent window must already be shown.

• Don’t call glutInit().
• Create your Fl_Window, and any FLTK widgets. Leave a blank area in the window for your GLUT window.
• show() the Fl_Window. Perhaps call show(argc, argv).
• Call window->begin() so that the GLUT window will be automatically added to it.
• Use glutInitWindowSize() and glutInitWindowPosition() to set the location in the parent window to put the GLUT window.
• Put your GLUT code next. It probably does not need many changes. Call window->end() immediately after the glutCreateWindow()!
• You can call either glutMainLoop(), Fl::run(), or loop calling Fl::wait() to run the program.
16.4 class Fl_Glut_Window

16.4.1 Class Hierarchy

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

16.4.2 Include Files

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

16.4.3 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().
```

16.4.4 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>
16.4.5 Methods

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.

virtual Fl_Glut_Window::~Fl_Glut_Window()

Destroys the GLUT window.

void Fl_Glut_Window::make_current()

Switches all drawing functions to the GLUT window.
Chapter 17

Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

**Warning:** The Forms compatibility is deprecated and no longer maintained in FLTK1, and is likely to be removed completely after the next official release.

### 17.1 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>`.

### 17.2 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.
17.3 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_atactivate() fl_set_form_atdeactivate()
- fl_set_form_property()
- fl_set_app_mainform(), fl_get_app_mainform()
- fl_set_form_minsize(), fl_set_form_maxsize()
- fl_set_form_event_cmask(), fl_get_form_event_cmask()
- fl_set_form_dblbuffer(), fl_set_object_dblbuffer() (use an Fl_Double_Window instead)
- fl_adjust_form_size()
- fl_register_raw_callback()
- fl_set_object_bw(), fl_set_border_width()
- fl_set_object_resize(), fl_set_object_gravity()
- fl_set_object_shortcutkey()
17.4 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 \texttt{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 \texttt{Fl\_GL\_Window} and write a \texttt{draw()} method and \texttt{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 \texttt{draw\_overlay()} method and call \texttt{redraw\_overlay()} on the OpenGL window.

One easy way to hack your program so it works is to make the \texttt{draw()} and \texttt{handle()} methods on your window set some static variables, storing what event happened. Then in the main loop of your program, call \texttt{Fl\::wait()} and then check these variables, acting on them as though they are events read from \texttt{fl\_queue}.

You Must Use OpenGL to Draw Everything

The file \texttt{<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 \texttt{fl\_make\_object} or directly setting the handle routine will not compile. You have to rewrite them to use a subclass of \texttt{Fl\_Widget}. It is important to note that the \texttt{handle()} method is not exactly the same as the \texttt{handle()} function of Forms. Where a Forms \texttt{handle()} returned non-zero, your \texttt{handle()} must call \texttt{do\_callback()}. And your \texttt{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 \texttt{<device.h>}

If you have written your own "free" widgets you will probably get a lot of errors about "getvaluator". You should substitute:

<table>
<thead>
<tr>
<th>Forms</th>
<th>FLTK</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE_X</td>
<td>Fl::event_x_root()</td>
</tr>
<tr>
<td>MOUSE_Y</td>
<td>Fl::event_y_root()</td>
</tr>
<tr>
<td>LEFTSHIFTKEY_RIGHTSHIFTKEY</td>
<td>Fl::event_shift()</td>
</tr>
<tr>
<td>CAPSLOCKKEY</td>
<td>Fl::event_capslock()</td>
</tr>
<tr>
<td>LEFTCTRLKEY_RIGHTCTRLKEY</td>
<td>Fl::event_ctrl()</td>
</tr>
<tr>
<td>LEFTALTKEY_RIGHTALTKEY</td>
<td>Fl::event_alt()</td>
</tr>
<tr>
<td>MOUSE1_RIGHTMOUSE</td>
<td>Fl::event_state()</td>
</tr>
<tr>
<td>MOUSE2_MIDDLEMOUSE</td>
<td>Fl::event_state()</td>
</tr>
<tr>
<td>MOUSE3_LEFTMOUSE</td>
<td>Fl::event_state()</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Anything else in `getvaluator` and you are on your own...

Font Numbers Are Different

The “style” numbers have been changed because I wanted to insert bold-italic versions of the normal fonts. If you use Times, Courier, or Bookman to display any text you will get a different font out of FLTK. If you are really desperate to fix this use the following code:

```cpp
fl_font_name(3, "*courier-medium-r-no*");
fl_font_name(4, "*courier-bold-r-no*");
fl_font_name(5, "*courier-medium-o-no*");
fl_font_name(6, "*times-medium-r-no*");
fl_font_name(7, "*times-bold-r-no*");
fl_font_name(8, "*times-medium-i-no*");
fl_font_name(9, "*bookman-light-r-no*");
fl_font_name(10, "*bookman-demi-r-no*");
fl_font_name(11, "*bookman-light-i-no*");
```
Chapter 18

Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

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

18.1 Accessing the OS Interfaces

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

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

This header file will define the appropriate interface for your environment. The pages that follow describe the functionality that is provided for each operating system.

Note

These definitions used to be in FL/x.H up to FLTK 1.3.x. Usage of FL/x.H is deprecated since FLTK 1.4.0. You should replace all references of FL/x.H with FL/platform.H if your target is FLTK 1.4 or later. FL/x.H will be retained for backwards compatibility for some releases but will be removed in a later (not yet specified) FLTK release.

WARNING:
The interfaces provided by this header file may change radically in new FLTK releases. Use them only when an existing generic FLTK interface is not sufficient.
18.2 The UNIX (X11) Interface

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

18.2.1 Handling Other X Events

void Fl::add_handler(int (∗f)(int))

Installs a function to parse unrecognized events. If FLTK cannot figure out what to do with an event, it calls each of these functions (most recent first) until one of them returns non-zero. If none of them returns non-zero then the event is ignored.

FLTK calls this for any X events it does not recognize, or X events with a window ID that FLTK does not recognize. You can look at the X event in the fl_xevent variable.

The argument is the FLTK event type that was not handled, or zero for unrecognized X events. These handlers are also called for global shortcuts and some other events that the widget they were passed to did not handle, for example FL_SHORTCUT.

extern XEvent ∗fl_xevent

This variable contains the most recent X event.

extern ulong fl_event_time

This variable contains the time stamp from the most recent X event that reported it; not all events do. Many X calls like cut and paste need this value.

Window fl_xid(const Fl_Window ∗)

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

Fl_Window ∗fl_find(ulong xid)

Returns the Fl_Window that corresponds to the given XID, or NULL if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.

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.
18.2 The UNIX (X11) Interface

18.2.2 Drawing using Xlib

The following global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current():

```c
extern Display *fl_display;
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:

```c
XDrawSomething(fl_display, fl_window, fl_gc, ...);
```

Other information such as the position or size of the X window can be found by looking at Fl_Window::current(), which returns a pointer to the Fl_Window being drawn.

```c
unsigned long fl_xpixel(Fl_Color i)
unsigned long fl_xpixel(uchar r, uchar g, uchar b)
```

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

```c
int fl_parse_color(const char ∗p, uchar & r, uchar & g, uchar & b)
```

Convert a name into the red, green, and blue values of a color by parsing the X11 color names. On other systems, fl_parse_color() can only convert names in hexadecimal encoding, for example #ff8083.

```c
extern XFontStruct ∗fl_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.

```c
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*.
18.2.3 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.

```
int Fl::display(const char ∗)
```

Set which X display to use. This actually does putenv("DISPLAY=...") so that child programs will display on the same screen if called with exec(). This must be done before the display is opened. This call is provided under MacOS and Windows but it has no effect.

```
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.

```
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.

```
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.

```
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.#".

```
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:

```
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);
```
18.2.4 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:

virtual void Fl_Window::show()

If the window is already shown() this must cause it to be raised, this can usually be done by calling Fl_Window::show(). If not shown() your implementation must call either Fl_X::set_xid() or Fl_X::make_xid().

An example:

```c
void MyWindow::show() {
  if (shown()) {Fl_Window::show(); return;} // you must do this!
  fl_open_display(); // necessary if this is first window
  // we only calculate the necessary visual colormap once:
  static XVisualInfo *visual;    static Colormap colormap;
  if (!visual) {visual = figure_out_visual();
    colormap = XCreateColormap(fl_display, RootWindow(fl_display, fl_screen), vis->visual, AllocNone);
  }
  Fl_X::make_xid(this, visual, colormap);
}
```

Fl_X *Fl_X::set_xid(Fl_Window*, Window xid)

Allocate a hidden class called an Fl_X, put the XID into it, and set a pointer to it from the Fl_Window. This causes Fl_Window::shown() to return true.

void Fl_X::make_xid(Fl_Window*, XVisualInfo* = fl_visual, Colormap = fl_colormap)

This static method does the most onerous parts of creating an X window, including setting the label, resize limitations, etc. It then does Fl_X::set_xid() with this new window and maps the window.

virtual void Fl_Window::flush()

This virtual function is called by Fl::flush() to update the window. For FLTK's own windows it does this by setting the global variables fl_window and fl_gc and then calling the draw() method. For your own windows you might just want to put all the drawing code in here.

The X region that is a combination of all damage() calls done so far is in Fl_X::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_damageExpose` 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);
        mypixmap = 0;
    }
    Fl_Window::hide(); // you must call this
}
```

virtual void Fl_Window::~Fl_Window()

Because of the way C++ works, if you override `hide()` you must override the destructor as well (otherwise only the base class `hide()` is called):

```c
MyWindow::~MyWindow() {
    hide();
}
```

Note

Access to the Fl_X hidden class requires to `#define FL_INTERNALS` before compilation.

### 18.2.5 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.

### 18.2.6 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`).
18.2.7 Display Scaling Factor

FLTK uses the value of the Xft.dpi resource divided by 96. to initialize the display scaling factor. That is also what is done by the gnome and KDE desktops.

18.3 The Windows Interface

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

18.3.1 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.

18.3.2 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.

18.3.3 Handling Other Windows API 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 window classes. The easiest way is to call Fl::wait(), as it will call DispatchMessage() for all messages to the other windows. If necessary you can let the other library take over as long as it calls DispatchMessage(), but you will have to arrange for the function Fl::flush() to be called regularly so that widgets are updated, timeouts are handled, and the idle functions are called.

extern MSG fl_msg

This variable contains the most recent message read by GetMessage(), which is called by Fl::wait(). This may not be the most recent message sent to an FLTK window, because silly Windows calls the handle procedures directly for some events (sigh).

void Fl::add_handler(int (∗f)(int))

Installs a function to parse unrecognized messages sent to FLTK windows. If FLTK cannot figure out what to do with a message, it calls each of these functions (most recent first) until one of them returns non-zero. The argument passed to the functions is the FLTK event that was not handled or zero for unknown messages. If all the handlers return zero then FLTK calls DefWindowProc().

HWND fl_xid(const Fl_Window ∗)

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

Fl_Window ∗fl_find(HWND xid)

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

When the virtual function Fl_Widget::draw() is called, FLTK stores all the extra arguments you need to make a proper GDI call in some global variables:

```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.

18.3.5 Display Scaling Factor

FLTK uses the value given by function GetDpiForMonitor() divided by 96. to initialize the scaling factor of each display in the system. This matches the value of "Change the size of text, apps and other items" found in section "System" subsection "Display" of Windows settings.

18.3.6 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.
18.3.7 How to Not Get a MSDOS Console Window

Windows has a really stupid mode switch stored in the executables that controls whether or not to make a console window.

To always get a console window you simply create a console application (the "\SUBSYSTEM:CONSOLE" option for the linker). For a GUI-only application create a Windows application (the "\SUBSYSTEM:WINDOWS" option for the linker).

FLTK includes a WinMain() function that calls the ANSI standard main() entry point for you. This function creates a console window when you use the debug version of the library.

Windows applications without a console cannot write to stdout or stderr, even if they are run from a console window. Any output is silently thrown away. Additionally, Windows applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.

18.3.8 Known Windows Bugs and Problems

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

- If a program is deactivated, Fl::wait() does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in Windows and probably impossible to get around.

- Fl_Gl_Window::can_do_overlay() returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.

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

- Compilation with gcc 3.4.4 and -Os exposes an optimisation bug in gcc. The symptom is that when drawing filled circles only the perimeter is drawn. This can for instance be seen in the symbols demo. Other optimisation options such as -O2 and -O3 seem to work OK. More details can be found in STR#1656

18.4 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 FL_CTRL, the 'option' key to FL_ALT and the 'Apple' key to FL_META. Furthermore, FL_COMMAND designates the 'Apple' key on Mac OS X and the 'control' key on other platforms. Keyboard events return the key name in Fl::event_key() and the keystroke translation in Fl::event_text(). For example, typing Option-Y on a Mac US keyboard will set FL_ALT in Fl::event_state(), set Fl::event_key() to ’y’ and return the Yen symbol in Fl::event_text().
Right Click simulation with Ctrl Click

The Apple HIG guidelines indicate applications should support 'Ctrl Click' to simulate 'Right Click' for e.g. context menus, so users with one-button mice and one-click trackpads can still access right-click features. However, paraphrasing Manolo's comment on the fltk.coredev newsgroup:

- FLTK does /not/ support Ctrl-Click == Right Click itself because Mac OS X event processing doesn't support this at the system level: the system reports left-clicks with the ctrl modifier when the user ctrl-clicks, and OS X system preferences don't allow changing this behavior. Therefore, applications must handle simulation of Right Click with Ctrl Click in the application code.

Ian MacArthur provided the following handle() method code snippet showing an example of how to do this:

```c
case FL_PUSH:
    {
        int btn = Fl::event_button();
        #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 sends an FL_CLOSE event to all open windows. If any window remains open, the termination request aborts. If all windows close, the application's event loop terminates, that is, Fl::run() returns. The application can then follow FLTK's normal termination path executing cleanup code that may be programmed after termination of the event loop, and returning from main(). Function Fl::program_should_quit() allows to detect whether the event loop terminated because of a program termination request.

Apple "Open" Event

Whenever the user drops a file onto an application icon, OS X generates an Apple Event of the type "Open". You can have FLTK notify you of an Open event by calling the fl_open_callback() function.

```c
void fl_open_display()
```
Opens the display. Does nothing if it is already open. You should call this if you wish to do Cocoa or Quartz calls and there is a chance that your code will be called before the first `show()` of a window.

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.

`Fl_Window *` fl_find(Windows id)

Returns the `Fl_Window` that corresponds to the given window reference, or `NULL` if not found.

void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut)

Attaches the callback `cb` to the "About myprog" item of the system application menu. `cb` will be called with NULL first argument and `user_data` second argument. This MacOS-specific function is deprecated in FLTK 1.4 and replaced by `Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *data)` which is cross-platform.

`Fl_Sys_Menu_Bar` class

The `Fl_Sys_Menu_Bar` class allows to build menu bars that, on Mac OS X, are placed in the system menu bar (at top-left of display), and, on other platforms, at a user-chosen location of a user-chosen window.

18.4.1 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

```xml
<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.

18.4.2 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/platform.H` to declare the `fl_gc` variable.
18.4.3 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:

```plaintext
"About %@" = "Über %@";
"Print Front Window" = "Frontfenster drucken";
"Services" = "Dienste";
"Hide %@" = "%@ ausblenden";
"Hide Others" = "Andere ausblenden";
"Show All" = "Alle einblenden";
"Quit %@" = "%@ 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.

18.4.4 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 useful on all platforms because `Fl_Gl_Window::w()` and `Fl_Gl_Window::h()` don't return, on HighDPI displays, the quantities in pixels necessary to OpenGL functions.

The `Fl_Gl_Window::pixels_per_unit()` method is useful when the OpenGL code depends on the pixel dimension of the GL scene. This occurs, e.g., if a window's handle() method uses `Fl::event_x()` and `Fl::event_y()` whose returned values should be multiplied by `Fl_Gl_Window::pixels_per_unit()` to obtain the adequate pixel units. This method may also be useful, for example, to adjust the width of a line in a high resolution GL scene.
18.4.5  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.

18.4.6  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
Chapter 19

Migrating Code from FLTK 1.3 to 1.4

This appendix describes the differences between the FLTK 1.3.x and FLTK 1.4.x functions and classes.

19.1 Migrating from FLTK 1.0 or 1.1 to 1.4

If you want to migrate your code from FLTK 1.0 or 1.1 to FLTK 1.4, then you should first consult the relevant appendices in FLTK 1.3 documentation online or by downloading the FLTK 1.3 documentation. See https://www.fltk.org/doc-1.3/index.html and/or https://www.fltk.org/software.php, respectively.

19.2 Minor Changes in Header Files

(to be documented)

19.3 Fl_Preferences

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preferences files has changed slightly. Please see Fl_Preferences::Fl_Preferences(Root, const char*, const char*) for details.

If you want to retain user preferences you may want to move the preferences file from its old location to the new location as documented in Fl_Preferences::Fl_Preferences(Root, const char*, const char*).
Chapter 20

Developer Information

This chapter describes FLTK development and documentation.

Example

/** \file
   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 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 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 
doxxygen comments for the class, and use the navigation links provided. Simply using \relatesalso as the first 
doxxygen command in the function’s comment puts it in the appropriate place. There is no need to have \defgroup and \ingroup as well, and indeed they don't work. So, to summarize:

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

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.
<!-- *** This *** is *** invisible *** -->
This will be visible again.

will be shown as:

The following text is a developer comment.
This will be visible again.
Different Headlines

You can use HTML tags \texttt{<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 \texttt{<H3>} and \texttt{<H4>} in the pdf output, whereas the html output uses different font sizes.

\begin{verbatim}
\texttt{\textless H1\textgreater Headline in big text \{H1\}\textless /H1\textgreater  \\ 
\texttt{\textless H2\textgreater Headline in big text \{H2\}\textless /H2\textgreater  \\ 
\texttt{\textless H3\textgreater Headline in big text \{H3\}\textless /H3\textgreater  \\ 
\texttt{\textless H4\textgreater Headline in big text \{H4\}\textless /H4\textgreater
\end{verbatim}

Headline in big text (H1)

Headline in big text (H2)

Headline in big text (H3)

Headline in big text (H4)

20.1 Non-ASCII Characters

Doxygen understands many HTML quoting characters like \texttt{&quot;}, \texttt{&uuml;}, \texttt{&ccedil;}, \texttt{&Ccedil;} but not all HTML quoting characters.

This will appear in the document:

Doxygen understands many HTML quoting characters like \texttt{&quot;}, \texttt{&uuml;}, \texttt{&ccedil;}, \texttt{&Ccedil;} but not all HTML quoting characters.

For further informations about HTML quoting characters see
\url{http://www.doxygen.org/manual/htmlcmds.html}

Alternatively you can use \texttt{UTF-8} encoding within Doxygen comments.

20.2 Document Structure

\begin{itemize}
\item \texttt{\textbackslash page} creates a named page
\item \texttt{\textbackslash section} creates a named section within that page
\item \texttt{\textbackslash subsection} creates a named subsection within the current section
\item \texttt{\textbackslash subsubsection} creates a named subsubsection within the current subsection
\end{itemize}
All these statements take a "name" as their first argument, and a title as their second argument. The title can contain spaces.

The page, section, and subsection titles are formatted in blue color and a size like "<H1>", "<H2>", and "<H3>", and "<H4>", respectively.

By FLTK documentation convention, a file like this one with a doxygen documentation chapter has the name "<chapter>.dox". The \page statement at the top of the page is "\page <chapter> This is the title". Sections within a documentation page must be called "<chapter>_<section>", where "<chapter>" is the name part of the file, and "<section>" is a unique section name within the page that can be referenced in links. The same for subsections and subsubsections.

These doxygen page and section commands work only in special documentation chapters, not within normal source or header documentation blocks. However, links from normal (e.g. class) documentation to documentation sections do work.

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

\section{Creating Links}

Links to other documents and external links can be embedded with

- doxygen \ref links to other doxygen \page, \section, \subsection and \anchor locations
- HTML links without markup - doxygen creates "http://..." links automatically
- standard, non-Doxygen, HTML links
  - see chapter \ref unicode creates a link to the named chapter unicode that has been created with a \page statement.
  - For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
  - see <a href="https://www.fltk.org/">FLTK Library</a> creates a standard HTML link

appears as:

- see chapter Unicode and UTF-8 Support creates a link to the named chapter unicode that has been created with a \page statement.
- For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
- see FLTK Library creates a standard HTML link
20.4 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.

\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.
20.5 Navigation Elements

Each introduction (tutorial) page ends with navigation elements. These elements must only be included in the html documentation, therefore they must be separated with `htmlonly` and `endhtmlonly`.

The following code gives the navigation bar at the bottom of this page:

```html
\htmlonly
<br>
<table summary="navigation bar" width="100%" border="0">
<tr>
    <td width="45%" align="LEFT">
        <a class="el" href="migration_1_4.html">
            [Prev]
            Migrating Code from FLTK 1.3 to 1.4
        </a>
    </td>
    <td width="10%" align="CENTER">
        <a class="el" href="index.html">
            [Index]
        </a>
    </td>
    <td width="45%" align="RIGHT">
        <a class="el" href="license.html">
            Software License
            [Next]
        </a>
    </td>
</tr>
</table>
\endhtmlonly
```
Chapter 21

Software License

December 11, 2001

The FLTK library and included programs are provided under the terms of the GNU Library General Public License (LGPL) with the following exceptions:

1. Modifications to the FLTK configure script, config header file, and makefiles by themselves to support a specific platform do not constitute a modified or derivative work.

   The authors do request that such modifications be contributed to the FLTK project - send all contributions through the "Software Trouble Report" on the following page: https://www.fltk.org/bugs.php

2. Widgets that are subclassed from FLTK widgets do not constitute a derivative work.

3. Static linking of applications and widgets to the FLTK library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared FLTK libraries, or link their applications or widgets against a user-supplied version of FLTK.

   If you link the application or widget to a modified version of FLTK, then the changes to FLTK must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs that are linked to the FLTK library, nor do you have to identify the FLTK license in your program or documentation as required by section 6 of the LGPL.

   However, programs must still identify their use of FLTK. The following example statement can be included in user documentation to satisfy this requirement:

   [program/widget] is based in part on the work of the FLTK project (https://www.fltk.org).

GNU LIBRARY GENERAL PUBLIC LICENSE
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor’s protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors’ reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone’s free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don’t assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.
Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.
b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY
15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
Chapter 22

Example Source Code

The FLTK distribution contains over 60 sample applications written in, or ported to, FLTK.

If the FLTK archive you received does not contain either an 'examples' or 'test' directory, you can download the complete FLTK distribution from 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.

22.1 Example Applications: Overview

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

22.1.1 adjuster

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

animated shows a window with an animated square that shows drawing with transparency (alpha channel).

22.1.3 arc

The arc demo explains how to derive your own widget to generate some custom drawings. The sample drawings use the matrix based arc drawing for some fun effects.

22.1.4 ask

ask shows some of FLTK's standard dialog boxes. Click the correct answers or you may end up in a loop, or you may end up in a loop, or you... .

22.1.5 bitmap

This simple test shows the use of a single color bitmap as a label for a box widget. Bitmaps are stored in the X11 '.bmp' file format and can be part of the source code.

22.1.6 blocks

A wonderful and addictive game that shows the usage of FLTK timers, graphics, and how to implement sound on all platforms. blocks is also a good example for the Mac OS X specific bundle format.

22.1.7 boxtype

boxtype gives an overview of readily available boxes and frames in FLTK. More types can be added by the application programmer. When using themes, FLTK shuffles boxtypes around to give your program a new look.

22.1.8 browser

browser shows the capabilities of the Fl_Browser widget. Important features tested are loading of files, line formatting, and correct positioning of the browser data window.

22.1.9 button

The button test is a simple demo of push-buttons and callbacks.
22.1 Example Applications: Overview

22.1.10 buttons

*buttons* shows a sample of FLTK button types.

22.1.11 cairo_test

*cairo_test* shows a sample of drawing with Cairo in a *Fl_Cairo_Window*. This program can only be built if FLTK was configured with Cairo support.

22.1.12 checkers

Written by Steve Poulsen in early 1979, *checkers* shows how to convert a VT100 text-terminal based program into a neat application with a graphical UI. Check out the code that drags the pieces, and how the pieces are drawn by layering. Then tell me how to beat the computer at Checkers.

22.1.13 clock

The *clock* demo shows two analog clocks. The innards of the *Fl_Clock* widget are pretty interesting, explaining the use of timeouts and matrix based drawing.

22.1.14 colbrowser

*colbrowser* runs only on X11 systems. It reads `/usr/lib/X11/rgb.txt` to show the color representation of every text entry in the file. This is beautiful, but only moderately useful unless your UI is written in *Motif*.

22.1.15 color_chooser

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

22.1.16 cube

The *cube* demo shows the speed of OpenGL. It also tests the ability to render two OpenGL buffers into a single window, and shows OpenGL text.

22.1.17 CubeView

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

The cursor demo shows all mouse cursor shapes that come standard with FLTK. The fgcolor and bgcolor sliders work only on few systems (some version of Irix for example).

22.1.19 curve

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

22.1.20 demo

This tool allows quick access to all programs in the test directory. demo is based on the visuals of the IrixGL demo program. The menu tree can be changed by editing test/demo.menu.

22.1.21 device

Exercises the Fl_image_surface, Fl_copy_surface, and Fl_Printer classes to draw to an Fl_image object, copy graphical data to the clipboard, and for print support.

Note

The clipboard.cxx program of the 'examples' directory is a clipboard watching application that continuously displays the textual or graphical content of the system clipboard (a.k.a pasteboard on Mac OS X) exercising Fl::paste().

22.1.22 doublebuffer

The doublebuffer demo shows the difference between a single buffered window, which may flicker during a slow redraw, and a double buffered window, which never flickers, but uses twice the amount of RAM. Some modern OS's double buffer all windows automatically to allow transparency and shadows on the desktop. FLTK is smart enough to not tripple buffer a window in that case.

22.1.23 editor

FLTK has two very different text input widgets. Fl_input and derived classes are rather light weight, however Fl_text_editor is a complete port of nedit (with permission). The editor test is almost a full application, showing custom syntax highlighting and dialog creation.

22.1.24 fast_slow

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

The standard FLTK file_chooser is the result of many iterations, trying to find a middle ground between a complex browser and a fast light implementation.

22.1.26 fonts

fonts shows all available text fonts on the host system. If your machine still has some pixmap based fonts, the supported sizes will be shown in bold face. Only the first 256 fonts will be listed.

22.1.27 forms

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

22.1.28 fractals

fractals shows how to mix OpenGL, Glut and FLTK code. FLTK supports a rather large subset of Glut, so that many Glut applications compile just fine.

22.1.29 fullscreen

This demo shows how to do many of the window manipulations that are popular for games. You can toggle the border on/off, switch between single- and double-buffered rendering, and take over the entire screen. More information in the source code.

22.1.30 gl_overlay

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

22.1.31 glpuzzle

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

22.1.32 hello

hello: Hello, World. Need I say more? Well, maybe. This tiny demo shows how little is needed to get a functioning application running with FLTK. Quite impressive, I'd say.
22.1.33 help_dialog

help_dialog displays the built-in FLTK help browser. The Fl_Help_Dialog understands a subset of html and renders various image formats. This widget makes it easy to provide help pages to the user without depending on the operating system's html browser.

22.1.34 icon

icon demonstrates how an application icon can be set from an image. This icon should be displayed in the window bar (label), in the task bar, and in the task switcher (Windows: Alt-Tab). This feature is platform specific, hence it is possible that you can't see the icon. On Unix/Linux (X11) this can even depend on the Window Manager (WM).

22.1.35 iconize

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

22.1.36 image

The image demo shows how an image can be created on the fly. This generated image contains an alpha (transparency) channel which lets previous renderings 'shine through', either via true transparency or by using screen door transparency (pixelation).

22.1.37 inactive

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

22.1.38 input

This tool shows and tests different types of text input fields based on Fl_Input_. The input program also tests various settings of Fl_Input::when().

22.1.39 input_choice

input_choice tests the latest addition to FLTK1, a text input field with an attached pulldown menu. Windows users will recognize similarities to the 'ComboBox'. input_choice starts up in 'plastic' scheme, but the traditional scheme is also supported.
22.1.40 keyboard

FLTK unifies keyboard events for all platforms. The keyboard test can be used to check the return values of Fl::event_key() and Fl::event_text(). It is also great to see the modifier buttons and the scroll wheel at work. Quit this application by closing the window. The ESC key will not work.

22.1.41 label

Every FLTK widget can have a label attached to it. The label demo shows alignment, clipping, and wrapping of text labels. Labels can contain symbols at the start and end of the text, like @FLTK or @circle uh-huh @square.

22.1.42 line_style

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

22.1.43 list_visuals

This little app finds all available pixel formats for the current X11 screen. But since you are now an FLTK user, you don’t have to worry about any of this.

22.1.44 mandelbrot

mandelbrot shows two advanced topics in one test. It creates grayscale images on the fly, updating them via the idle callback system. This is one of the few occasions where the idle callback is very useful by giving all available processor time to the application without blocking the UI or other apps.

22.1.45 menubar

The menubar tests many aspects of FLTK’s popup menu system. Among the features are radio buttons, menus taller than the screen, arbitrary sub menu depth, and global shortcuts.

22.1.46 message

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

22.1.47 minimum

The minimum test program verifies that the update regions are set correctly. In a real life application, the trail would be avoided by choosing a smaller label or by setting label clipping differently.
22.1.48 native-filechooser

The native-filechooser program invokes the platform specific file chooser, if available (see Fl_Native_File_Chooser widget).

22.1.49 navigation

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

22.1.50 offscreen

offscreen shows how to draw into an offscreen image and display the offscreen image in the program window.

22.1.51 output

output shows the difference between the single line and multi line mode of the Fl_Output widget. Fonts can be selected from the FLTK standard list of fonts.

22.1.52 overlay

The overlay test app shows how easy an FLTK window can be layered to display cursor and manipulator style elements. This example derives a new class from Fl_Overlay_Window and provides a new function to draw custom overlays.

22.1.53 pack

The pack test program demonstrates the resizing and repositioning of children of the Fl_Pack group. Putting an Fl_Pack into an Fl_Scroll is a useful way to create a browser for large sets of data.

22.1.54 pixmap

This simple test shows the use of a LUT based pixmap as a label for a box widget. Pixmaps are stored in the X11 `.xpm` file format and can be part of the source code. Pixmaps support one transparent color.

22.1.55 pixmap_browser

Pixmap_browser tests the shared-image interface. When using the same image multiple times, Fl_Shared_Image will keep it only once in memory.
22.1.56 preferences

I do have my preferences in the morning, but sometimes I just can't remember a thing. This is where the Fl_Preferences come in handy. They remember any kind of data between program launches.

22.1.57 radio

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

22.1.58 resizebox

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

22.1.59 rotated_text

rotated_text shows how text can be rotated, i.e. drawn in any given angle. This demo is device specific, for instance it works under X11 only if configured with Xft.

22.1.60 resize

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

22.1.61 scroll

scroll shows how to scroll an area of widgets, one of them being a slow custom drawing. Fl_Scroll uses clipping and smart window area copying to improve redraw speed. The buttons at the bottom of the window control decoration rendering and updates.

22.1.62 shape

shape is a very minimal demo that shows how to create your own OpenGL rendering widget. Now that you know that, go ahead and write that flight simulator you always dreamt of.

22.1.63 subwindow

The subwindow demo tests messaging and drawing between the main window and 'true' sub windows. A sub window is different to a group by resetting the FLTK coordinate system to 0, 0 in the top left corner. On Win32 and X11, subwindows have their own operating system specific handle.
22.1.64 sudoku

Another highly addictive game - don't play it, I warned you. The implementation shows how to create application icons, how to deal with OS specifics, and how to generate sound.

22.1.65 symbols

symbols are a speciality of FLTK. These little vector drawings can be integrated into labels. They scale and rotate, and with a little patience, you can define your own. The rotation number refers to 45 degree rotations if you were looking at a numeric keypad (2 is down, 6 is right, etc.).

22.1.66 table

The table demo shows the features of the Fl_Table widget.

22.1.67 tabs

The tabs tool was created with fluid. It tests correct hiding and redisplaying of tabs, navigation across tabs, resize behavior, and no unneeded redrawing of invisible widgets.

The tabs application shows the Fl_Tabs widget on the left and the Fl_Wizard widget on the right side for direct comparison of these two panel management widgets.

22.1.68 threads

FLTK can be used in a multithreading environment. There are some limitations, mostly due to the underlying operating system. threads shows how to use Fl::lock(), Fl::unlock(), and Fl::awake() in secondary threads to keep FLTK happy. Although locking works on all platforms, this demo is not available on every machine.

22.1.69 tile

The tile tool shows a nice way of using Fl_Tile. To test correct resizing of subwindows, the widget for region 1 is created from an Fl_Window class.

22.1.70 tiled_image

The tiled_image demo uses a small image as the background for a window by repeating it over the full size of the widget. The window is resizable and shows how the image gets repeated.
22.1.71  tree

The tree demo shows the features of the Fl_Tree widget.

22.1.72  twowin

The twowin program tests focus transfer from one window to another window.

22.1.73  unittests

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

22.1.74  utf8

utf8 shows all fonts available to the platform that runs it, and how each font draws each of the Unicode code points ranging between U+0020 and U+FFFF.

22.1.75  valuators

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

22.1.76  windowfocus

windowfocus shows a very special case when a new window is shown while the focus stays in the original window.

22.1.77  fluid

fluid is not only a big test program, but also a very useful visual UI designer. Many parts of fluid were created using fluid. See the Fluid Tutorial for more details.

22.2  Example Applications: Images

This chapter contains a few selected images of the test and example applications listed above. It is not meant to be complete or a full reference. The reason some images are included here is to show how the display should look when running the example programs.
22.2.1 cairo_test

The cairo_test demo program shows three shiny buttons drawn with Cairo in an Fl_Cairo_Window.

![Figure 22.1 Buttons drawn with Cairo](image1.jpg)

22.2.2 icon

The icon program lets you set the program icon from an image (here an Fl_RGB_Image).

![Figure 22.2 Green icon (Windows 10)](image2.jpg)

22.2.3 unittests

Select "drawing images" in the browser at the left side to see the image drawing example:
Figure 22.3 Image Drawing
Chapter 23

FAQ (Frequently Asked Questions)

A list of frequently asked questions about FLTK.

This appendix describes various frequently asked questions regarding FLTK.

- Where do I start learning FLTK?
- How do I make a box with text?
- Can I use FLTK to make closed-source commercial applications?
- Hitting the 'Escape' key closes windows - how do I prevent this?

23.1 Where do I start learning FLTK?

It is assumed you know C++, which is the language all FLTK programs are written in, including FLTK itself.

If you like reading manuals to work your way into things, a good start is the FLTK documentation's Introduction to FLTK. Under the FLTK Basics section there's an example 'hello world' program that includes a line-by-line description.

If you like looking at simple code first to pique your interest, and then read up from there, start with the example programs in the test/ and examples/ directory that is included with the source code. A good place to start is the 'hello world' program in test/hello.cxx. Also do a google search for "FLTK example programs". "Erco’s Cheat Page" is one that shows many simple examples of how to do specific things.

If you like to run example programs and look for ones that are like yours and then read them, download and build FLTK from the source, then run the test/demo program. Also, go into the 'examples/' directory and run 'make', then run some of those programs.

If you prefer watching TV to reading books and code, google search for "FLTK video tutorials" which has some introductory examples of how to write FLTK programs in C++ and build them.

23.2 How do I make a box with text?

The 'hello world' program shows how to make a box with text. All widgets have labels, so picking a simple widget like Fl_Box and setting its label() and using align() to align the label and labelfont() to set the font, and labelsize() to set the size, you can get text just how you want.

Labels are not selectable though; if you want selectable text, you can use Fl_Output or Fl_Multiline_Output for simple text that doesn't include scrollbars. For more complex text that might want scrollbars and multiple colors/fonts, use either Fl_Text_Display which handles plain text, or Fl_Help_View which handles simple HTML formatted text.
23.3 Can I use FLTK to make closed-source commercial applications?

Yes. The FLTK Software License is standard LGPL, but also includes a special clause ("exception") to allow for static linking. Specifically:

[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 (https://www.fltk.org).

23.4 Hitting the 'Escape' key closes windows - how do I prevent this?

[From FLTK article #378]

1. FLTK has a "global event handler" that makes Escape try to close the window, the same as clicking the close box. To disable this everywhere you can install your own that pretends it wants the escape key and thus stops the default one from seeing it (this may not be what you want, see below about the callbacks):

```c
static int my_handler(int event) {
    if (event == FL_SHORTCUT) return 1; // eat all shortcut keys
    return 0;
}
```

...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:

```c
void my_callback(Fl_Widget*, void*) {
    exit(0);
}
```

...main_window->callback(my_callback);
...

If you don't want Escape to close the main window and exit you can check for and ignore it. This is better than replacing the global handler because Escape will still close pop-up windows:
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_choice("Are you sure you want to quit?", "continue", "quit", NULL))
        exit(0);
}
Chapter 24

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!

Module Box Types
   Description of boxtypes is incomplete. See below for the defined enum Fl_Boxtype.

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

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

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

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

Member fl_casenumericsort (struct dirent **A, struct dirent **B)
   Make comparison UTF-8 aware.

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
   Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

Member Fl_File_Input::errorcolor (Fl_Color c)
   Remove Fl_File_Input::errorcolor(Fl_Color) in FLTK 1.5.0 or higher.

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, int keepmark=0)
Add comment and parameters

Member **Fl_Input::handletext** (int e, int, int, int, int)
Add comment and parameters

**Class Fl_Label**

There is an aspiration that the **Fl_Label** type will become a widget by itself. That way we will be avoiding a lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in place in this release.

Member **Fl_Menu::add** (const char *, int shortcut, Fl_Callback *, void * = 0, int = 0)

Member **Fl_Preferences::Fl_Preferences** (Root root, const char * vendor, const char * application)
(Matt) Before the release of 1.4.0, I want to make a further attempt to write a preferences file smarter. I plan to use a subgroup of the "runtime" preferences to store data and stay accessible until the application exits. Data would be stored under ./${vendor}/${application}.prefs in RAM, but not on disk.
(Matt) I want a way to access the type of the root preferences (SYSTEM, USER, MEMORY), and the state of the file access (OK, FILE_SYSTEM_FAIL, PERMISSION_FAIL, etc.), and probably the dirty() flag as well.
(Matt) Also, I need to explain runtime preferences.
(Matt) Lastly, I think I have to put short sample code in the Doxygen docs. The test app ist just not enough.

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 * win=0)
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_Shortcut**
Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on all programs being coded with symbolic names and not hard coded bit values.

Member **Fl_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::highlight_data (Fl_Text_Buffer *styleBuffer, const Style_Table_Entry *styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)

"extendRangeForStyleMods" does not exist (might be a hangover from the port from nedit). Find the correct function.

Member Fl_Text_Display::maintain_absolute_top_line_number (int state)

TextDPosToLineAndCol does not exist (nedit port?)

Member Fl_Text_Display::overstrike (const char *text)

Unicode? Find out exactly what we do here and simplify.

Member Fl_Text_Display::position_to_linecol (int pos, int *lineNum, int *column) const

a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

Member Fl_Text_Display::scroll (int topLineNum, int horizOffset)

Column numbers make little sense here.

Member Fl_Text_Display::scrollbar_width () const

This method should eventually be removed.

Member Fl_Text_Display::scrollbar_width (int width)

This method should eventually be removed

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::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::wrap_uses_character (int lineEndPos) const

TextDEndOfLine and BufEndOfLine functions don't exist (nedit port?)

Member Fl_Text_Display::wrapped_column (int row, int column) const

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Unicode?

Member Fl_Text_Display::wrapped_row (int row) const

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Member Fl_Tiled_Image::Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)

Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char *).

Member Fl_Tree::handle (int e)

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

[Internal] The user_data value must be implemented using fl_intptr_t or similar to avoid 64-bit platform 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.

**Module Mouse and Keyboard Events**

`FL_Button` and `FL_key...` constants could be structured better (use an enum or some doxygen grouping ?)

**Page Unicode and UTF-8 Support**

FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus (`Fl_Shortcut`) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.

Work through the code and this documentation to harmonize the `[OksiD]` and `[fltk2]` functions.

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()`.
Chapter 25

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_Browser_::scrollbar_width () const
   Use scrollbar_size() instead.

Member Fl_Browser_::scrollbar_width (int width)
   Use scrollbar_size() 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_File_Input::errorcolor () const
   Will be removed in FLTK 1.5.0 or higher.

Member Fl_File_Input::errorcolor (Fl_Color c)
   Will be removed in FLTK 1.5.0 or higher.

Member Fl_Group::focus (Fl_Widget ∗W)
   This is for backwards compatibility only.

Member Fl_Group::sizes ()
   Deprecated since 1.4.0. Please use bounds() instead.

Member Fl_Image::draw_scaled (int X, int Y, int W, int H)
   Only for API compatibility with FLTK 1.3.4.

Member Fl_Image_Surface::highres_image ()
   Use image() 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_Text_Display::scrollbar_width()` const
  Use `scrollbar_size()` instead.

Member `Fl_Text_Display::scrollbar_width(int width)`
  Use `scrollbar_size()` instead.

Member `Fl_Tree::first_visible()`
in 1.3.3 ABI – use `first_visible_item()` instead.

Member `Fl_Tree::item_clicked(Fl_Tree_Item *val)`
in 1.3.3 ABI – use `callback_item()` instead.

Member `Fl_Tree::item_clicked()`
in 1.3.3 ABI – use `callback_item()` instead.

Member `Fl_Tree::last_visible()`
in 1.3.3 – use `last_visible_item()` instead.

Member `Fl_Tree_Item::Fl_Tree_Item(const Fl_Tree_Prefs &prefs)`
in 1.3.3 ABI – you must use `Fl_Tree_Item(Fl_Tree*)` for proper horizontal scrollbar behavior.

Member `Fl_Tree_Item::next_displayed(Fl_Tree_Prefs &prefs)`
in 1.3.3 for confusing name, use `next_visible()` instead.

Member `Fl_Tree_Item::prev_displayed(Fl_Tree_Prefs &prefs)`
in 1.3.3 for confusing name, use `prev_visible()`

Member `FL_VERSION`
This double version number is retained for compatibility with existing program code. New code should use `int FL_API_VERSION` instead. `FL_VERSION` is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

Member `Fl_Widget::color2(unsigned a)`
Use `selection_color(unsigned)` instead.

Member `Fl_Widget::color2()` const
Use `selection_color()` instead.

Member `Fl_Window::free_position()`
please use `force_position(0)` instead

Member `Fl_Window::icon()` const
in 1.3.3

Member `Fl_Window::icon(const void *ic)`
in 1.3.3
# Chapter 26

## Module Index

### 26.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callback function typedefs</td>
<td>229</td>
</tr>
<tr>
<td>runtime graphics driver configuration</td>
<td>231</td>
</tr>
<tr>
<td>runtime printer driver configuration</td>
<td>232</td>
</tr>
<tr>
<td>runtime window and event manager configuration</td>
<td>233</td>
</tr>
<tr>
<td>runtime system configuration</td>
<td>234</td>
</tr>
<tr>
<td>Windows handling functions</td>
<td>235</td>
</tr>
<tr>
<td>Events handling functions</td>
<td>238</td>
</tr>
<tr>
<td>Selection &amp; Clipboard functions</td>
<td>256</td>
</tr>
<tr>
<td>Screen functions</td>
<td>260</td>
</tr>
<tr>
<td>Color &amp; Font functions</td>
<td>267</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>278</td>
</tr>
<tr>
<td>Multithreading support functions</td>
<td>313</td>
</tr>
<tr>
<td>Safe widget deletion support functions</td>
<td>315</td>
</tr>
<tr>
<td>Cairo Support Functions and Classes</td>
<td>319</td>
</tr>
<tr>
<td>Unicode and UTF-8 functions</td>
<td>321</td>
</tr>
<tr>
<td>Mac OS X-specific symbols</td>
<td>342</td>
</tr>
<tr>
<td>Common Dialogs classes and functions</td>
<td>344</td>
</tr>
<tr>
<td>File names and URI utility functions</td>
<td>359</td>
</tr>
</tbody>
</table>
Chapter 27

Hierarchical Index

27.1 Class Hierarchy

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

- Fl_Preferences::Entry .............................................. 367
- Fl ................................................................. 367
- Fl_Cairo_State ......................................................... 466
- FL_CHART_ENTRY ..................................................... 474
- Fl_End .............................................................. 521
- Fl_File_Chooser ..................................................... 528
- Fl_File_Icon .......................................................... 540
- Fl_Gl_Choice .......................................................... 564
- Fl_Glut_Bitmap_Font ............................................... 576
- Fl_Glut_StrokeChar .................................................. 576
- Fl_Glut_StrokeFont .................................................. 576
- Fl_Glut_StrokeStrip .................................................. 577
- Fl_Glut_StrokeVertex ................................................. 577
- Fl_Help_Block ........................................................ 594
- Fl_Help_Dialog ....................................................... 594
- Fl_Help_Font_Stack ................................................... 599
- Fl_Help_Font_Style ................................................... 600
- Fl_Help_Link .......................................................... 601
- Fl_Help_Target ....................................................... 601
- Fl_Image ............................................................. 617
  - Fl_Bitmap .......................................................... 403
  - Fl_XBM_Image ....................................................... 1238
  - Fl_Pixmap .......................................................... 749
  - Fl_Gif_Img .......................................................... 563
  - Fl_XPM_Image ........................................................ 1240
  - Fl_RGB_Image ........................................................ 812
  - Fl_BMP_Img .......................................................... 407
  - Fl_JPEG_Img .......................................................... 667
  - Fl_PNG_Img .......................................................... 756
  - Fl_PNM_Img .......................................................... 758
  - Fl_SVG_Img .......................................................... 884
  - Fl_Shared_Img ......................................................... 839
  - Fl_Tiled_Img ........................................................ 1029
- Fl_Image_Reader ..................................................... 628
- Fl_Label .............................................................. 668
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Browser</td>
<td>439</td>
</tr>
<tr>
<td>Fl_Browser</td>
<td>411</td>
</tr>
<tr>
<td>Fl_File_Browser</td>
<td>526</td>
</tr>
<tr>
<td>Fl_Hold_Browser</td>
<td>613</td>
</tr>
<tr>
<td>Fl_Multi_Browser</td>
<td>721</td>
</tr>
<tr>
<td>Fl_Select_Browser</td>
<td>838</td>
</tr>
<tr>
<td>Fl_Check_Browser</td>
<td>475</td>
</tr>
<tr>
<td>Fl_Browser_Surface</td>
<td>1203</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>505</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>522</td>
</tr>
<tr>
<td>Fl_Image_Surface</td>
<td>628</td>
</tr>
<tr>
<td>Fl_Paged_Device</td>
<td>743</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>764</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>792</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>879</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>935</td>
</tr>
<tr>
<td>Fl_Text_Selection</td>
<td>1020</td>
</tr>
<tr>
<td>Fl_Tooltip</td>
<td>1037</td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1096</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1124</td>
</tr>
<tr>
<td>Fl_Tree_Prefs</td>
<td>1128</td>
</tr>
<tr>
<td>Fl_Widget</td>
<td>1155</td>
</tr>
<tr>
<td>Fl_Browser_Surface</td>
<td>1155</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td>518</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1203</td>
</tr>
<tr>
<td>Fl_Clock_Output</td>
<td>493</td>
</tr>
<tr>
<td>Fl_Clock</td>
<td>490</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>823</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>555</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>556</td>
</tr>
<tr>
<td>Fl_FormsText</td>
<td>559</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>560</td>
</tr>
<tr>
<td>Fl_Group</td>
<td>580</td>
</tr>
<tr>
<td>Fl_Browser_Surface</td>
<td>1203</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td>518</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1203</td>
</tr>
<tr>
<td>Fl_Clock_Output</td>
<td>493</td>
</tr>
<tr>
<td>Fl_Clock</td>
<td>490</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>823</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>555</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>556</td>
</tr>
<tr>
<td>Fl_FormsText</td>
<td>559</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>560</td>
</tr>
<tr>
<td>Fl_Group</td>
<td>580</td>
</tr>
<tr>
<td>Fl_Browser</td>
<td>439</td>
</tr>
<tr>
<td>Fl_Browser</td>
<td>411</td>
</tr>
<tr>
<td>Fl_File_Browser</td>
<td>526</td>
</tr>
<tr>
<td>Fl_Hold_Browser</td>
<td>613</td>
</tr>
<tr>
<td>Fl_Multi_Browser</td>
<td>721</td>
</tr>
<tr>
<td>Fl_Select_Browser</td>
<td>838</td>
</tr>
<tr>
<td>Fl_Check_Browser</td>
<td>475</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Fl_Color_Chooser</td>
<td>499</td>
</tr>
<tr>
<td>Fl_Help_View</td>
<td>602</td>
</tr>
<tr>
<td>Fl_Input_Choice</td>
<td>660</td>
</tr>
<tr>
<td>Fl_Pack</td>
<td>741</td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>824</td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>867</td>
</tr>
<tr>
<td>Fl_Table</td>
<td>900</td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>923</td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>926</td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>958</td>
</tr>
<tr>
<td>Fl_Simple_Terminal</td>
<td>849</td>
</tr>
<tr>
<td>Fl_Text_Editor</td>
<td>1006</td>
</tr>
<tr>
<td>Fl_Tile</td>
<td>1025</td>
</tr>
<tr>
<td>Fl_Tree</td>
<td>1043</td>
</tr>
<tr>
<td>Fl_Window</td>
<td>1209</td>
</tr>
<tr>
<td>Fl_Double_Window</td>
<td>518</td>
</tr>
<tr>
<td>Fl_Cairo_Window</td>
<td>467</td>
</tr>
<tr>
<td>Fl_Overlay_Window</td>
<td>737</td>
</tr>
<tr>
<td>Fl_Gl_Window</td>
<td>565</td>
</tr>
<tr>
<td>Fl_Glut_Window</td>
<td>577</td>
</tr>
<tr>
<td>Fl_Single_Window</td>
<td>861</td>
</tr>
<tr>
<td>Fl_Menu_Window</td>
<td>717</td>
</tr>
<tr>
<td>Fl_Wizard</td>
<td>1236</td>
</tr>
<tr>
<td>Fl_Input</td>
<td>637</td>
</tr>
<tr>
<td>Fl_File_Input</td>
<td>548</td>
</tr>
<tr>
<td>Fl_Float_Input</td>
<td>554</td>
</tr>
<tr>
<td>Fl_Int_Input</td>
<td>666</td>
</tr>
<tr>
<td>Fl_Multiline_Input</td>
<td>725</td>
</tr>
<tr>
<td>Fl_Output</td>
<td>735</td>
</tr>
<tr>
<td>Fl_Multiline_Output</td>
<td>726</td>
</tr>
<tr>
<td>Fl_Secret_Input</td>
<td>836</td>
</tr>
<tr>
<td>Fl_Spinner::Fl_Spinner_Input</td>
<td>875</td>
</tr>
<tr>
<td>Fl_Menu</td>
<td>675</td>
</tr>
<tr>
<td>Fl_Choice</td>
<td>486</td>
</tr>
<tr>
<td>Fl_Menu_Bar</td>
<td>693</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>889</td>
</tr>
<tr>
<td>Fl_Menu_Button</td>
<td>696</td>
</tr>
<tr>
<td>Fl_Positioner</td>
<td>759</td>
</tr>
<tr>
<td>Fl_Progress</td>
<td>799</td>
</tr>
<tr>
<td>Fl_Timer</td>
<td>1032</td>
</tr>
<tr>
<td>Fl_Valuator</td>
<td>1135</td>
</tr>
<tr>
<td>Fl_Adjuster</td>
<td>401</td>
</tr>
<tr>
<td>Fl_Counter</td>
<td>509</td>
</tr>
<tr>
<td>Fl_Simple_Counter</td>
<td>848</td>
</tr>
<tr>
<td>Fl_Dial</td>
<td>515</td>
</tr>
<tr>
<td>Fl_Fill_Dial</td>
<td>551</td>
</tr>
<tr>
<td>Fl_Line_Dial</td>
<td>673</td>
</tr>
<tr>
<td>Fl_Roller</td>
<td>819</td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>862</td>
</tr>
<tr>
<td>Fl_Fill_Slider</td>
<td>553</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider</td>
<td>614</td>
</tr>
<tr>
<td>Fl_Hor_Nice_Slider</td>
<td>615</td>
</tr>
<tr>
<td>Fl_Hor_Slider</td>
<td>615</td>
</tr>
<tr>
<td>Fl_Nice_Slider</td>
<td>735</td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>831</td>
</tr>
<tr>
<td>Fl_Value_Slider</td>
<td>1151</td>
</tr>
<tr>
<td>Class/Method</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Fl_Hor_Value_Slider</td>
<td>616</td>
</tr>
<tr>
<td>Fl_Value_Input</td>
<td>1141</td>
</tr>
<tr>
<td>Fl_Value_Output</td>
<td>1147</td>
</tr>
<tr>
<td>Fl_Widget_Tracker</td>
<td>1207</td>
</tr>
<tr>
<td>Fl_XColor</td>
<td>1239</td>
</tr>
<tr>
<td>Fl_Text_Editor::Key_Binding</td>
<td>1241</td>
</tr>
<tr>
<td>Fl_Preferences::Name</td>
<td>1241</td>
</tr>
<tr>
<td>Fl_Preferences::Node</td>
<td>1243</td>
</tr>
<tr>
<td>Fl_Paged_Device::page_format</td>
<td>1243</td>
</tr>
<tr>
<td>Fl_Preferences::RootNode</td>
<td>1244</td>
</tr>
<tr>
<td>Fl_Scroll::ScrollInfo</td>
<td>1244</td>
</tr>
<tr>
<td>Fl_Text_Display::Style_Table_Entry</td>
<td>1245</td>
</tr>
</tbody>
</table>
Chapter 28

Class Index

28.1 Class List

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

**Fl_Preferences::Entry**
- Page 367

**Fl**
- The Fl is the FLTK global (static) class containing state information and global methods for the current application.
- Page 367

**Fl_Adjuster**
- Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.
- Page 401

**Fl_Bitmap**
- Supports caching and drawing of mono-color (bitmap) images.
- Page 403

**Fl_BMP_Image**
- Supports loading, caching, and drawing of Windows Bitmap (BMP) image files.
- Page 407

**Fl_Box**
- This widget simply draws its box, and possibly its label.
- Page 408

**Fl_Browser**
- Displays a scrolling list of text lines, and manages all the storage for the text.
- Page 411

**Fl_Browser**
- This is the base class for browsers.
- Page 439

**Fl_Button**
- Buttons generate callbacks when they are clicked by the user.
- Page 460

**Fl_Cairo_State**
- Contains all the necessary info on the current cairo context.
- Page 466

**Fl_Cairo_Window**
- This defines a FLTK window with cairo support.
- Page 467

**Fl_Chart**
- **Fl_Chart** displays simple charts.
- Page 469

**FL_CHART_ENTRY**
- For internal use only.
- Page 474

**Fl_Check_Browser**
- Displays a scrolling list of text lines that may be selected and/or checked by the user.
- Page 475

**Fl_Check_Button**
- A button with a "checkmark" to show its status.
- Page 484

**Fl_Choice**
- A button that is used to pop up a menu.
- Page 486

**Fl_Clock**
- This widget provides a round analog clock display.
- Page 490
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Clock_Output</td>
<td>This widget can be used to display a program-supplied time.</td>
<td>493</td>
</tr>
<tr>
<td>Fl_Color_Chooser</td>
<td>Standard RGB color chooser.</td>
<td>499</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>Supports copying of graphical data to the clipboard.</td>
<td>505</td>
</tr>
<tr>
<td>Fl_Counter</td>
<td>Controls a single floating point value with button (or keyboard) arrows.</td>
<td>509</td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
<td>This plugin socket allows the integration of new device drivers for special window or screen types.</td>
<td>513</td>
</tr>
<tr>
<td>Fl_Dial</td>
<td>Circular dial to control a single floating point value.</td>
<td>515</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td>A display to which the computer can draw.</td>
<td>518</td>
</tr>
<tr>
<td>Fl_Double_Window</td>
<td>The Fl_Double_Window provides a double-buffered window.</td>
<td>518</td>
</tr>
<tr>
<td>Fl_End</td>
<td>This is a dummy class that allows you to end a Fl_Group in a constructor list of a class.</td>
<td>521</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>Encapsulated PostScript drawing surface.</td>
<td>522</td>
</tr>
<tr>
<td>Fl_File_Browser</td>
<td>Displays a list of filenames, optionally with file-specific icons.</td>
<td>526</td>
</tr>
<tr>
<td>Fl_File_Chooser</td>
<td>Displays a standard file selection dialog that supports various selection modes.</td>
<td>528</td>
</tr>
<tr>
<td>Fl_File_Icon</td>
<td>Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget.</td>
<td>540</td>
</tr>
<tr>
<td>Fl_File_Input</td>
<td>This widget displays a pathname in a text input field.</td>
<td>548</td>
</tr>
<tr>
<td>Fl_Fill_Dial</td>
<td>Draws a dial with a filled arc.</td>
<td>551</td>
</tr>
<tr>
<td>Fl_Fill_Slider</td>
<td>Widget that draws a filled horizontal slider, useful as a progress or value meter.</td>
<td>553</td>
</tr>
<tr>
<td>Fl_Float_Input</td>
<td>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)</td>
<td>554</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>Forms compatibility Bitmap Image Widget.</td>
<td>555</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>Forms pixmap drawing routines.</td>
<td>556</td>
</tr>
<tr>
<td>Fl FormsText</td>
<td></td>
<td>559</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>Emulation of the Forms &quot;free&quot; widget.</td>
<td>560</td>
</tr>
<tr>
<td>Fl_GIF_Image</td>
<td>Supports loading, caching, and drawing of Compuserve GIF&lt;sup&gt;SM&lt;/sup&gt; images.</td>
<td>563</td>
</tr>
<tr>
<td>Fl_Gl_Choice</td>
<td></td>
<td>564</td>
</tr>
<tr>
<td>Fl_Gl_Window</td>
<td>Sets things up so OpenGL works.</td>
<td>565</td>
</tr>
<tr>
<td>Fl_Glut_Bitmap_Font</td>
<td>Fltk glut font/size attributes used in the glutXXX functions.</td>
<td>576</td>
</tr>
<tr>
<td>Fl_Glut_StrokeChar</td>
<td></td>
<td>576</td>
</tr>
<tr>
<td>Fl_Glut_StrokeFont</td>
<td></td>
<td>576</td>
</tr>
<tr>
<td>Fl_Glut_StrokeStrip</td>
<td></td>
<td>577</td>
</tr>
<tr>
<td>Fl_Glut_StrokeVertex</td>
<td></td>
<td>577</td>
</tr>
<tr>
<td>Fl_Glut_Window</td>
<td>GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):</td>
<td>577</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Fl_Group
    FLTK container widget .......................................................... 580
Fl_Help_Block
    Displays a standard help dialog window using the Fl_Help_View widget .......... 594
Fl_Help_Dialog
    Fl_Help_View font stack element definition .................................. 600
Fl_Help_Font_Stack
    Definition of a link for the html viewer .................................... 601
Fl_Help_Font_Style
    Fl_Help_Target structure ....................................................... 601
Fl_Help_Link
    Displays HTML text .............................................................. 602
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 .............................................. 613
Fl_Hor_Fill_Slider
    Horizontal Slider class .......................................................... 615
Fl_Hor_Nice_Slider
    .............................................................. 615
Fl_Hor_Slider
    Horizontal Slider class .......................................................... 615
Fl_Hor_Value_Slider
    .............................................................. 616
Fl_Image
    Base class for image caching, scaling and drawing .......................... 617
Fl_Image_Reader
    Directs all graphics requests to an Fl_Image ................................ 628
Fl_Image_Surface
    This class provides a low-overhead text input field .......................... 637
Fl_Input
    A combination of the input widget and a menu button ......................... 660
Fl_Int_Input
    Subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xabcdef) .............................................. 666
Fl_JPEG_Image
    Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images .............................................. 667
Fl_Label
    This struct stores all information for a text or mixed graphics label ...... 668
Fl_Light_Button
    This subclass displays the "on" state by turning on a light, rather than drawing pushed in .... 670
Fl_Line_Dial
    .............................................................. 673
Fl_Mac_App_Menu
    .............................................................. 673
Fl_Menu
    Base class of all widgets that have a menu in FLTK .......................... 675
Fl_Menu_Bar
    This widget provides a standard menubar interface .................................. 693
Fl_Menu_Button
    This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects .............................................. 696
Fl_Menu_Item
    The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu__class .............................................. 700
Fl_Menu_Window
    Window type used for menus ..................................................... 717
Fl_Multi_Browser
    Subclass of Fl_Browser which lets the user select any set of the lines ...... 721
**Fl_Multi_Label**
Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.

**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_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_Radio_Button**

**Fl_Radio_Light_Button**

**Fl_Radio_Round_Button**

**Fl_Rect**
Rectangle with standard FLTK coordinates (X, Y, W, H).

**Fl_Scroll::ScrollInfo::Fl_Region_LRTB**
A local struct to manage a region defined by left/right/top/bottom.

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

**Fl_Repeat_Button**
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.

**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.

**Fl_RGB_Image**
Supports caching and drawing of full-color images with 1 to 4 channels of color information.
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Roller</td>
<td>&quot;dolly&quot; control commonly used to move 3D objects</td>
</tr>
<tr>
<td>Fl_Round_Button</td>
<td>Buttons generate callbacks when they are clicked by the user</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>A clock widget of type FL_ROUND_CLOCK</td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>This container widget lets you maneuver around a set of widgets much larger than your window</td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>Displays a slider with arrow buttons at the ends of the scrollbar</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Scrollbar_Data</td>
<td>A local struct to manage a scrollbar's xywh region and tab values</td>
</tr>
<tr>
<td>Fl_Secret_Input</td>
<td>Subclass of Fl_Input that displays its input as a string of placeholders</td>
</tr>
<tr>
<td>Fl_Select_Browser</td>
<td>The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space</td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>This class supports caching, loading, and drawing of image files</td>
</tr>
<tr>
<td>Fl_Simple_Counter</td>
<td>This widget creates a counter with only 2 arrow buttons</td>
</tr>
<tr>
<td>Fl_Simple_Terminal</td>
<td>This is a continuous text scroll widget for logging and debugging output, much like a terminal</td>
</tr>
<tr>
<td>Fl_Single_Window</td>
<td>This is the same as Fl_Window</td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>Sliding knob inside a box</td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>This widget is a combination of a numerical input widget and repeat buttons</td>
</tr>
<tr>
<td>Fl_Spinner::Fl_Spinner_Input</td>
<td></td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>A drawing surface that's susceptible to receive graphical output</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>A drawing surface producing a Scalable Vector Graphics (SVG) file</td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>Supports loading, caching and drawing of scalable vector graphics (SVG) images</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>A class to create and modify menus that appear on macOS in the menu bar at the top of the screen</td>
</tr>
<tr>
<td>Fl_Table</td>
<td>A table of widgets or other content</td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>A table with row selection capabilities</td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>&quot;file card tabs&quot; interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>This class manages Unicode text displayed in one or more Fl_Text_Display widgets</td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>Rich text display widget</td>
</tr>
<tr>
<td>Fl_Text_Editor</td>
<td>This is the FLTK text editor widget</td>
</tr>
<tr>
<td>Fl_Text_Selection</td>
<td>This is an internal class for Fl_Text_Buffer to manage text selections</td>
</tr>
<tr>
<td>Fl_Tile</td>
<td>Lets you resize its children by dragging the border between them</td>
</tr>
<tr>
<td>Fl_Tiled_Image</td>
<td>This class supports tiling of images over a specified area</td>
</tr>
</tbody>
</table>
FL_Timer
This is provided only to emulate the Forms Timer widget .................................................. 1032
FL_Toggle_Button
The toggle button is a push button that needs to be clicked once to toggle on, and one more time
to toggle off .......................................................................................................................... 1035
FL_Tooltip
Tooltip support for all FLTK widgets ...................................................................................... 1037
FL_Tree
Tree widget .............................................................................................................................. 1043
FL_Tree_Item
Tree widget item ..................................................................................................................... 1096
FL_Tree_Item_Array
Manages an array of FL_Tree_Item pointers .......................................................................... 1124
FL_Tree_Prefs
Tree widget's preferences ........................................................................................................ 1128
FL_Valuator
Controls a single floating-point value and provides a consistent interface to set the value, range,
and step, and insures that callbacks are done the same for every object .................................. 1135
FL_Value_Input
Displays a numeric value ......................................................................................................... 1141
FL_Value_Output
Displays a floating point value ................................................................................................. 1147
FL_Value_Slider
FL_Slider widget with a box displaying the current value ...................................................... 1151
FL_Widget
FL_Widget is the base class for all widgets in FLTK ................................................................ 1155
FL_Widget_Surface
A surface on which any FLTK widget can be drawn ................................................................ 1203
FL_Widget_Tracker
This class should be used to control safe widget deletion ....................................................... 1207
FL_Window
This widget produces an actual window .................................................................................... 1209
FL_Wizard
This widget is based off the FL_Tabs widget, but instead of displaying tabs it only changes "tabs"
under program control ............................................................................................................. 1236
FL_XBM_Image
Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files ............................... 1238
FL_XColor ............................................................ ................................................................. 1239
FL_XPM_Image
Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency .. 1240
FL_Text_Editor::Key_Binding
Simple linked list item associating a key/state to a function .................................................. 1241
FL_Preferences::Name
'Name' provides a simple method to create numerical or more complex procedural names for
entries and groups on the fly ......................................................................................................... 1241
FL_Preferences::Node .................................................................................................................. 1243
FL_Paged_Device::page_format
Width, height and name of a page format .................................................................................. 1243
FL_Preferences::RootNode .......................................................................................................... 1244
FL_Scroll::ScrollInfo
Structure to manage scrollbar and widget interior sizes .......................................................... 1244
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 ............................................................................................................................. 1245
Chapter 29

File Index

29.1 File List

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

- abi-version.h ........................... ??
- android.H .............................. ??
- armsgci_8.h ................................ ??
- ascii.h .................................. ??
- big5.h ................................... ??
- big5_emacs.h .............................. ??
- case.h .................................. ??
- cgdebug.h ................................ ??
- config_lib.h ................................ ??
- cp1133.h ................................ ??
- cp1251.h ................................ ??
- cp1255.h ................................ ??
- cp1256.h ................................ ??
- cp936ext.h ................................ ??
- dingbats_.h ................................ ??

Enumerations.H
- This file contains type definitions and general enumerations ....................... 1247

- fastarrow.h ................................ ??

- filename.H ............................... 1276
  - File names and URI utility functions ........................................ 1276

- Fl.cxx ..................................................... 1277
  - Implementation of the member functions of class Fl ........................ 1277

- Fl.H ..................................................... 1279
  - Fl static class ........................................ 1279

- Fl_Adjuster.H ................................. ??

- fl_arc.cxx ........................................... 1280
  - Utility functions for drawing arcs and circles ............................. 1280

- fl_ask.cxx ........................................... 1280
  - Utility functions for common dialogs ....................................... 1280

- fl_ask.H ............................................. 1282
  - API for common dialogs .................................................... 1282

- Fl_Bitmap.H ........................................ ??

- Fl_BMP_Image.H ............................... ??

- Fl_Box.H ............................................ ??

- fl_boxtype.cxx ..................................... 1284
  - Drawing code for common box types ......................................... 1284
Fl_Browser.H ................................................................. ??
Fl_Browser_H ............................................................. ??
Fl_Button.H ................................................................. ??
Fl_Cairo.H ................................................................. ??
Fl_Cairo_Window.H ......................................................... ??
Fl_Chart.H ................................................................. ??
Fl_Check_Browser.H ....................................................... ??
Fl_Check_Button_H ......................................................... ??
Fl_Choice.H ................................................................. ??
Fl_Clock.H ................................................................. ??
fl_cmap.h ................................................................. ??
fl_color.cxx
  Color handling ......................................................... 1286
Fl_Color_Chooser.H
  Fl_Color_Chooser widget ............................................. 1287
Fl_compose.cxx
  Utility functions to support text input ......................... 1287
Fl_Copy_Surface.H ....................................................... ??
Fl_Counter.H ............................................................. ??
fl_curve.cxx
  Utility for drawing Bezier curves, adding the points to the current fl_begin/fl_vertex/fl_end path . 1288
Fl_Device.H
  Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin ......................... 1288
Fl_Dial.H ................................................................. ??
Fl_Double_Window.cxx
  Fl_Double_Window implementation ................................ 1288
Fl_Double_Window.H .................................................... ??
fl_draw.H
  Utility header to pull drawing functions together ............ 1289
Fl_Export.H ............................................................... ??
Fl_File_Browser.H ....................................................... ??
Fl_File_Chooser.H ....................................................... ??
Fl_File_Icon.H ........................................................... ??
Fl_File_Input.H ............................................................ ??
Fl_Fill_Dial.H ............................................................... ??
Fl_Fill_Slider.H ............................................................ ??
Fl_Float_Input.H ............................................................. ??
Fl_FormsBitmap.H .......................................................... ??
Fl_FormsPixmap.H ........................................................... ??
Fl_Free.H ................................................................. ??
Fl_GIF_Image.H ............................................................. ??
Fl_Gl_Choice.H ............................................................. ??
Fl_Gl_Window.H ............................................................. ??
Fl_Gl_Window.Driver.H .................................................. ??
Fl_Graphics_Driver.H ................................................... ??
Fl_Group.H ................................................................. ??
Fl_Help_Dialog.H ........................................................... ??
Fl_Help_View.H ............................................................ ??
Fl_Hold_Browser.H ......................................................... ??
Fl_Hor_Fill_Slider.H ...................................................... ??
Fl_Hor_Nice_Slider.H ..................................................... ??
Fl_Hor_Slider.H ............................................................. ??
Fl_Hor_Value_Slider.H .................................................... ??
Fl_Imag.H
  Fl_Imag, Fl_RGB_Imag classes ....................................... 1294
Fl_Image_Reader.h ........................................................ ??
Fl_Image_Surface.H ........................................................ ??
Fl_Input.H ................................................................. ??
29.1 File List

Fl_Shared_Image.H .......................................................... ??
Fl_Input_H ........................................................................ ??
Fl_Input_Choice.H ............................................................ ??
Fl_Int_H ........................................................................... ??
Fl_JPEG_H ........................................................................ ??
Fl_Light_Button.H ............................................................ ??
Fl_Line_Dial.H ................................................................. ??
Fl_Menu.H .......................................................................... ??
Fl_Menu_H .......................................................................... ??
Fl_Menu_Bar.H .................................................................... ??
Fl_Menu_Button.H .............................................................. ??
Fl_Menu_Item.H ................................................................... 1295
Fl_Menu_Window.H .............................................................. ??
fl_message.H ...................................................................... ??
Fl_Multi_Browser.H ............................................................ ??
Fl_Multi_Label.H ............................................................... ??
Fl_Multiline_H ..................................................................... ??
Fl_Multiline_Output.H ......................................................... ??
Fl_Native_File_Chooser.H ..................................................... ??
Fl_Native_File_Chooser widget .............................................. 1296
Fl_Nice_Slider.H ............................................................... ??
Fl_Object.H ........................................................................ ??
Fl_Output.H .......................................................................... ??
Fl_Overlay_Window.H ........................................................ ??
Fl_Pack.H ............................................................................ ??
Fl_Paged_Device.cxx .......................................................... 1297
Implementation of class Fl_Paged_Device ............................... 1297
Fl_Paged_Device.H .............................................................. ??
Declaration of class Fl_Paged_Device ................................ ..... 1297
Fl_Pixmap.H ........................................................................ ??
Fl_Plugin.H ........................................................................ ??
Fl_PNG_H ........................................................................... ??
Fl_PNM_H ............................................................................. ??
Fl_Positioner.H ............................................................... ??
Fl_PostScript_H ......................................................................
Declaration of classes Fl_PostScript_Graphics_Driver, Fl_PostScript_File_Device ................ 1297
Fl_Preferences.H ............................................................... ??
Fl_Printer.H ........................................................................ ??
Declaration of class Fl_Printer ............................................. 1298
Fl_Progress.H ...................................................................... ??
Fl_Radio_Button.H ............................................................ ??
Fl_Radio_Light_Button.H ................................................... ??
Fl_Radio_Round_Button.H .................................................. ??
fl_rect.cxx
Drawing and clipping routines for rectangles .......................... 1298
Fl_Rect.H ............................................................................ ??
Fl_Repeat_Button.H ........................................................... ??
Fl_Return_Button.H ............................................................ ??
Fl_RGB_H .............................................................................. ??
Fl_Roller.H ......................................................................... ??
Fl_Round_Button.H ............................................................ ??
Fl_Round_Clock.H .............................................................. ??
Fl_Screen_Driver.H ............................................................. ??
Fl_Scroll.H .......................................................................... ??
Fl_Scrollbar.H ...................................................................... ??
Fl_Secret_H .......................................................................... ??
Fl_Select_Browser.H ........................................................... ??
Fl_Shared_Image.H ............................................................. ??
Fl_Shared_Image class ........................................................ 1299

Generated by Doxygen
This file defines wrapper functions for OpenGL in FLTK

Mac OS X-specific symbols

Definitions of platform-dependent types
Xutf8.h

??
Chapter 30

Module Documentation

30.1 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.
30.1.1 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

30.1.2 Typedef Documentation

30.1.2.1 Fl_Event_Dispatch

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)
30.2 runtime graphics driver configuration

Variables

- static bool Fl::cfg_gfx_cairo = 0
  Cairo rendering available, available on many platforms.
- static bool Fl::cfg_gfx_directx = 0
  DirectX rendering available, usually on Windows systems.
- static bool Fl::cfg_gfx_gdi = 0
  GDI rendering available, usually on Windows systems.
- static bool Fl::cfg_gfx_opengl = 0
  OpenGL rendering available, available on many platforms.
- static bool Fl::cfg_gfx_quartz = 0
  Quartz rendering available, usually on OS X systems.
- static bool Fl::cfg_gfx_xlib = 0
  X11 Xlib rendering available, usually on Linux systems.

30.2.1 Detailed Description
30.3 runtime printer driver configuration

Variables

- static bool Fl::cfg_prn_gdi = 0
  
  *GDI rendering available, usually on Windows systems.*

- static bool Fl::cfg_prn_ps = 0
  
  *PostScript rendering available, usually on Linux systems.*

- static bool Fl::cfg_prn_quartz = 0
  
  *Quartz rendering available, usually on OS X systems.*

30.3.1 Detailed Description
30.4 runtime window and event manager configuration

Variables

- static bool Fl::cfg_win_cocoa = 0
  
  Cocoa window management available, usually on OS X systems.

- static bool Fl::cfg_win_win32 = 0
  
  Windows window management available, on low level Windows.

- static bool Fl::cfg_win_x11 = 0
  
  X11 window management available, usually on Linux systems.

30.4.1 Detailed Description
30.5 runtime system configuration

Variables

- static bool Fl::cfg_sys_posix = 0
  
  Posix system available, usually on Linux and OS X systems, but also Cygwin.

- static bool Fl::cfg_sys_win32 = 0
  
  Windows system available, on Windows.

30.5.1 Detailed Description
30.6 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.

30.6.1 Detailed Description

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

30.6.2 Function Documentation

30.6.2.1 default_atclose()

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.
30.6.2.2 first_window() [1/2]

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.

30.6.2.3 first_window() [2/2]

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.

30.6.2.4 grab() [1/2]

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.

30.6.2.5 grab() [2/2]

void Fl::grab ( Fl_Window * win ) [static]

Selects the window to grab.

This is used when pop-up menu systems are active.

Send all events to the passed window no matter where the pointer or focus is (including in other programs). The window does not have to be shown(), this lets the handle() method of a "dummy" window override all event handling and allows you to map and unmap a complex set of windows (under both X and Windows some window must be mapped because the system interface needs a window id).

If grab() is on it will also affect show() of windows by doing system-specific operations (on X it turns on override-redirect). These are designed to make menus popup reliably and faster on the system.

To turn off grabbing do Fl::grab(0).

Be careful that your program does not enter an infinite loop while grab() is on. On X this will lock up your screen!

To avoid this potential lockup, all newer operating systems seem to limit mouse pointer grabbing to the time during which a mouse button is held down. Some OS's may not support grabbing at all.
30.6.2.6  modal()

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).

30.6.2.7  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 |

30.6.2.8  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::callback(Fl_Callback*)

30.6.3  Variable Documentation

30.6.3.1  atclose

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

Back compatibility: default window callback handler.

See also

Fl::set_atclose()
30.7 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.
- static void Fl::compose_reset ()
  If the user moves the cursor, be sure to call Fl::compose_reset().
- static void Fl::disable_im ()
  Disables the system input methods facilities.
- static void Fl::enable_im ()
  Enables the system input methods facilities.
- static int Fl::event ()
  Returns the last event that was processed.
- static int Fl::event_alt ()
  Returns non-zero if the Alt key is pressed.
- static int Fl::event_button ()
  Gets which particular mouse button caused the current event.
- static int Fl::event_button1 ()
  Returns non-zero if mouse button 1 is currently held down.
- static int Fl::event_button2 ()
  Returns non-zero if button 2 is currently held down.
- static int Fl::event_button3 ()
  Returns non-zero if button 3 is currently held down.
- static int Fl::event_buttons ()
  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- static int Fl::event_clicks ()
  Returns non zero if we had a double click event.
- static void Fl::event_clicks (int i)
  Manually sets the number returned by Fl::event_clicks().
- static void * Fl::event_clipboard ()
  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- static const char * Fl::event_clipboard_type ()
  Returns the type of the pasted data during an FL_PASTE event.
- static int Fl::event_command ()
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- static int Fl::event_ctrl ()
  Returns non-zero if the Control key is pressed.
- static void Fl::event_dispatch (Fl_Event_Dispatch d)
30.7 Events handling functions

Set a new event dispatch function.

- static Fl_Event_Dispatch Fl::event_dispatch()
  Return the current 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(int, int, int, int)
  Returns whether or not the mouse event is inside the given rectangle.

- 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_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 *)

Generated by Doxygen
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_KEY 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.

#### 30.7.1 Detailed Description

Fl class events handling API declared in `<FL/Fl.H>`.

#### 30.7.2 Function Documentation

##### 30.7.2.1 add_handler()

```c
void Fl::add_handler ( 
    Fl_Event_Handler hs ) [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 libXrender.so shared library to be loadable at run-time and the X server to implement the RandR extension.

- **FL_ZOOM_EVENT** events.

- System events that FLTK does not recognize. See fl_xevent.

- Some other events when the widget FLTK selected returns zero from its handle() method. Exactly which ones may change in future versions, however.

See also

- Fl::remove_handler(Fl_Event_Handler)
- Fl::event_dispatch(Fl_Event_Dispatch d)
- Fl::handle(int, Fl_Window ∗)
30.7.2.2 add_system_handler()

```cpp
void Fl::add_system_handler ( 
   Fl_System_Handler ha, 
   void ∗ data ) [static]
```

Install a function to intercept system events.

FLTK calls each of these functions as soon as a new system event is received. The processing will stop at the first function to return non-zero. If all functions return zero then the event is passed on for normal handling by FLTK.

Each function will be called with a pointer to the system event as the first argument and `data` as the second argument. The system event pointer will always be `void ∗`, but will point to different objects depending on the platform:

- X11: XEvent
- Windows: MSG
- OS X: NSEvent

**Parameters**

<table>
<thead>
<tr>
<th>ha</th>
<th>The event handler function to register</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>User data to include on each call</td>
</tr>
</tbody>
</table>

See also

`Fl::remove_system_handler(Fl_System_Handler)`

30.7.2.3 belowmouse() [1/2]

```cpp
static Fl_Widget ∗ Fl::belowmouse ( ) [inline], [static]
```

Gets the widget that is below the mouse.

See also

`belowmouse(Fl_Widget ∗)`

30.7.2.4 belowmouse() [2/2]

```cpp
void Fl::belowmouse ( 
   Fl_Widget ∗ o ) [static]
```

Sets the widget that is below the mouse.

This is for highlighting buttons. It is not used to send FL_PUSH or FL_MOVE directly, for several obscure reasons, but those events typically go to this widget. This is also the first widget tried for FL_SHORTCUT events.

If you change the belowmouse widget, the previous one and all parents (that don't contain the new widget) are sent FL_LEAVE events. Changing this does **not** send FL_ENTER to this or any widget, because sending FL_ENTER is supposed to test if the widget wants the mouse (by it returning non-zero from handle()).
30.7.2.5  compose()

```c
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. Widgets should also call `void Fl::reset_marked_text()` when processing `FL_UNFOC←↩US` 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_ACCENT←↩S_MENU);` in their constructor if they want to use the feature introduced with Mac OS 10.7 "Lion" where pressing and holding certain keys 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.

30.7.2.6  compose_reset()

```c
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.

30.7.2.7  disable_im()

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

Disables the system input methods facilities.

See also

```c
    enable_im()
```
30.7 Events handling functions

30.7.2.8 enable_im()

void Fl::enable_im ( ) [static]

Enables the system input methods facilities.

This is the default.

See also

   disable_im()

30.7.2.9 event()

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.

30.7.2.10 event_alt()

static int Fl::event_alt ( ) [inline], [static]

Returns non-zero if the Alt key is pressed.

30.7.2.11 event_button()

static int Fl::event_button ( ) [inline], [static]

Gets which particular mouse button caused the current event.

This returns garbage if the most recent event was not a FL_PUSH or FL_RELEASE event.

Return values

<table>
<thead>
<tr>
<th>FL_LEFT_MOUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_MIDDLE_MOUSE</td>
</tr>
<tr>
<td>FL_RIGHT_MOUSE</td>
</tr>
</tbody>
</table>

See also

   Fl::event_buttons()
30.7.2.12  event_button1()

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().

30.7.2.13  event_button2()

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

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

30.7.2.14  event_button3()

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

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

30.7.2.15  event_buttons()

static int Fl::event_buttons ( ) [inline], [static]

Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
This function returns the button state at the time of the event. During an FL_RELEASE event, the state of the released button will be 0. To find out, which button caused an FL_RELEASE event, you can use Fl::event_button() instead.

Returns

a bit mask value like { [FL_BUTTON1] | [FL_BUTTON2] | [FL_BUTTON3] }
30.7.2.17  event_clicks() [2/2]

static void Fl::event_clicks ( int i ) [inline], [static]

Manually sets the number returned by Fl::event_clicks().

This can be used to set it to zero so that later code does not think an item was double-clicked.

Parameters

| in | i | corresponds to no double-click if 0, i+1 mouse clicks otherwise |

See also

int event_clicks()

30.7.2.18  event_clipboard()

static void Fl::event_clipboard ( ) [inline], [static]

During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.

The returned data is an Fl_RGB_Image* when the result of Fl::event_clipboard_type() is Fl::clipboard_image.

30.7.2.19  event_clipboard_type()

static const char* Fl::event_clipboard_type ( ) [inline], [static]

Returns the type of the pasted data during an FL_PASTE event.

This type can be Fl::clipboard_plain_text or Fl::clipboard_image.

30.7.2.20  event_command()

static int Fl::event_command ( ) [inline], [static]

Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
30.7.21 event_ctrl()

static int Fl::event_ctrl ( ) [inline], [static]

Returns non-zero if the Control key is pressed.

30.7.22 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 \( \text{Fl::handle} \) (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 \( \text{Fl::handle}() \), then this will return the correct value.

The event dispatch can be used to handle exceptions in FLTK events and callbacks before they reach the native event handler:

```c
int myHandler(int e, Fl_Window *w) {
  try {
    return Fl::handle_(e, w);
  } catch () {
    ...  
  }
}
```

main() {
  Fl::event_dispatch(myHandler);
  ...  
  Fl::run();
}

Parameters

\( d \) new dispatch function, or NULL

See also

- \( \text{Fl::add_handler(Fl_Event_Handler)} \)
- \( \text{Fl::handle(int, Fl_Window*)} \)
- \( \text{Fl::handle_(int, Fl_Window*)} \)

30.7.23 event_dx()

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

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.

Right is positive.
### 30.7.2.24 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.

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

```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

### 30.7.2.26 event_inside() [2/2]

```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

\texttt{Fl::event_inside(int, int, int, int)}

Parameters

\begin{tabular}{|l|}
\hline
\texttt{in \( o \)} & child widget to be tested \\
\hline
\end{tabular}

Returns

non-zero, if mouse event is inside the widget

30.7.27 \texttt{event_is_click()} [1/2]

\begin{verbatim}
static int Fl::event_is_click () [inline], [static]
\end{verbatim}

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.

30.7.28 \texttt{event_is_click()} [2/2]

\begin{verbatim}
static void Fl::event_is_click ( 
    int i ) [inline], [static]
\end{verbatim}

Clears the value returned by \texttt{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.

30.7.29 \texttt{event_key()} [1/2]

\begin{verbatim}
static int Fl::event_key () [inline], [static]
\end{verbatim}

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, \texttt{Fl::event_key()} may return the 'key code' for this key, and NOT the character '5'. To always get the '5', use \texttt{Fl::event_text()} instead.

Returns

an integer 'key code', or 0 if the last event was not a key press or release.

See also

\texttt{int event_key(int), event_text(), compose(int&)}.
30.7.2.30 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 numbered function keys use the number on the function key plus FL_F. The highest possible number is FL_F_Last so you can range-check a value.
- Buttons on the mouse are considered keys, and use the button number (where the left button is 1) plus FL_Button.
- All other keys on the keypad have a symbol: FL_Escape, FL_BackSpace, FL_Tab, FL_Enter, FL_Print, FL←_Scroll_Lock, FL_Pause, FL_Insert, FL_Home, FL_Page_Up, FL_Delete, FL_End, FL_Page_Down, FL_Left, FL_Up, FL_Right, FL_Down, FL_Iso_Key, FL_Shift_L, FL_Shift_R, FL_Control_L, FL_Control_R, FL_Caps←_Lock, FL_Alt_L, FL_Alt_R, FL_Meta_L, FL_Meta_R, FL_Menu, FL_Num_Lock, FL_KP_Enter. Be careful not to confuse these with the very similar, but all-caps, symbols used by Fl::event_state().

On X Fl::get_key(FL_Button+n) does not work.

On Windows Fl::get_key(FL_KP_Enter) and Fl::event_key(FL_KP_Enter) do not work.

30.7.2.31 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.

30.7.2.32 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.
30.7.2.33 event_shift()

static int Fl::event_shift ( ) [inline], [static]

Returns non-zero if the Shift key is pressed.

30.7.2.34 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.

30.7.2.35 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().
30.7.2.36 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.

30.7.2.37 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()`.

30.7.2.38 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()`.

30.7.2.39 focus() [1/2]

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

Gets the current Fl::focus() widget.

See also

Fl::focus(Fl_Widget*)

---

Generated by Doxygen
30.7.2.40 focus() [2/2]

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()).

Widgets can set the NEEDS_KEYBOARD flag to indicate that a keyboard is essential for the widget to function. Touchscreen devices will be sent a request to show an on-screen keyboard if no hardware keyboard is connected.

See also
  Fl_Widget::take_focus()

30.7.2.41 get_key()

int Fl::get_key (  
  int key ) [static]

Returns true if the given key is held down now.

Under X this requires a round-trip to the server and is much slower than Fl::event_key(int).

See also
  event_key(int)

30.7.2.42 get_mouse()

void Fl::get_mouse (  
  int & x,  
  int & y ) [static]

Return where the mouse is on the screen by doing a round-trip query to the server.

You should use Fl::event_x_root() and Fl::event_y_root() if possible, but this is necessary if you are not sure if a mouse event has been processed recently (such as to position your first window). If the display is not open, this will open it.

30.7.2.43 handle()

int Fl::handle (  
  int e,  
  Fl_Window * window ) [static]

Handle events from the window system.

This is called from the native event dispatch after native events have been converted to FLTK notation. This function calls Fl::handle_(int, Fl_Window*) unless the user sets a dispatch function. If a user dispatch function is set, the user must make sure that Fl::handle_( ) is called, or the event will be ignored.
30.7 Events handling functions

### 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)

#### 30.7.2.44 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>$e$</th>
<th>the event type (Fl::event_number() is not yet set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>the window that caused this event</td>
</tr>
</tbody>
</table>

### Returns

0 if the event was not handled

### See also

Fl::event_dispatch(Fl_Event_Dispatch)

#### 30.7.2.45 pushed() [1/2]

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

Gets the widget that is being pushed.

### See also

void pushed(Fl_Widget*)
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()).

void Fl::remove_handler (Fl_Event_Handler ha) [static]

Removes a previously added event handler.

See also

Fl::handle(int, Fl_Window*)

void Fl::remove_system_handler (Fl_System_Handler ha) [static]

Removes a previously added system event handler.

Parameters

| ha      | The event handler function to remove |

See also

Fl::add_system_handler(Fl_System_Handler)

int Fl::testShortcut (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::testShortcut().
30.7 Events handling functions

Returns

non-zero if there is a match.

30.7.3 Variable Documentation

30.7.3.1 fl_eventnames

const char* const fl_eventnames[]

This is an array of event names you can use to convert event numbers into names.

The array gets defined inline wherever your ‘#include <FL/names.h>’ appears.

Example:

```c
#include <FL/names.h> // array will be defined here
int MyClass::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e);
    // ..resulting output might be e.g. "Event was FL_PUSH (1)".. 
    [..]
}
```

30.7.3.2 fl_fontnames

const char* const fl_fontnames[]

Initial value:

```
[ "FL_HELVETICA",
  "FL_HELVETICA_BOLD",
  "FL_HELVETICA_ITALIC",
  "FL_HELVETICA_BOLD_ITALIC",
  "FL_COURIER",
  "FL_COURIER_BOLD",
  "FL_COURIER_ITALIC",
  "FL_COURIER_BOLD_ITALIC",
  "FL_TIMES",
  "FL_TIMES_BOLD",
  "FL_TIMES_ITALIC",
  "FL_TIMES_BOLD_ITALIC",
  "FL_SYMBOL",
  "FL_SCREEN",
  "FL_SCREEN_BOLD",
  "FL_ZAPF_DINGBATS",
]
```

This is an array of font names you can use to convert font numbers into names.

The array gets defined inline wherever your ‘#include <FL/names.h>’ appears.

Example:

```c
#include <FL/names.h> // array will be defined here
int MyClass::my_callback(Fl_Widget *w, void*) { 
    int fnum = w->labelfont(); 
    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)".. 
    [..]
}
```
30.8 Selection & Clipboard functions

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

**Functions**

1. **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.*

2. **static int Fl::clipboard_contains (const char *type)**
   
   *Returns non 0 if the clipboard contains data matching type.*

3. **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).*

4. **static int Fl::dnd ()**
   
   *Initiate a Drag And Drop operation.*

5. **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.*

6. **static void Fl::paste (Fl_Widget &receiver)**
   
   *Backward compatibility only.*

7. **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.*

8. **static void Fl::selection (Fl_Widget &owner, const char *, int len)**
   
   *Changes the current selection.*

9. **static Fl_Widget * Fl::selection_owner ()**
   
   *back-compatibility only: Gets the widget owning the current selection*

10. **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**

1. **static char const *const Fl::clipboard_image = "image"**
   
   *Denotes image data.*

2. **static char const *const Fl::clipboard_plain_text = "text/plain"**
   
   *Denotes plain textual data.*

30.8.1 Detailed Description

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

30.8.2 Function Documentation
30.8.2.1 add_clipboard_notify()

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);
    //...
}
```

Note

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

30.8.2.2 clipboard_contains()

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.

30.8.2.3 copy()

void Fl::copy (const char * stuff,
    int len,
    int destination = 0,
    const char * type = Fl::clipboard_plain_text ) [static]

Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).

Copying to both is only relevant on X11, on other platforms it maps to the clipboard (1). len is the number of relevant bytes in stuff. type is always Fl::clipboard_plain_text. The selection buffer is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations.

Note

This function is, at present, intended only to copy UTF-8 encoded textual data. To copy graphical data, use the Fl_Copy_Surface class. The type argument may allow in the future to copy other kinds of data.
30.8.2.4  dnd()

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)

30.8.2.5  paste() [1/2]

void Fl::paste ( Fl_Widget & receiver,
int source,
const char ∗type = Fl::clipboard_plain_text ) [static]

Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.

The selection buffer (source is 0) is used for middle-mouse pastes and for drag-and-drop selections. The clipboard (source is 1) is used for copy/cut/paste operations.

If source is 1, the optional type argument indicates what type of data is requested from the clipboard. At present, Fl::clipboard_plain_text (requesting text data) and Fl::clipboard_image (requesting image data) are possible. Set things up so the handle function of the receiver widget will be called with an FL_PASTE event some time in the future if the clipboard does contain data of the requested type.

The handle function of receiver can process the FL_PASTE event as follows:

- If the receiver widget is known to only receive text data, the text string from the specified source is in Fl::event_text() with UTF-8 encoding, and the number of bytes is in Fl::event_length(). If Fl::paste() gets called during the drop step of a files-drag-and-drop operation, Fl::event_text() contains a list of filenames (see Drag and Drop Events).

- If the receiver widget can potentially receive non-text data, use Fl::event_clipboard_type() to determine what sort of data is being sent. If Fl::event_clipboard_type() returns Fl::clipboard_plain_text, proceed as above. It it returns 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. If receiver accepts the clipboard image, receiver.handle() should return 1 and the application should take ownership of this image (that is, delete it after use). Conversely, if receiver.handle() returns 0, the application must not use the image.

The receiver should be prepared to be called directly by this, or for it to happen later, or possibly not at all. This allows the window system to take as long as necessary to retrieve the paste buffer (or even to screw up completely) without complex and error-prone synchronization code in FLTK.

Platform details for image data:

- Unix/Linux platform: Clipboard images in PNG or BMP formats are recognized. Requires linking with the fltk_images library.
- Windows platform: Both bitmap and vectorial (Enhanced metafile) data from clipboard can be pasted as image data.
- Mac OS X platform: Both bitmap (TIFF) and vectorial (PDF) data from clipboard can be pasted as image data.
30.8.2.6 `paste()` [2/2]

```cpp
template<>
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)`

30.8.2.7 `selection()`

```cpp
template<>
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 might be the same buffer returned by `event_text()`). The `selection_owner()` widget is set to the passed owner.

30.8.2.8 `selection_owner()` [1/2]

```cpp
template<>
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*)`

30.8.2.9 `selection_owner()` [2/2]

```cpp
template<>
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.*
30.9 Screen functions

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

Functions

- static int Fl::h ()
  
  Returns the height in pixels of the main screen work area.

- static void Fl::keyboard_screen_scaling (int value)
  
  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.

- static int Fl::screen_count ()
  
  Gets the 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 float Fl::screen_scale (int n)
  
  Current value of the GUI scaling factor for screen number n.

- static void Fl::screen_scale (int n, float factor)
  
  Set the value of the GUI scaling factor for screen number n.

- static int Fl::screen_scaling_supported ()
  
  See if scaling factors are supported by this platform.

- static void Fl::screen_work_area (int &X, int &Y , int &W, int &H, 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_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_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 n)
  
  Gets the screen bounding rect for the given screen.

- static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)
  
  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by mx, my, mw, mh.

- static 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.
30.9.1 Detailed Description

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

FLTK supports high-DPI screens using a screen scaling factor. The scaling factor is initialized by the library to a value based on information obtained from the OS. If this initial value is not satisfactory, the FLTK_SCALING_F﹤ACTOR environment variable can be set to a value FLTK will use as initial scaling factor. The scaling factor value can be further changed at runtime by typing ctrl/+/-/0/ (cmd/+/-/0/ under MacOS). FLTK sends the FL_ZOOM_←EVENT when the factor value is changed, to which a callback can be associated with Fl::add_handler(). By default, FLTK also displays the new scaling factor value in a yellow, transient window. This can be changed with option Fl::OPTION_SHOW_SCALING. The scaling factor value is programmatically get and set with the Fl::screen_scale() functions.

30.9.2 Function Documentation

30.9.2.1 h()

```c
int Fl::h ( ) [static]
```

Returns the height in pixels of the main screen work area.

30.9.2.2 keyboard_screen_scaling()

```c
void Fl::keyboard_screen_scaling ( int value ) [static]
```

Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.

This function **should** be called before fl_open_display() runs. If it is not called, the default is to handle these keys for window scaling.

**Note**

This function can currently only be used to switch the internal handler **off**, i.e. `value` must be 0 (zero) - all other values result in undefined behavior and are reserved for future extension.

**Parameters**

- `value` 0 to stop recognition of ctrl/+/-/0/ (or cmd/+/-/0/ under macOS) keys as window scaling.

30.9.2.3 screen_dpi()

```c
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>

See also

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

30.9.2.4 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 |

30.9.2.5 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 |
30.9.2.6 screen_scale()

```c
void Fl::screen_scale (  
    int n, 
    float factor ) [static]
```

Set the value of the GUI scaling factor for screen number `n`.

When this function is called before the first window is show() it sets the application's initial scaling factor value. Otherwise, it sets the scale factor value of all windows mapped to screen number `n`.

30.9.2.7 screen_scaling_supported()

```c
int Fl::screen_scaling_supported ( ) [static]
```

See if scaling factors are supported by this platform.

Returns

0 if scaling factors are not supported by this platform, 1 if a single scaling factor value is shared by all screens, 2 if each screen can have its own scaling factor value.

See also

`Fl::screen_scale(int)`

30.9.2.8 screen_work_area()[1/3]

```c
void Fl::screen_work_area (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int mx,  
    int my ) [static]
```

Gets the bounding box of the work area of a screen that contains the specified screen position `mx, my`.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the work area bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mx, my</td>
<td>the absolute screen position</td>
</tr>
</tbody>
</table>

30.9.2.9 screen_work_area()[2/3]

```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

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the work area 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

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

30.9.2.10 screen_work_area() [3/3]

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

Gets the bounding box of the work area of the screen that contains the mouse pointer.

Parameters

| out | X, Y, W, H | the work area bounding box |

See also

void screen_work_area(int &x, int &y, int &w, int &h, int mx, int my)

30.9.2.11 screen_xywh() [1/4]

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

Gets the bounding box of a screen that contains the mouse pointer.
30.9 Screen functions

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)
```

30.9.2.12 **screen_xywh()** [2/4]

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

Gets the bounding box of a screen that contains the specified screen position `mx, my`.

Parameters

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

30.9.2.13 **screen_xywh()** [3/4]

```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 Windows, Mac OS X, and the Gnome desktop, screen #0 contains the menubar/taskbar

Parameters

| out | X, Y, W, H | the corresponding screen bounding box |
| in  | n          | the screen number (0 to Fl::screen_count() - 1) |
void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

30.9.2.14 screen_xywh()

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

| out | X,Y,W,H | the corresponding screen bounding box |
| in  | mx,my,mw,mh | the rectangle to search for intersection with |

See also

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

30.9.2.15 w()

int Fl::w ( ) [static]

Returns the width in pixels of the main screen work area.

30.9.2.16 x()

int Fl::x ( ) [static]

Returns the leftmost x coordinate of the main screen work area.

30.9.2.17 y()

int Fl::y ( ) [static]

Returns the topmost y coordinate of the main screen work area.
30.10 Color & Font functions

fl global color, font functions.

Functions

- **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_Color fl_color ()**
  
  Returns the last fl_color() that was set.

- **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_EXPORT void fl_font (Fl_Font face, Fl_Fontsize fsize)**
  
  Sets the current font, which is then used in various drawing routines.

- **Fl_Font fl_font ()**
  
  Returns the face set by the most recent call to fl_font().

- **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.

- **FLEXPORT 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 using the current fl_font().

- **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 using the current fl_font().

- **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.

- **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.

- **static const char * Fl::get_font (Fl_Font)**
  Gets the string for this face.

- **static const char * Fl::get_font_name (Fl_Font, int *attributes=0)**
  Get a human-readable string describing the family of this face.

- **static int Fl::get_font_sizes (Fl_Font, int *sizep)**
  Return an array of sizes in sizep.

- **static void Fl::set_color (Fl_Color, uchar, uchar, uchar)**
  Sets an entry in the fl_color index table.

- **static void Fl::set_color (Fl_Color i, unsigned c)**
  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.

30.10.1 Detailed Description

Fl global color, font functions.

These functions are declared in `<FL/Fl.H>` or `<FL/fl_draw.H>`.

30.10.2 Function Documentation

30.10.2.1 fl_color()[1/3]

```c
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.
30.10 Color & Font functions

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>c</td>
</tr>
</tbody>
</table>

30.10.2.2 fl_color()

```c
void fl_color ( uchar r, uchar g, uchar b )
```

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>type</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>r,g,b</td>
</tr>
</tbody>
</table>

30.10.2.3 fl_color()

```
Fl_Color fl_color ( )
```

Returns the last fl_color() that was set.

This can be used for state save/restore.

30.10.2.4 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>

30.10.2.5  fl_contrast()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>fg, bg</th>
<th>foreground and background colors</th>
</tr>
</thead>
</table>

Returns

contrasting color

30.10.2.6  fl_font()[1/2]

```c
FL_EXPORT void fl_font ( 
    Fl_Font face, 
    Fl_Fontsize fsize )
```

Sets the current font, which is then used in various drawing routines.

You may call this outside a draw context if necessary to measure text, for instance by calling fl_width(), fl_measure(), or fl_text_extents(), but on X this will open the display.

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

30.10.2.7  fl_font()[2/2]

```c
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.
30.10 Color & Font functions

30.10.2.8 fl_height() [1/2]

\begin{verbatim}
int fl_height ( ) [inline]
\end{verbatim}

Returns the recommended minimum line spacing for the current font.

You can also use the value of \texttt{size} passed to \texttt{fl_font()}

30.10.2.9 fl_height() [2/2]

\begin{verbatim}
FL_EXPORT int fl_height ( 
    int font,  
    int size )
\end{verbatim}

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.

30.10.2.10 fl_latin1_to_local()

\begin{verbatim}
FL_EXPORT const char * fl_latin1_to_local ( 
    const char * t,  
    int n = -1 )
\end{verbatim}

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

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{t} & character string (latin1 encoding) \\
\textbf{in} & \textbf{n} & optional number of characters to convert (default is all) \\
\hline
\end{tabular}

Returns

pointer to internal buffer containing converted characters
30.10.2.11 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</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

30.10.2.12 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.

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

30.10.2.13 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>
30.10 Color & Font functions

Returns

pointer to internal buffer containing converted characters

30.10.2.14  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 30.1 fl_show_colormap](image)

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

Generated by Doxygen
30.10.2.15  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.

30.10.2.16  fl_text_extents()

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

Determines the minimum pixel dimensions of a nul-terminated string using the current fl_font().

Usage: given a string "txt" drawn using fl_draw(txt, x, y) you would determine its pixel extents on the display using fl_text_extents(txt, dx, dy, wo, ho) such that a bounding box that exactly fits around the text could be drawn with fl_rect(x+dx, y+dy, wo, ho). Note the dx, dy values hold the offset of the first "colored in" pixel of the string, from the draw origin.

Note the desired font and font size must be set with fl_font() before calling this function.

This differs slightly from fl_measure() in that the dx/dy values are also returned.

No FLTK symbol expansion will be performed.

Example use:

int dx,dy,W,H;
fl_font (FL_HELVETICA, 12);  // set font face+size first
fl_text_extents("Some text",dx,dy,W,H);  // get width and height of string
printf("text's width=%d, height=%d
", W,H);

30.10.2.17  fl_text_extents()

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 using the current fl_font().

See also

  fl_text_extents(const char*, int& dx, int& dy, int& w, int& h)
30.10.2.18 fl_width() [1/3]

FL_EXPORT double fl_width (const char * txt)

Returns the typographical width of a nul-terminated string using the current font face and size.

30.10.2.19 fl_width() [2/3]

double fl_width (const char * txt, int n) [inline]

Returns the typographical width of a sequence of n characters using the current font face and size.

30.10.2.20 fl_width() [3/3]

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 fltk window is available when called.

30.10.2.21 free_color()

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

Frees the specified color from the colormap, if applicable.

If overlay is non-zero then the color is freed from the overlay colormap.

30.10.2.22 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.
get_color() [2/2]

```cpp
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

```cpp
unsigned get_color(Fl_Color c)
```

get_font() [2/2]

```cpp
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.

get_font_name() [2/2]

```cpp
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.

get_font_sizes() [2/2]

```cpp
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.
30.10 Color & Font functions

30.10.2.27  set_color() [1/2]

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.

30.10.2.28  set_color() [2/2]

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

Sets an entry in the fl_color index table.

You can set it to any 8-bit RGB color. The color is not allocated until fl_color(i) is used.

30.10.2.29  set_font() [1/2]

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.

30.10.2.30  set_font() [2/2]

void Fl::set_font (  
    Fl_Font fnum,  
    Fl_Font from ) [static]

Copies one face to another.

30.10.2.31  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 Windows 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.
30.11 Drawing functions

FLTK global graphics and GUI drawing functions.

Macros

- \#define \texttt{fl\_clip fl\_push\_clip}

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

Enumerations

- \texttt{enum}

  - \texttt{FL\_SOLID = 0, FL\_DASH = 1, FL\_DOT = 2, FL\_DASHDOT = 3,}
  - \texttt{FL\_DASHDOTDOT = 4, FL\_CAP\_FLAT = 0x100, FL\_CAP\_ROUND = 0x200, FL\_CAP\_SQUARE = 0x300,}
  - \texttt{FL\_JOIN\_MITER = 0x1000, FL\_JOIN\_ROUND = 0x2000, FL\_JOIN\_BEVEL = 0x3000}

Functions

- \texttt{FL\_EXPORT int fl\_add\_symbol (const char *name, void(*drawit)(Fl\_Color), int scalable)}

  Adds a symbol to the system.

- \texttt{void fl\_arc (int x, int y, int w, int h, double a1, double a2)}

  Draws ellipse sections using integer coordinates.

- \texttt{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.

- \texttt{void fl\_begin\_complex\_polygon ()}

  Starts drawing a complex filled polygon.

- \texttt{void fl\_begin\_line ()}

  Starts drawing a list of lines.

- \texttt{void fl\_begin\_loop ()}

  Starts drawing a closed sequence of lines.

- \texttt{FL\_EXPORT void fl\_begin\_offscreen (Fl\_Offscreen ctx)}

  Sends all subsequent drawing commands to this offscreen buffer.

- \texttt{void fl\_begin\_points ()}

  Starts drawing a list of points.

- \texttt{void fl\_begin\_polygon ()}

  Starts drawing a convex filled polygon.

- \texttt{char fl\_can\_do\_alpha\_blending ()}

  Checks whether platform supports true alpha blending for RGBA images.

- \texttt{FL\_EXPORT Fl\_RGB\_Image * fl\_capture\_window\_part (Fl\_Window *win, int x, int y, int w, int h)}

  Captures the content of a rectangular zone of a mapped window.

- \texttt{FL\_EXPORT void fl\_chord (int x, int y, int w, int h, double a1, double a2)}

  \texttt{fl\_chord} declaration is a place holder - the function does not yet exist

- \texttt{void fl\_circle (double x, double y, double r)}

  \texttt{fl\_circle(x,y,r)} is equivalent to \texttt{fl\_arc(x,y,r,0,360)}, but may be faster.

- \texttt{int fl\_clip\_box (int x, int y, int w, int h, int \&X, int \&Y, int \&W, int \&H)}

  Intersects a rectangle with the current clip region and returns the bounding box of the result.

- \texttt{void fl\_clip\_region (Fl\_Region r)}

  Replaces the top of the clipping stack with a clipping region of any shape.
Returns the current clipping region.

- **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_EXPORT Fl_Offscreen fl_create_offscreen (int w, int h)**
  Creation of an offscreen graphics buffer.

- **FL_EXPORT void fl_delete_offscreen (Fl_Offscreen ctx)**
  Deletion of an offscreen graphics buffer.

- **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.

- **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 \( \text{angle} \) degrees counter-clockwise.

- **void fl_draw (const char *str, int n, int x, int y)**
  Draws starting at the given \( x, y \) location a UTF-8 string of length \( n \) bytes.

- **void fl_draw (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 (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, Fl_Image *img=0, void(*)(const char *, int, int, int) callthis, int draw_symbols=1)**
  The same as fl_draw(const char *, int, int, 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.

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

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

- **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.

- **FL_EXPORT void fl_end_offscreen ()**
Quit 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.

- void fl_focus_rect (int x, int y, int w, int h)
  Draw a dotted rectangle, used to indicate keyboard focus on a widget.

- FL_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 * const * data, int &w, int &h)
  Get the dimensions of a pixmap.

- FL_EXPORT int fl_measure_pixmap (const char const *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_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.

- 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_rescale_offscreen(Fl_Offscreen &ctx)

Adapts an offscreen buffer to a changed value of the scale factor.

- FL_EXPORT void fl_reset_spot(void)

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, double y)

Concatenates scaling transformation onto the current one.

- void fl_scale(double x)

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)

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 (x1,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)

30.11.1 Detailed Description

FLTK global graphics and GUI drawing functions.

These functions are declared in `<FL/fl_draw.H>`, and in `<FL/platform.H>` for offscreen buffer-related ones.

30.11.2 Macro Definition Documentation

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

30.11.3 Enumeration Type Documentation
30.11 Drawing functions

30.11.3.1 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_BEVERL| join style: line join is tidied |

30.11.4 Function Documentation

30.11.4.1 fl_add_symbol()

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

Adds a symbol to the system.

Parameters

| in   | name         | name of symbol (without the "@") |
| in   | drawit       | function to draw symbol          |
| in   | scalable     | set to 1 if drawit uses scalable vector drawing |

Returns

1 on success, 0 on failure

30.11.4.2 fl_arc() [1/2]

```c
void fl_arc ( int x,
```

Generated by Doxygen
int y,
int w,
int h,
double a1,
double a2 ) [inline]

Draw ellipse sections using integer coordinates.

These functions match the rather limited circle drawing code provided by X and Windows. The advantage over using \texttt{fl\_arc} with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

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

\texttt{fl\_arc()} draws a series of lines to approximate the arc. Notice that the integer version of \texttt{fl\_arc()} has a different number of arguments than the double version \texttt{fl\_arc(double x, double y, double r, double start, double end)}

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, y, w, h )</td>
<td>bounding box of complete circle</td>
</tr>
</tbody>
</table>
| \( a_1, a_2 \) | start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. \( a_2 \) must be greater than or equal to \( a_1 \).

**Figure 30.2 fl\_pie() and fl\_arc()**

30.11.4.3 fl\_arc() [2/2]

void \texttt{fl\_arc} 
  
  \begin{verbatim}
  double x,
  double y,
  double r,
  double start,
  double end ) [inline]
  \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()}.  

---

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in x, y, r</th>
<th>center and radius of circular arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>in 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:

```c
// 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();
```

30.11.4.4 fl_begin_complex_polygon()

```c
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.
30.11.4.5  fl_begin_offscreen()

void fl_begin_offscreen (Fl_Offscreen ctx)

Send all subsequent drawing commands to this offscreen buffer.

Parameters

ctx  the offscreen buffer.

Note

The ctx argument must have been created by fl_create_offscreen().

30.11.4.6  fl_begin_points()

void fl_begin_points () [inline]

Starts drawing a list of points.

Points are added to the list with fl_vertex()

30.11.4.7  fl_can_do_alpha_blending()

char fl_can_do_alpha_blending () [inline]

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

30.11.4.8  fl_capture_window_part()

FL_EXPORT Fl_RGB_Image* fl_capture_window_part (Fl_Window * win,
int x,
int y,
int w,
int h)

Captures the content of a rectangular zone of a mapped window.
30.11 Drawing functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td>a mapped Fl_Window (derived types including Fl_Gl_Window are also possible)</td>
</tr>
<tr>
<td>x, y, w, h</td>
<td>window area to be captured. Intersecting sub-windows are captured too.</td>
</tr>
</tbody>
</table>

Returns

The captured pixels as an Fl_RGB_Image. The raw and drawing sizes of the image can differ. Returns NULL when capture was not successful. The image depth may differ between platforms.

Version

1.4

30.11.4.9 fl_circle()

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

fl_circle(x, y, r) is equivalent to fl_arc(x, y, r, 0, 360), but may be faster.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, r</td>
<td>center and radius of circle</td>
</tr>
</tbody>
</table>

30.11.4.10 fl_clip_box()

int fl_clip_box (int x, int y, int w, int h, int & X, int & Y, int & W, int & H) [inline]

Intersects a 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. The given rectangle (x, y, w, h) should be entirely inside its window, otherwise the result may be unexpected, i.e. this function may not clip the rectangle to the window coordinates and size. In particular x and y should not be negative.

Generated by Doxygen
The resulting bounding box can be used to limit the necessary drawing to this rectangle.

Example:

```c
void MyGroup::draw() {
    int X = 0, Y = 0, W = 0, H = 0;
    int ret = fl_clip_box(x(), y(), w(), h(), X, Y, W, H);
    if (ret == 0) { // entire group is visible (not clipped)
        // full drawing code here
    } else { // parts of this group are clipped
        // partial drawing code here (uses X, Y, W, and H to test)
    }
}
```

\( W \) and \( H \) are set to zero if the rectangle is completely outside the clipping region. In this case \( X \) and \( Y \) are undefined and should not be used. Possible values are \((0, 0), (x, y)\), or anything else (platform dependent).

Note

This function is platform-dependent. If the given rectangle is not entirely inside the window, the results are not guaranteed to be the same on all platforms.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>( x,y,w,h )</th>
<th>position and size of rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>( X,Y,W,H )</td>
<td>position and size of resulting bounding box.</td>
</tr>
</tbody>
</table>

Returns

Non-zero if the resulting rectangle is different to the original.

See also

`fl_not_clipped()`

30.11.4.11  `fl_clip_region()` [1/2]

```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.

Note

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its effect can be null if the current drawing surface is not the display.
Parameters

| in | r | clipping region |

30.11.4.12  \texttt{fl\_clip\_region()} [2/2]

\texttt{Fl\_Region fl\_clip\_region ( ) [inline]}

Returns the current clipping region.

\textbf{Note}

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its return value can be always NULL if the current drawing surface is not the display.

30.11.4.13  \texttt{fl\_copy\_offscreen()}

\texttt{void fl\_copy\_offscreen ( int x, int y, int w, int h, Fl\_Offscreen pixmap, int srcx, int srcy ) [inline]}

Copy a rectangular area of the given offscreen buffer into the current drawing destination.

Parameters

| x,y | position where to draw the copied rectangle |
| w,h | size of the copied rectangle |
| pixmap | offscreen buffer containing the rectangle to copy |
| srcx,srcy | origin in offscreen buffer of rectangle to copy |

30.11.4.14  \texttt{fl\_create\_offscreen()}

\texttt{Fl\_Offscreen fl\_create\_offscreen ( int w, int h )}

Creation of an offscreen graphics buffer.
Parameters

\begin{verbatim}
w,h          width and height in FLTK units of the buffer.
\end{verbatim}

Returns

the created graphics buffer.

The pixel size of the created graphics buffer is equal to the number of pixels in an area of the screen containing the current window sized at \( w,h \) FLTK units. This pixel size varies with the value of the scale factor of this screen.

Note

Work with the \texttt{fl\_XXX\_offscreen()} functions is equivalent to work with an \texttt{Fl\_Image\_Surface} object, as follows:

\begin{verbatim}
<table>
<thead>
<tr>
<th>Fl_Offscreen-based approach</th>
<th>Fl_Image_Surface-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Offscreen (w, h)</td>
<td>Fl_Image_Surface (w, h, 1)</td>
</tr>
<tr>
<td>fl_begin_offscreen()</td>
<td>Fl_Surface_Device::push_current(surface)</td>
</tr>
<tr>
<td>fl_end_offscreen()</td>
<td>Fl_Surface_Device::pop_current()</td>
</tr>
<tr>
<td>fl_copy_offscreen(x,y,w,h, off, sx, sy)</td>
<td>fl_copy_offscreen(x,y,w,h, surface-&gt;offscreen(), sx, sy)</td>
</tr>
<tr>
<td>fl_rescale_offscreen(off)</td>
<td>surface-&gt;rescale()</td>
</tr>
<tr>
<td>fl_delete_offscreen(off)</td>
<td>delete surface</td>
</tr>
</tbody>
</table>
\end{verbatim}

30.11.4.15  \texttt{fl\_cursor()}

\begin{verbatim}
FL\_EXPORT void fl\_cursor ( Fl\_Cursor c )
\end{verbatim}

Sets the cursor for the current window to the specified shape and colors.

The cursors are defined in the \texttt{<FL/Enumerations.H>} header file.

30.11.4.16  \texttt{fl\_curve()}

\begin{verbatim}
void fl\_curve ( double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3 ) [inline]
\end{verbatim}

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 \( X3,Y3 \).
30.11 Drawing functions

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>

30.11.4.17 fl_delete_offscreen()

```c
void fl_delete_offscreen(
    Fl_Offscreen ctx
);
```

Deletion of an offscreen graphics buffer.

Parameters

| ctx | the buffer to be deleted. |

Note

The `ctx` argument must have been created by `fl_create_offscreen()`.

30.11.4.18 fl_draw() [1/4]

```c
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.

30.11.4.19 fl_draw() [2/4]

```c
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 counterclockwise.

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.
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.

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.

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.
### 30.11 Drawing functions

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
<tr>
<td>in</td>
<td>c</td>
<td>color</td>
</tr>
</tbody>
</table>

#### 30.11.4.23 `fl_draw_image()` [1/2]

```c
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>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 \times D )), and may be larger than ( W \times D ) to crop data, or negative to flip the image vertically</td>
</tr>
</tbody>
</table>

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

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

Gray scale (1-channel) images may be drawn. This is done if abs(D) is less than 3, or by calling `fl_draw_image←_mono()`. Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting \( D \) greater than 1 will let you display one channel of a color image.

**Note:**

The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.
void fl_draw_image ( 
    Fl_Draw_Image_Cb cb, 
    void ∗ data, 
    int X, 
    int Y, 
    int W, 
    int H, 
    int D = 3 ) [inline]

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.

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)
30.11.4.26  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

fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D)
```

30.11.4.27  fl_draw_pixmap() [1/2]

```c
FL_EXPORT int fl_draw_pixmap ( 
    const char *const * data, 
    int x, 
    int y, 
    Fl_Color bg = FL_GRAY )

Draw XPM image data, with the top-left corner at the given position.

The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps.

Parameters

<table>
<thead>
<tr>
<th></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.

30.11.4.28  fl_draw_pixmap() [2/2]

```c
int fl_draw_pixmap ( 
    char *const * data, 
    int x, 
    int y, 
    Fl_Color bg = FL_GRAY ) [inline]

Draw XPM image data, with the top-left corner at the given position.
See also

\begin{verbatim}
fl_draw_pixmap(const char* const data, int x, int y, Fl_Color bg)
\end{verbatim}

### 30.11.4.29 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**

- **label**: name of symbol
- **x, y**: position of symbol
- **w, h**: size of symbol
- **col**: color of symbox

**Returns**

- 1 on success, 0 on failure

### 30.11.4.30 fl_expand_text()

```c
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.
30.11 Drawing functions

30.11.4.31 fl_frame()

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.

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

| in | s         | sets of 4 grayscale values in top, left, bottom, right order |
|    | x,y,w,h   | position and size                                           |

30.11.4.32 fl_frame2()

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.

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

| in | s         | sets of 4 grayscale values in bottom, right, top, left order |
|    | x,y,w,h   | position and size                                           |

30.11.4.33 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.
### fl_line_style()

```c
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)`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>style</code></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><code>width</code></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><code>dashes</code></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 <code>NULL</code> 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.

### fl_measure()

```c
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.
30.11 Drawing functions

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>nul-terminated string</th>
</tr>
</thead>
<tbody>
<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

30.11.4.36 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

30.11.4.37 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)
30.11.4.38  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

| In  | a,b,c,d,x,y | transformation matrix elements such that \( X' = aX + cY + x \) and \( Y' = bX + dY + y \) |

30.11.4.39  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.

See also

fl_clip_box()
30.11 Drawing functions

30.11.4.40 fl_oldShortcut()

```c
FL_EXPORT unsigned int fl_oldShortcut(const char * s)
```

Emulation of XForms named shortcuts.

Converts ASCII shortcut specifications (eg. "^c") into the FLTK integer equivalent (eg. FL_CTRL+'c')

These ASCII characters are used to specify the various keyboard modifier keys:

- `¶` - Alt
- `+` - Shift
- `^` - Control
- `!` - Meta
- `@` - Command (Ctrl on linux/win, Meta on OSX)

These special characters can be combined to form chords of modifier keys. (See 'Remarks' below)

After the optional modifier key prefixes listed above, one can either specify a single keyboard character to use as the shortcut, or a numeric sequence in hex, decimal or octal.

Examples:

- "c" -- Uses 'c' as the shortcut
- "^c" -- Same as FL_CTRL+'c'
- "#^c" -- Same as FL_ALT|FL_CTRL|'c'
- "^8c" -- Same as FL_COMMAND|'c' (see FL_COMMAND for platform specific behavior)
- "0x63" -- Same as "c" (hex 63=='c')
- "99" -- Same as "c" (dec 99=='c')
- "0143" -- Same as "c" (octal 0143=='c')
- "0x63" -- Same as (FL_CTRL|'c'), or (FL_CTRL|0x63)
- "99" -- Same as (FL_CTRL|'c'), or (FL_CTRL|99)
- "0143" -- Same as (FL_CTRL|'c'), or (FL_CTRL|0143)

Remarks

Due to XForms legacy, there are some odd things to consider when using the modifier characters.

1. You can use the special modifier keys for chords only if the modifiers are provided in this order: #, +, ^, !, @. Other ordering can yield undefined results.
   So for instance, Ctrl-Alt-c must be specified as "#^c" (and not "^#c"), due to the above ordering rule.
2. If you want to make a shortcut that uses one of the special modifier characters (as the character being modified), then to avoid confusion, specify the numeric equivalent, e.g.

<table>
<thead>
<tr>
<th>If you want..</th>
<th>Then use..</th>
</tr>
</thead>
<tbody>
<tr>
<td>'¶' as the shortcut..</td>
<td>&quot;0x23&quot; (instead of just &quot;¶&quot;).</td>
</tr>
<tr>
<td>'c' as the shortcut..</td>
<td>&quot;0x2b&quot; (instead of just &quot;c&quot;).</td>
</tr>
<tr>
<td>'^' as the shortcut..</td>
<td>&quot;0x5e&quot; (instead of just &quot;^&quot;),</td>
</tr>
<tr>
<td>Alt-+ as the shortcut..</td>
<td>&quot;0x2b&quot; (instead of &quot;++&quot;).</td>
</tr>
<tr>
<td>Alt-^ as the shortcut..</td>
<td>&quot;0x5e&quot; (instead of &quot;^^&quot;).</td>
</tr>
<tr>
<td>..etc..</td>
<td>..etc..</td>
</tr>
</tbody>
</table>

As a general rule that's easy to remember, unless the shortcut key to be modified is a single alpha-numeric character [A-Z,a-z,0-9], it's probably best to use the numeric equivalents.
Don't fix these silly legacy issues in a future release. Nobody is using this anymore.
30.11.4.41 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>x, y, w, h</td>
<td>bounding box of complete circle</td>
</tr>
<tr>
<td>a1, a2</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>

![Figure 30.4 fl_pie() and fl_arc()](image)

30.11.4.42 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.
30.11.4.3 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.

30.11.4.44 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.

30.11.4.45 fl_push_clip()  

```c
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 |

30.11.4.46 fl_push_matrix()

```c
void fl_push_matrix ( ) [inline]
```

Saves the current transformation matrix on the stack.

The maximum depth of the stack is 32.
30.11.4.47  fl_read_image()

FL_EXPORT uchar* fl_read_image (  
  uchar * p,  
  int X,  
  int Y,  
  int w,  
  int h,  
  int alpha )

Reads an RGB(A) image from the current window or off-screen buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>pixel buffer, or NULL to allocate one</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y</td>
<td>position of top-left of image to read</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
<td>width and height of image to read</td>
</tr>
<tr>
<td>in</td>
<td>alpha</td>
<td>alpha value for image (0 for none)</td>
</tr>
</tbody>
</table>

Returns

pointer to pixel buffer, or NULL if allocation failed.

The `p` argument points to a buffer that can hold the image and must be at least `w*h*3` bytes when reading RGB images, or `w*h*4` bytes when reading RGBA images. If NULL, `fl_read_image()` will create an array of the proper size which can be freed using `delete[]`.

The `alpha` parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.

See also

`fl_capture_window_part()`

30.11.4.48  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.
30.11.4.49  fl_rectf()

```c
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.

30.11.4.50  fl_rescale_offscreen()

```c
void fl_rescale_offscreen {
    Fl_Offscreen & ctx
}
```

Adapts an offscreen buffer to a changed value of the scale factor.

The \( ctx \) argument must have been created by `fl_create_offscreen()` and the calling context must not be between `fl_begin_offscreen()` and `fl_end_offscreen()`. The graphical content of the offscreen is preserved. The current scale factor value is given by `Fl_Graphics_Driver::default_driver().scale()`.

Version

1.4

30.11.4.51  fl_reset_spot()

```c
FL_EXPORT void fl_reset_spot {
    void
}
```

**Todo** provide user documentation for `fl_reset_spot` function

30.11.4.52  fl_rotate()

```c
void fl_rotate {
    double d
}
```

Concatenates rotation transformation onto the current one.
Parameters

| in  | d   | - rotation angle, counter-clockwise in degrees (not radians) |

30.11.4.53  fl_scale() [1/2]

```c
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 |

30.11.4.54  fl_scale() [2/2]

```c
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 |

30.11.4.55  fl_scroll()

```c
FL_EXPORT void fl_scroll (  
    int X,
    int Y,
    int W,
    int H,
    int dx,
    int dy,
    void(*)(void *, int, int, int) draw_area,
    void *data )
```

Scroll a rectangle and draw the newly exposed portions.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>position of top-left of rectangle</td>
</tr>
<tr>
<td>W, H</td>
<td>size of rectangle</td>
</tr>
<tr>
<td>dx, dy</td>
<td>pixel offsets for shifting rectangle</td>
</tr>
<tr>
<td>draw_area</td>
<td>callback function to draw rectangular areas</td>
</tr>
<tr>
<td>data</td>
<td>pointer to user data for callback The contents of the rectangular area is first shifted by dx and dy pixels. The draw_area callback is then called for every newly exposed rectangular area.</td>
</tr>
</tbody>
</table>

30.11.4.56 fl_set_spot()

```c
FLEXPORT void fl_set_spot (    int font,    int size,    int X,    int Y,    int W,    int H,
     Fl_Window * win = 0 )
```

Todo provide user documentation for fl_set_spot function

30.11.4.57 fl_set_status()

```c
FLEXPORT void fl_set_status (    int X,    int Y,    int W,    int H )
```

Todo provide user documentation for fl_set_status function
Get a human-readable string from a shortcut value.

Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero then an empty string is returned. The return value points at a static buffer that is overwritten with each call.

Since

FLTK 1.3.4 modifier key names can be localized, but key names can not yet be localized. This may be added to a future FLTK version.

Modifier key names (human-readable shortcut names) can be defined with the following global const char * pointer variables:

- fl_local_ctrl => name of FL_CTRL
- fl_local_alt => name of FL_ALT
- fl_local_shift => name of FL_SHIFT
- fl_local_meta => name of FL_META

Note

Due to random static initialization order this should always be done from code in main() or called by main() as opposed to static initialization since the default strings in the FLTK library are set by static initializers. Otherwise this might result in the wrong order so FLTK's internal initialization overwrites your strings.

The shortcut name will be constructed by adding all modifier names in the order defined above plus the name of the key. A '+' character is added to each modifier name unless it has a trailing '\' or a trailing '+'.

Example:

Ctrl+Alt+Shift+Meta+F12

The default values for modifier key names are as given above for all platforms except macOS. macOS uses graphical characters that represent the typical macOS modifier names in menus, e.g. cloverleaf, saucepan, etc. You may, however, redefine macOS modifier names as well.

Parameters

|  | shortcut | the integer value containing the ASCII character or extended keystroke plus modifiers |
Returns

a pointer to a static buffer containing human readable text for the shortcut

30.11.4.59 fl_shortcut_label() [2/2]

FL_EXPORT const char* fl_shortcut_label ( 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

fl_shortcut_label(unsigned int shortcut)

30.11.4.60 fl_transform_dx()  

double fl_transform_dx ( double x,
                        double y ) [inline]

Transforms distance using current transformation matrix.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>coordinate</th>
</tr>
</thead>
</table>

30.11.4.61 fl_transform_dy()  

double fl_transform_dy ( double x,
                        double y ) [inline]
Transforms distance using current transformation matrix.
30.11 Drawing functions

Parameters

30.11.4.62 fl_transform_x()

double fl_transform_x ( double x, double y ) [inline]

Transforms coordinate using the current transformation matrix.

Parameters

30.11.4.63 fl_transform_y()

double fl_transform_y ( double x, double y ) [inline]

Transforms coordinate using the current transformation matrix.

Parameters

30.11.4.64 fl_transformed_vertex()

void fl_transformed_vertex ( double xf, double yf ) [inline]

Adds coordinate pair to the vertex list without further transformations.

Parameters

Generated by Doxygen
30.11.4.65  fl_translate()

```c
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 |

30.11.4.66  fl_vertex()

```c
void fl_vertex (  
        double x,  
        double y )  [inline]
```

Adds a single vertex to the current path.

**Parameters**

| in  | x,y | coordinate |

Generated by Doxygen
30.12 Multithreading support functions

fl multithreading support functions declared in `<FL/Fl.H>`

**Functions**

- 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::awake (Fl_Awake_Handler cb, void *message=0)
  
  See void awake(void *message=0).

- 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.

30.12.1 Detailed Description

fl multithreading support functions declared in `<FL/Fl.H>`

30.12.2 Function Documentation

30.12.2.1 awake() [1/2]

void Fl::awake (  
    void * msg = 0 ) [static]

Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

Multiple calls to Fl::awake() will queue multiple pointers for the main thread to process, up to a system-defined (typically several thousand) depth. The default message handler saves the last message which can be accessed using the Fl::thread_message() function.

In the context of a threaded application, a call to Fl::awake() with no argument will trigger event loop handling in the main thread. Since it is not possible to call Fl::flush() from a subsidiary thread, Fl::awake() is the best (and only, really) substitute.

See also: Multithreading
### 30.12.2.2 awake()

```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)
```

### 30.12.2.3 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

### 30.12.2.4 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

### 30.12.2.5 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
30.13 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.
- static void Fl::watch_widget_pointer (Fl_Widget *&w)
  Adds a widget pointer to the watch list.

30.13.1 Detailed Description

These functions, declared in `<FL/Fl.H>`, support deletion of widgets inside callbacks.

Fl::delete_widget() should be called when deleting widgets or complete widget trees (Fl_Group, Fl_Window, ...) inside callbacks.

The other functions are intended for internal use. The preferred way to use them is by using the helper class Fl_Widget_Tracker.

The following is to show how it works ...

There are three groups of related methods:

1. scheduled widget deletion
   - Fl::delete_widget() schedules widgets for deletion
   - Fl::do_widget_deletion() deletes all scheduled widgets

2. widget watch list ("smart pointers")
   - Fl::watch_widget_pointer() adds a widget pointer to the watch list
   - Fl::release_widget_pointer() removes a widget pointer from the watch list
   - Fl::clear_widget_pointer() clears a widget pointer in the watch list

3. the class Fl_Widget_Tracker:
   - the constructor calls Fl::watch_widget_pointer()
   - the destructor calls Fl::release_widget_pointer()
   - the access methods can be used to test, if a widget has been deleted
     See also
     Fl_Widget_Tracker.
30.13.2 Function Documentation

30.13.2.1 clear_widget_pointer()

```cpp
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

30.13.2.2 delete_widget()

```cpp
void Fl::delete_widget ( 
    Fl_Widget * wi ) [static]
```

Schedules a widget for deletion at the next call to the event loop.

Use this method to delete a widget inside a callback function.

To avoid early deletion of widgets, this function should be called toward the end of a callback and only after any call to the event loop (Fl::wait(), Fl::flush(), Fl::check(), fl_ask(), etc.).

When deleting groups or windows, you must only delete the group or window widget and not the individual child widgets.

Since

FLTK 1.3.4 the widget will be hidden immediately, but the actual destruction will be delayed until the event loop is finished. Up to FLTK 1.3.3 windows wouldn't be hidden before the event loop was done, hence you had to hide() a window in your window close callback if you called Fl::delete_widget() to destroy (and hide) the window.

FLTK 1.3.0 it is not necessary to remove widgets from their parent groups or windows before calling this, because it will be done in the widget's destructor, but it is not a failure to do this nevertheless.

Note

In FLTK 1.1 you must remove widgets from their parent group (or window) before deleting them.

See also

Fl_Widget::~Fl_Widget()
30.13 Safe widget deletion support functions

30.13.2.3 do_widget_deletion()

```cpp
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)`

30.13.2.4 release_widget_pointer()

```cpp
void Fl::release_widget_pointer ( Fl_Widget *& w ) [static]
```

Releases a widget pointer from the watch list.

This is used to remove a widget pointer that has been added to the watch list with `Fl::watch_widget_pointer()`, when it is not needed anymore.

Note

- Internal use only, please use class `Fl_Widget_Tracker` instead.

See also

- `Fl::watch_widget_pointer()`

30.13.2.5 watch_widget_pointer()

```cpp
void Fl::watch_widget_pointer ( Fl_Widget *& w ) [static]
```

Adds a widget pointer to the widget watch list.
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
### 30.14 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 FLTK window with cairo support.

#### Functions
- **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 bool Fl::cairo_autolink_context ()**
  
  Gets the current autolink mode for cairo support.
- **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 wi.

### 30.14.1 Detailed Description

### 30.14.2 Function Documentation

#### 30.14.2.1 cairo_autolink_context() [1/2]

```c
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

#### 30.14.2.2 cairo_autolink_context() [2/2]

```c
static bool Fl::cairo_autolink_context () [inline], [static]
```

Gets the current autolink mode for cairo support.
Return values

| false  | if no cairo context autolink is made for each window. |
| true   | if any fltk window is attached a cairo context when it is current. |

See also

void cairo_autolink_context(bool alink)

Note

Only available when configure has the –enable-cairo option

30.14.2.3 cairo_cc() [1/2]

static cairo_t* Fl::cairo_cc() [inline], [static]

Gets the current cairo context linked with a fltk window.

30.14.2.4 cairo_cc() [2/2]

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

30.14.2.5 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_make_current() does not need to be freed as it is freed every time a new cairo context is created. When the program terminates, a call to Fl::cairo_make_current(0) will destroy any residual context.

Note

A new cairo context is not always re-created when this method is used. In particular, if the current graphical context and the current window didn’t change between two calls, the previous gc is internally kept, thus optimizing the drawing performances. Also, after this call, Fl::cairo_cc() is adequately updated with this cairo context.

Only available when configure has the –enable-cairo option

Returns

the valid cairo_t+ cairo context associated to this window.
30.15 Unicode and UTF-8 functions

fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

**Macros**

- `#define ERRORS_TO_CP1252 1`
  
  Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.

- `#define ERRORS_TO_ISO8859_1 1`
  
  Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.

- `#define NBC 0xFFFF + 1`

  A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.

**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_chdir (const char * path)`
  
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

- `FL_EXPORT int fl_chmod (const char * f, int mode)`
  
  Cross-platform function to set a file's 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 * buf, int len)`
  
  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 int fl_nonspacing (unsigned int ucs)`
  
  Returns true if the Unicode character `ucs` is non-spacing.

- `FL_EXPORT int fl_open (const char * fname, int oflags,...)`
  
  Cross-platform function to open files with a UTF-8 encoded name.

- `FL_EXPORT int fl_open_ext (const char * fname, int binary, int oflags,...)`
  
  Cross-platform function to open files with a UTF-8 encoded name.

- `FL_EXPORT int fl_putenv (const char * var)`
  
  Cross-platform function to write environment variables with a UTF-8 encoded name or value.

- `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)
  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
• FL_EXPORT int fl_unlink (const char *fname)
  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_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_utf8locale ()
  Returns true if the "locale" seems to indicate that UTF-8 encoding is used.
• FL_EXPORT 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.
• FL_EXPORT 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).
• FL_EXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.
• FL_EXPORT unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 16-bit characters.
• FL_EXPORT unsigned fl_utf8toc (const char *src, unsigned srclen, wchar_t * dst, unsigned dstlen)
  Converts a UTF-8 string into a wide character string.
• 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_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 ()
  Returns true if the "locale" seems to indicate that UTF-8 encoding is used.
• FL_EXPORT 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.
• FL_EXPORT 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).
• FL_EXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.
• FL_EXPORT unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 16-bit characters.
• 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.

30.15.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in <FL/fl_utf8.h>

30.15.2 Macro Definition Documentation

30.15.2.1 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.

30.15.2.2 ERRORS_TO_ISO8859_1

#define ERRORS_TO_ISO8859_1 1

Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.

If this is zero they are instead turned into the Unicode REPLACEMENT CHARACTER, of value 0xfffd. If this is on fl_utf8decode() will correctly map most (perhaps all) human-readable text that is in ISO-8859-1. This may allow you to completely ignore character sets in your code because virtually everything is either ISO-8859-1 or UTF-8.

30.15.2.3 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.
30.15.3 Function Documentation

30.15.3.1 fl_access()

```c
int fl_access (  
    const char * f,  
    int mode )
```

Cross-platform function to test a files access() with a UTF-8 encoded name or value.

This function is especially useful on the Windows platform where the standard access() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f</code></td>
<td>the UTF-8 encoded filename</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>the mode to test</td>
</tr>
</tbody>
</table>

Returns

the return value of _waccess() on Windows or access() on other platforms.

30.15.3.2 fl_chdir()

```c
int fl_chdir (  
    const char * path )
```

Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

This function is especially useful on the Windows platform where the standard _wchdir() function needs a `path` in UTF-16 encoding.

The `path` is converted to a system specific encoding if necessary and the system specific `chdir(converted←_path)` function is called.

The function returns 0 on success and -1 on error. Depending on the platform, `errno` may be set if an error occurs.

Note

The possible `errno` values are platform specific. Refer to the documentation of the platform specific `chdir()` function.

If the function is not implemented on a particular platform the default implementation returns -1 and `errno` is not set.

If the `path` is NULL the function returns -1, but `errno` is not changed. This is a convenience feature of `fl_chdir()` as opposed to `chdir()`.
30.15 Unicode and UTF-8 functions

Parameters

| in | path | the target directory for chdir (may be NULL) |

Returns

0 if successful, -1 on error (errno may be set)

30.15.3.3 fl_chmod()

```c
int fl_chmod {
    const char * f,
    int mode 
}
```

Cross-platform function to set a files mode() with a UTF-8 encoded name or value.

This function is especially useful on the Windows platform where the standard chmod() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

| in | f | the UTF-8 encoded filename |
| in | mode | the mode to set |

Returns

the return value of _wchmod() on Windows or chmod() on other platforms.

30.15.3.4 fl_fopen()

```c
FILE * fl_fopen {
    const char * f,
    const char * mode 
}
```

Cross-platform function to open files with a UTF-8 encoded name.

This function is especially useful on the Windows platform where the standard fopen() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

| f | the UTF-8 encoded filename |
| mode | same as the second argument of the standard fopen() function |
Returns

a FILE pointer upon successful completion, or NULL in case of error.

See also

fl_open().

30.15.3.5 fl_getcwd()

```c
char * fl_getcwd (  
    char * buf,  
    int len  
)
```

Cross-platform function to get the current working directory as a UTF-8 encoded value.

This function is especially useful on the Windows platform where the standard _wgetcwd() function returns UTF-16 encoded non-ASCII filenames.

If `buf` is `NULL` a buffer of size `(len+1)` is allocated, filled with the current working directory, and returned. In this case the buffer must be released by the caller with `free()` to prevent memory leaks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>buf</th>
<th>the buffer to populate (may be NULL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>the length of the buffer</td>
</tr>
</tbody>
</table>

Returns

the CWD encoded as UTF-8

30.15.3.6 fl_getenv()

```c
char * fl_getenv (  
    const char * v  
)
```

Cross-platform function to get environment variables with a UTF-8 encoded name or value.

This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters. The returned value of the variable is encoded in UTF-8 as well.

On platforms other than Windows this function calls `getenv` directly. The return value is returned as-is.

The return value is a pointer to an implementation defined buffer:

- an internal buffer that is (re)allocated as needed (Windows) or
• the string in the environment itself (Unix, Linux, MacOS) or
• any other implementation (other platforms). This string must be considered read-only and must not be freed
by the caller.

If the resultant string is to be used later it must be copied to a safe place. The next call to fl_getenv() or any other
environment changes may overwrite the string.

Note
This function is not thread-safe.

Parameters

| in | v | the UTF-8 encoded environment variable |

Returns

the environment variable in UTF-8 encoding, or NULL in case of error.

30.15.3.7 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.

Parameters

| in | path | a Unix style ("/" forward slashes) absolute or relative pathname |

Returns

1 if the path was created, 0 if creating the path failed at some point

30.15.3.8 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.
30.15.3.9 fl_mkdir()

```c
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 Windows platform where the standard _wmkdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>in f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<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.

30.15.3.10 fl_nonspacing()

```c
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/#base_character)
- [http://unicode.org/glossary/#nonspacing_mark](http://unicode.org/glossary/#nonspacing_mark)
- [http://unicode.org/glossary/#combining_character](http://unicode.org/glossary/#combining_character)

30.15.3.11 fl_open()

```c
int fl_open (const char ∗ fname, int oflags, ... )
```

Cross-platform function to open files with a UTF-8 encoded name.

This function is especially useful on the Windows platform where the standard open() function fails with UTF-8 encoded non-ASCII filenames.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fname</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<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_open(), fl_open_ext().

30.15.3.12 fl_open_ext()

```
int fl_open_ext (  
    const char ∗ fname,  
    int binary,  
    int oflags,  
    ... )
```

Cross-platform function to open files with a UTF-8 encoded name.

In comparison with fl_open(), this function allows to control whether the file is opened in binary (a.k.a. untranslated) mode. This is especially useful on the Windows platform where files are by default opened in text (translated) mode.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fname</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>binary</td>
<td>if non-zero, the file is to be accessed in binary (a.k.a. untranslated) mode.</td>
</tr>
<tr>
<td>in</td>
<td>oflags</td>
<td>these arguments are as in the standard open() function. Setting oflags to zero opens the file for reading.</td>
</tr>
</tbody>
</table>

Returns

a file descriptor upon successful completion, or -1 in case of error.

30.15.3.13 fl_putenv()

```
int fl_putenv (  
    const char ∗ var )
```

Cross-platform function to write environment variables with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters.

The given argument `var` must be encoded in UTF-8 in the form "name=value". The `name` part must conform to platform dependent restrictions on environment variable names.

The string given in `var` is copied and optionally converted to the required encoding for the platform. On platforms other than Windows this function calls `putenv` directly.

The return value is zero on success and non-zero in case of error. The value in case of error is platform specific and returned as-is.

Note

The copied string is allocated on the heap and "lost" on some platforms, i.e. calling `fl_putenv()` to change environment variables frequently may cause memory leaks. There may be an option to avoid this in a future implementation.

This function is not thread-safe.

Parameters

| in | var | the UTF-8 encoded environment variable 'name=value' |

Returns

0 on success, non-zero in case of error.

30.15.3.14 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 Windows platform where the standard `_wrename()` function expects UTF-16 encoded non-ASCII filenames.

Parameters

| in | f | the UTF-8 encoded filename to change |
| in | n | the new UTF-8 encoded filename to set |

Returns

the return value of `_wrename()` on Windows or `rename()` on other platforms.
30.15.3.15 fl_rmdir()

```c
int fl_rmdir (const char * f )
```

Cross-platform function to remove a directory with a UTF-8 encoded name.

This function is especially useful on the Windows platform where the standard _wrmdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

- **in** `f`: the UTF-8 encoded filename to remove

Returns

- the return value of _wrmdir() on Windows or rmdir() on other platforms.

30.15.3.16 fl_stat()

```c
int fl_stat (const char * f, struct stat * b )
```

Cross-platform function to stat() a file using a UTF-8 encoded name or value.

This function is especially useful on the Windows platform where the standard stat() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

- **in** `f`: the UTF-8 encoded filename
- **in** `b`: the stat struct to populate

Returns

- the return value of _wstat() on Windows or stat() on other platforms.

30.15.3.17 fl_system()

```c
int fl_system (const char * cmd )
```

Cross-platform function to run a system command with a UTF-8 encoded string.

This function is especially useful on the Windows platform where non-ASCII program (file) names must be encoded as wide characters.

On platforms other than Windows this function calls system() directly.

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>the UTF-8 encoded command string</th>
</tr>
</thead>
</table>

Returns

the return value of _wsystem() on Windows or system() on other platforms.

30.15.3.18 fl_ucs_to_Utf16()

unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short * dst, const unsigned dstlen)

Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.

These are used by some system calls, especially on Windows.

ucss 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.

30.15.3.19 fl_unlink()

int fl_unlink (const char * fname)

Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

This function is especially useful on the Windows platform where the standard function expects UTF-16 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>the filename to unlink</th>
</tr>
</thead>
</table>

Generated by Doxygen
Returns

the return value of _wunlink() on Windows or unlink() on other platforms.

30.15.3.20  fl_utf8back()

const char * fl_utf8back {
    const char * p,
    const char * start,
    const char * end)

Move p backward until it points to the start of a UTF-8 character.

If it already points at the start of one then it is returned unchanged. Any UTF-8 errors are treated as though each byte of the error is an individual character.

start is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.

end is the end of the string and is assumed to be a break between characters. It is assumed to be greater than p.

If you wish to decrement a UTF-8 pointer, pass p-1 to this.

30.15.3.21  fl_utf8bytes()

int fl_utf8bytes {
    unsigned ucs }

Return the number of bytes needed to encode the given UCS4 character in UTF-8.

Returns number of bytes that utf8encode() will use to encode the character ucs.

Parameters

| in  | ucs  | UCS4 encoded character |

Returns

number of bytes required

30.15.3.22  fl_utf8decode()

unsigned fl_utf8decode {
    const char * p,
    const char * end,
    int * len
Decode a single UTF-8 encoded character starting at \( p \).

The resulting Unicode value (in the range 0-0x10ffff) is returned, and \( \text{len} \) is set to the number of bytes in the UTF-8 encoding (adding \( \text{len} \) to \( p \) will point at the next character).

If \( p \) points at an illegal UTF-8 encoding, including one that would go past \( \text{end} \), or where a code uses more bytes than necessary, then \( *(\text{unsigned char}*)p \) is translated as though it is in the Microsoft CP1252 character set and \( \text{len} \) is set to 1. Treating errors this way allows this to decode almost any ISO-8859-1 or CP1252 text that has been mistakenly placed where UTF-8 is expected, and has proven very useful.

If you want errors to be converted to error characters (as the standards recommend), adding a test to see if the length is unexpectedly 1 will work:

```c
if (*p & 0x80) { // what should be a multibyte encoding
    code = fl_utf8decode(p,end,&len);
    if (len<2) code = 0xFFFD; // Turn errors into REPLACEMENT CHARACTER
    else { // handle the 1-byte UTF-8 encoding:
        code = *p;
        len = 1;
    }
}
```

Direct testing for the 1-byte case (as shown above) will also speed up the scanning of strings where the majority of characters are ASCII.

### 30.15.3.23 fl_utf8encode()

```c
int fl_utf8encode {
    unsigned ucs,
    char * buf }
```

Write the UTF-8 encoding of \( ucs \) into \( buf \) and return the number of bytes written.

Up to 4 bytes may be written. If you know that \( ucs \) is less than 0x10000 then at most 3 bytes will be written. If you wish to speed this up, remember that anything less than 0x80 is written as a single byte.

If \( ucs \) is greater than 0x10000 this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER).

RFC 3629 also says many other values for \( ucs \) are illegal (in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff). However I encode these as though they are legal, so that utf8encode/fl_utf8decode will be the identity for all codes between 0 and 0x10000.

### 30.15.3.24 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 \( \text{dstlen} \) bytes are written to \( \text{dst} \), 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 \( \text{dstlen} \) then if you malloc a new array of size \( n+1 \) you will have the space needed for the entire string. If \( \text{dstlen} \) 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 \( \text{fl_utf8test()} \) returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.
30.15.3.25  fl_utf8froma()

unsigned fl_utf8froma (  
    char * dst,  
    unsigned dstlen,  
    const char * src,  
    unsigned srclen )

Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

It is possible this should convert Microsoft's CP1252 to UTF-8 instead. This would translate the codes in the range 0x80-0x9f to different characters. Currently it does not do this.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

srclen is the number of bytes in src to convert.

If the return value equals srclen then this indicates that no conversion is necessary, as only ASCII characters are in the string.

30.15.3.26  fl_utf8fromwc()

unsigned fl_utf8fromwc (  
    char * dst,  
    unsigned dstlen,  
    const wchar_t * src,  
    unsigned srclen )

Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

srclen is the number of words in src to convert. On Windows this is not necessarily the number of characters, due to there possibly being "surrogate pairs" in the UTF-16 encoding used. On Unix wchar_t is 32 bits and each location is a character.

On Unix if a src word is greater than 0x10ffff then this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER). Characters in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff are also illegal according to RFC 3629. However I encode these as though they are legal, so that fl_utf8towc will return the original data.

On Windows "surrogate pairs" are converted to a single character and UTF-8 encoded (as 4 bytes). Mismatched halves of surrogate pairs are converted as though they are individual characters.
30.15.3.27 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.

30.15.3.28 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`

30.15.3.29 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`
30.15.3.30 fl_utf8locale()

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.

30.15.3.31 fl_utf8test()

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.

30.15.3.32 fl_utf8to_mb()

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.
30.15.3.33  fl_utf8toa()

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.

30.15.3.34  fl_utf8toUtf16()

unsigned fl_utf8toUtf16 (  
    const char * src,  
    unsigned srclen,  
    unsigned short * dst,  
    unsigned dstlen )

Convert a UTF-8 sequence into an array of 16-bit characters.

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).
30.15 Unicode and UTF-8 functions

30.15.3.35 fl_utf8towc()

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.

30.15.3.36 fl_utf_strcasecmp()

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | if the strings are equal  
| 1 | if s1 is greater than s2  
| -1 | if s1 is less than s2 |

30.15.3.37 fl_utf_strncasecmp()

int fl_utf_strncasecmp (  
    const char * s1,  
    const char * s2,  
    unsigned n )

UTF-8 aware strncasecmp - converts to Unicode and tests.
UTF-8 aware `strncasecmp` - converts to lower case Unicode and tests.

Parameters

| $s1,s2$ | the UTF-8 strings to compare |
| $n$     | the maximum number of UTF-8 characters to compare |

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 $s2$

30.15.3.38  `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 $3 \times len$ [for 16-bit Unicode]

30.15.3.39  `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 $3 \times len$ [for 16-bit Unicode]

30.15.3.40  `fl_wcwidth()`

```c
int fl_wcwidth (  
    const char * src )
```

extended wrapper around `fl_wcwidth_(unsigned int ucs)` function.
Parameters

in 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 fl_wcwidth(), and handle them separately.

30.15.3.41 fl_wcwidth()

int fl_wcwidth_ (unsigned int ucs)

Wrapper to adapt Markus Kuhn’s implementation of wcwidth() for FLTK.

Parameters

in ucs Unicode character value

Returns

width of character in columns

See http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c for Markus Kuhn’s original implementation of wcwidth() and wcswidth() (defined in IEEE Std 1002.1-2001) for Unicode.

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 fl_width(const char * src) instead.
30.16 Mac OS X-specific symbols

Mac OS X-specific symbols declared in `<FL/platform.H>`

Classes

- class Fl_Mac_App_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(*)(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, 101300 for 10.13).

30.16.1 Detailed Description

Mac OS X-specific symbols declared in `<FL/platform.H>`

See also

- The Apple OS X Interface

30.16.2 Function Documentation

30.16.2.1 fl_mac_set_about()

```
void fl_mac_set_about ( Fl_Callback * cb, 
                      void * user_data, 
                      int shortcut = 0 )
```

Attaches a callback to the "About myprog" item of the system application menu.

For back-compatibility. Equivalent to Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *user_data).
30.16.2.2 fl_open_callback()

```c
void fl_open_callback (  
    void(*) (const char *) cb  )
```

Register a function called for each file dropped onto an application icon.

This function is effective only on the Mac OS X platform. cb will be called with a single Unix-style file name and path. If multiple files were dropped, cb will be called multiple times.

This function should be called before fl_open_display() is called, either directly or indirectly (this happens at the first show() of a window), to be effective for files dropped on the application icon at launch time. It can also be called at any point to change the function used to open dropped files. A call with a NULL argument, after a previous call, makes the app ignore files dropped later.

30.16.3 Variable Documentation

30.16.3.1 fl_mac_os_version

```c
int fl_mac_os_version
```

The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13).

FLTK initializes this global variable before main() begins running. If the value is needed in a static initializer, a previous call to Fl::system_driver() makes sure fl_mac_os_version has been initialized.
30.17 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.

- **int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)**
  
  Pops up a window to let the user pick an arbitrary RGB color.

- **int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)**
  
  Pops up a window to let the user pick an arbitrary RGB color.

- **char * fl_dir_chooser (const char *message, const char *fname, int relative)**
  
  Shows a file chooser dialog and gets a directory.

- **char * fl_file_chooser (const char *message, const char *pat, const char *fname, int relative)**
  
  Shows a file chooser dialog and gets a filename.

- **void fl_file_chooser_callback (void((cb)(const char *))**
  
  Set the file chooser callback.

- **void fl_file_chooser_ok_label (const char *l)**
  
  Set the "OK" button label.

- **const char * fl_input (const char *fmt, const char *defstr,...)**
  
  Shows an input dialog displaying the fmt message.

- **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()
30.17 Common Dialogs classes and functions

Gets the preferred position for the common message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- void `fl_message_title` (const char *title)
  Sets the title of the dialog window used in many common dialogs.

- void `fl_message_title_default` (const char *title)
  Sets the default title of the dialog window used in many common dialogs.

- const char * `fl_password` (const char *fmt, const char *defstr, ...)
  Shows an input dialog displaying the `fmt` message.

Variables

- static void (*`Fl::error`) (const char *, ...) = `Fl_System_Driver::error`
  FLTK calls `Fl::error()` to output a normal error message.

- static void (*`Fl::fatal`) (const char *, ...) = `Fl_System_Driver::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 *, ...) = `Fl_System_Driver::warning`
  FLTK calls `Fl::warning()` to output a warning message.

30.17.1 Detailed Description

30.17.2 Function Documentation

30.17.2.1 `fl_alert()`

```c
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

| in | fmt | can be used as an sprintf-like format and variables for the message text |

30.17.2.2 fl_ask()

```c
int fl_ask (const char * fmt,
            ...
)
```

Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.

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

| in | fmt | can be used as an sprintf-like format and variables for the message text |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the no button is selected or another dialog box is still open</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if yes is selected</td>
</tr>
</tbody>
</table>

**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.

30.17.2.3 fl_beep()

```c
void fl_beep (int type )
```

Emits a system beep message.

Parameters

| in | type | The beep type from the Fl_Beep enumeration. |
30.17 Common Dialogs classes and functions

Note

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

30.17.2.4 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 which are specified in order of right-to-left in the dialog, e.g.

![Figure 30.5 fl_choice() button ordering](image)

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:

![Figure 30.6 fl_choice() three choices with printf formatting](image)

```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:

![Figure 30.7 fl_choice() three choices](image)
```c
switch ( fl_choice("How many bedrooms?", "Zero", "One", "Two") ) {
    case 0: .. // "Zero"
    case 1: .. // "One" (default)
    case 2: .. // "Two"
}
```

Two choice example:

![Figure 30.8 fl_choice() two choices](image)

```c
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
    case 0: .. // Yes
    case 1: .. // No (default)
}
```

One choice example:

![Figure 30.9 fl_choice() one choice](image)

```c
fl_choice("All hope is lost.", "OK", 0, 0); // "OK" default
```

### Parameters

| in | fmt | can be used as an sprintf-like format and variables for the message text |
| in | b0  | text label for right button 0 |
| in | b1  | text label for middle button 1 (can be 0) |
| in | b2  | text label for left button 2 (can be 0) |

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the button with b0 text is pushed or another dialog box is still open</td>
</tr>
<tr>
<td>1</td>
<td>if the button with b1 text is pushed</td>
</tr>
<tr>
<td>2</td>
<td>if the button with b2 text is pushed</td>
</tr>
</tbody>
</table>

### 30.17.2.5 fl_color_chooser() [1/2]

```c
int fl_color_chooser (  
    const char * name,  
    double & r,  
    double & g,  
    double & b,  
    int cmode ) [related]
```

Pops up a window to let the user pick an arbitrary RGB color.
30.17 Common Dialogs classes and functions

Note

#include <FL/Fl_Color_Chooser.H>

Figure 30.10 fl_color_chooser

Parameters

<table>
<thead>
<tr>
<th>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 mode(int). 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>

30.17.2.6 fl_color_chooser() [2/2]

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 30.11 fl_color_chooser](image)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>String</td>
<td>Title label for the window</td>
</tr>
<tr>
<td><code>r,g,b</code></td>
<td>int</td>
<td>Color components in the range 0 to 255.</td>
</tr>
<tr>
<td><code>cmode</code></td>
<td>int</td>
<td>Optional mode for color chooser. See <code>mode(int)</code>. Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

**Return values**

```
1  if user confirms the selection
0  if user cancels the dialog
```

### 30.17.2.7 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>
```
30.17 Common Dialogs classes and functions

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>message</th>
<th>title bar text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fname</td>
<td>initial/default directory name</td>
</tr>
<tr>
<td>in</td>
<td>relative</td>
<td>0 for absolute path return, relative otherwise</td>
</tr>
</tbody>
</table>

Returns

the directory path string chosen by the user or NULL if user cancels

30.17.2.8 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 30.12 Fl_File_Chooser
Parameters

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>text in title bar</td>
</tr>
<tr>
<td>pat</td>
<td>filename pattern filter</td>
</tr>
<tr>
<td>fname</td>
<td>initial/default filename selection</td>
</tr>
<tr>
<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

30.17.2.9 fl_file_chooser_callback()

```c
void fl_file_chooser_callback ( 
    void(*)(const char *) cb ) [related]
```

Set the file chooser callback.

Note

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

30.17.2.10 fl_file_chooser_ok_label()

```c
void fl_file_chooser_ok_label ( 
    const char * l ) [related]
```

Set the "OK" button label.

Note

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

30.17.2.11 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.

```
#include <FL/fl_ask.H>
```
30.17 Common Dialogs classes and functions

Parameters

| in  | fmt | can be used as an sprintf-like format and variables for the message text |
| in  | defstr | defines the default returned string if no text is entered |

Returns

the user string input if OK was pushed, NULL if Cancel was pushed or another dialog box was still open

30.17.2.12 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.

#include <FL/fl_ask.H>

Parameters

| in  | fmt | can be used as an sprintf-like format and variables for the message text |

30.17.2.13 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.

Note

#include <FL/fl_ask.H>

Parameters

| in  | enable | non-zero enables hotspot behavior, 0 disables hotspot |
30.17.2.14  fl_message_hotspot() [2/2]

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.

Note

   #include <FL/fl_ask.H>

Returns

   0 if disable, non-zero otherwise

See also

   fl_message_hotspot(int)

30.17.2.15  fl_message_icon()

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()

Note

   #include <FL/fl_ask.H>

30.17.2.16  fl_message_position() [1/3]

void fl_message_position (  
   const int x,  
   const int y,  
   const int center  )

Sets the preferred position for the common message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

Resets after every call to any of the common dialogs.

The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by fl_message_hotspot(int). If the optional argument center is non-zero (true) the message box will be centered at the given coordinates rather than using the X/Y position as the window position (top left corner).

Note

   #include <FL/fl_ask.H>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>Preferred X position</td>
</tr>
<tr>
<td>( y )</td>
<td>Preferred Y position</td>
</tr>
<tr>
<td>( \text{center} )</td>
<td>1 = centered, 0 = absolute</td>
</tr>
</tbody>
</table>

See also

```c
int fl_message_position(int \*x, int \*y)
```

### 30.17.2.17 `fl_message_position()` [2/3]

```c
void fl_message_position (
    Fl_Widget * widget )
```

Sets the preferred position for the common message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

The common message box will be centered over the given widget or window extensions.

Everything else is like `fl_message_position(int, int, int)` with argument 'center' set to 1.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{widget} )</td>
<td>Widget or window to position the message box over.</td>
</tr>
</tbody>
</table>

See also

```c
int fl_message_position(int x, int y, int center)
```

### 30.17.2.18 `fl_message_position()` [3/3]

```c
int fl_message_position ( 
    int \* x, 
    int \* y )
```

Gets the preferred position for the common message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

Note

```c
#include <FL/fl_ask.H>
```
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x</th>
<th>Preferred X position, returns -1 if not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>Preferred Y position, returns -1 if not set</td>
</tr>
</tbody>
</table>

Returns

whether position is currently set or not

Return values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>position is not set (may be hotspot or not)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>position is set (window position)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>position is set (message box centered)</td>
<td></td>
</tr>
</tbody>
</table>

See also

- `fl_message_position(int, int)`
- `fl_message_hotspot(int)`
- `int fl_message_hotspot()`

30.17.2.19   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.

Note

```c
#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>
30.17.2.20 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**

- **in title** default window label, string copied internally

---

30.17.2.21 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**

- **in fmt** can be used as an sprintf-like format and variables for the message text
- **in defstr** defines the default returned string if no text is entered

**Returns**

the user string input if OK was pushed, NULL if Cancel was pushed or another dialog box was still open
30.17.3 Variable Documentation

30.17.3.1 error

```c
void(* Fl::error)(const char *format,...) = Fl_System_Driver::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.
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>
```

30.17.3.2 fatal

```c
void(* Fl::fatal)(const char *format,...) = Fl_System_Driver::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>
```

30.17.3.3 warning

```c
void(* Fl::warning)(const char *format,...) = Fl_System_Driver::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>
```
30.18 File names and URI utility functions

File names and URI functions defined in `<FL/filename.H>`

Macros

- `#define FL_PATH_MAX 2048`
  
  *all path buffers should use this length*

Typedefs

- `typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)`
  
  *File sorting function.*

Functions

- `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 extension 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).*

30.18.1 Detailed Description

File names and URI functions defined in `<FL/filename.H>`
30.18.2 Typedef Documentation

30.18.2.1 Fl_File_Sort_F

typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

File sorting function.

See also

fl_filename_list()

30.18.3 Function Documentation

30.18.3.1 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.

30.18.3.2 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"

fl_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="/\foo.txt"
fl_filename_absolute(out, sizeof(out), "/\log/messages");  // out="/\log/messages"

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>resulting absolute filename</td>
</tr>
<tr>
<td>to</td>
<td>relative filename</td>
</tr>
<tr>
<td>tolen</td>
<td>size of the absolute filename buffer</td>
</tr>
<tr>
<td>from</td>
<td>relative filename</td>
</tr>
</tbody>
</table>
30.18.3.3  fl_filenameexpand()


def fl_filenameexpand ( 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_filenameexpand(out, sizeof(out), "~fred/.cshrc"); // out="/usr/fred/.cshrc"
fl_filenameexpand(out, sizeof(out), "~/file"); // out="/usr/<yourname>/file"
fl_filenameexpand(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

30.18.3.4  fl_filenameext()


def fl_filenameext ( const char * buf )

Gets the extension of a filename.

```c
#include <FL/filename.H>
[..
const char * out;
out = fl_filenameext("/some/path/foo.txt"); // result: "txt"
out = fl_filenameext("/some/path/foo"); // result: NULL
```

Generated by Doxygen
### 30.18.3.5 fl_filename_free_list()

```c
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>

### 30.18.3.6 fl_filename_isdir()

```c
FL_EXPORT int fl_filename_isdir (  
    const char * n )
```

Determines if a file exists and is a directory from its filename.

```c
#include <FL/filename.H>
[..]
fl_filename_isdir("/etc");  // returns non-zero
fl_filename_isdir("/etc/hosts");  // returns 0
```

**Parameters**

<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
30.18.3.7 fl_filename_list()

```c
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 nul-terminated file name. A pointers array to these dirent's is created and a pointer to the array is returned in ∗list. The number of entries is given as a return value. If there is an error reading the directory a number less than zero is returned, and errno has the reason; errno does not work under Windows.

**Include:**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>d</th>
<th>the name of the directory to list. It does not matter if it has a trailing slash.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>list</td>
<td>table containing the resulting directory listing</td>
</tr>
<tr>
<td>in</td>
<td>sort</td>
<td>sorting functor:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_alphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared according to their ASCII ordering uppercase before lowercase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casealphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared equally case is not significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casenumericsort: 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>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_numericsort: 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>
</tr>
</tbody>
</table>

**Returns**

the number of entries if no error, a negative value otherwise.

30.18.3.8 fl_filename_match()

```c
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.
• [^set] or [!set] matches any character not in the set.
• {X\|Y\|Z} or {X,Y,Z} matches any one of the subexpressions literally.
• \x quotes the character x so it has no special meaning.
• x all other characters 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>in</td>
<td>p</td>
<td>the string pattern</td>
</tr>
</tbody>
</table>

Returns

non zero if the string matches the pattern

30.18.3.9  fl_filename_name()

```
FL_EXPORT const char * fl_filename_name ( const char * filename )
```

Gets the file name from a path.

Similar to basename(3), exceptions shown below.

```
#include <FL/filename.H>
[..]
const char *out;
out = fl_filename_name("/usr/lib"); // out="lib"  (basename(3) returns "usr" instead)
out = fl_filename_name("/usr/");    // out=""     (basename(3) returns "/" instead)
out = fl_filename_name("/usr");     // out="usr"  (basename(3) returns "/" instead)
out = fl_filename_name(".");        // out="."    (basename(3) returns "." instead)
out = fl_filename_name("..");       // out=".."   (basename(3) returns "." instead)
```

Returns

a pointer to the char after the last slash, or to filename if there is none.
30.18.3.10  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.

#include <FL/filename.H>
[..]
fl_chdir("/var/tmp/somedir"); // set cwd to /var/tmp/somedir
[..]
char out[FL_PATH_MAX];
fl_filename_relative(out, sizeof(out), "/var/tmp/somedir/foo.txt");  // out="foo.txt", return=1
fl_filename_relative(out, sizeof(out), "/var/tmp/foo.txt");         // out="..foo.txt", return=1
fl_filename_relative(out, sizeof(out), "foo.txt");                // out="foo.txt", return=0 (no change)
fl_filename_relative(out, sizeof(out), ".foo.txt");               // out="..foo.txt", return=0 (no change)

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

30.18.3.11  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

#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.
30.18.3.12 \texttt{fl\_open\_uri()}

```c
int fl_open_uri (  
  const char * uri,  
  char * msg,  
  int msglen )
```

Opens the specified Uniform Resource Identifier (URI).

Uses an operating-system dependent program or interface. For URIs using the "ftp", "http", or "https" schemes, the system default web browser is used to open the URI, while "mailto" and "news" URIs are typically opened using the system default mail reader and "file" URIs are opened using the file system navigator.

On success, the (optional) msg buffer is filled with the command that was run to open the URI; on Windows, this will always be "open uri".

On failure, the msg buffer is filled with an English error message.

Note

**Platform Specific Issues: Windows**

With "file:" based URIs on Windows, you may encounter issues with anchors being ignored. Example: "file:///c:/some/index.html#anchor" may open in the browser without the "#anchor" suffix. The behavior seems to vary across different Windows versions. Workaround: open a link to a separate html file that redirects to the desired "file:" URI.

**Example**

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

char errmsg[512];
if ( !fl_open_uri("http://google.com/", errmsg, sizeof(errmsg)) ) {  
  char warnmsg[768];  
  sprintf(warnmsg, "Error: %s", errmsg);  
  fl_alert(warnmsg);  
}
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{uri}</td>
<td>The URI to open</td>
</tr>
<tr>
<td>\textit{msg}</td>
<td>Optional buffer which contains the command or error message</td>
</tr>
<tr>
<td>\textit{msglen}</td>
<td>Length of optional buffer</td>
</tr>
</tbody>
</table>

**Returns**

1 on success, 0 on failure
Chapter 31

Class Documentation

31.1 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

31.2 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_TOOLTIP, OPTION_FNFC_USES_GTK, OPTION_PRINTER_USES_GTK, OPTION_SHOW_SCALING, OPTION_LAS
  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, int when, Fl_FD_Handler cb, void **=0)
  Adds file descriptor fd to listen to.
- static void add_fd (int fd, 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 int args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)
  Parse command line switches using the cb argument handler.
- static void args (int argc, char **argv)
  Parse all command line switches matching standard FLTK options only.
- 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 int awake (Fl_Awake_Handler cb, void *message=0)
  See void awake(void* message=0).
- 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 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 bool cairo_autolink_context ()
  Gets the current autolink mode for cairo support.
• 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 void damage (int d)
  If true then flush() will do something.
• static int damage ()
  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 *)
  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 void dnd_text_ops (int v)
  Sets whether drag and drop text operations are supported.
• static int dnd_text_ops ()
  Gets 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** `draw_GL_text_with_textures (int val)`
  
  sets whether OpenGL uses textures to draw all text.

- **static int** `draw_GL_text_with_textures ()`
  
  returns whether OpenGL uses textures to draw all text.

- **static void** `enable_im ()`
  
  Enables the system input methods facilities.

- **static int** `event ()`
  
  Returns the last event that was processed.

- **static int** `event_alt ()`
  
  Returns non-zero if the Alt key is pressed.

- **static int** `event_button ()`
  
  Gets which particular mouse button caused the current event.

- **static int** `event_button1 ()`
  
  Returns non-zero if mouse button 1 is currently held down.

- **static int** `event_button2 ()`
  
  Returns non-zero if button 2 is currently held down.

- **static int** `event_button3 ()`
  
  Returns non-zero if button 3 is currently held down.

- **static int** `event_buttons ()`
  
  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.

- **static int** `event_clicks ()`
  
  Returns non zero if we had a double click event.

- **static void** `event_clicks (int i)`
  
  Manually sets the number returned by Fl::event_clicks().

- **static void** `* event_clipboard ()`

  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.

- **static const char** `* event_clipboard_type ()`

  Returns the type of the pasted data during an FL_PASTE event.

- **static int** `event_command ()`
  
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.

- **static int** `event_ctrl ()`
  
  Returns non-zero if the Control key is pressed.

- **static void** `event_dispatch (Fl_Event_Dispatch d)`

  Set a new event dispatch function.

- **static Fl_Event_Dispatch** `event_dispatch ()`

  Return the current 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 (int, int, int, int)`

  Returns whether or not the mouse event is inside the given rectangle.

- **static int** `event_inside (const Fl_Widget *)`

  Returns whether or not the mouse event is inside a given child widget.

- **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 ()`
31.2 Fl Class Reference

371

Gets which key on the keyboard was last pushed.

- static int event_key (int key)
  Returns true if the given key was held down (or pressed) during the last event.

- static int event_length ()
  Returns the length of the text in Fl::event_text().

- static int event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.

- static int event_shift ()
  Returns non-zero if the Shift key is pressed.

- static int event_state ()
  Returns the keyboard and mouse button states of the last event.

- static int event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.

- static const char * event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.

- static int event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

- static int event_x_root ()
  Returns the mouse position on the screen of the event.

- static int event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

- static int event_y_root ()
  Returns the mouse position on the screen of the event.

- static Fl_Window * first_window ()
  Returns the first top-level window in the list of shown() windows.

- static void first_window (Fl_Window *)
  Sets the window that is returned by first_window().

- static void flush ()
  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.

- static Fl_Widget * focus ()
  Gets the current Fl::focus() widget.

- static void focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.

- static void foreground (uchar, uchar, uchar)
  Changes fl_color(FL_FOREGROUND_COLOR).

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

- static int get_awake_handler_ (Fl_Awake_Handler &, void * &)
  Gets the last stored awake handler for use in awake().

- static Fl_Box_Draw_F * get_boxtype (Fl_Boxtype)
  Gets the current box drawing function for the specified box type.

- static unsigned get_color (Fl_Color i)
  Returns the RGB value(s) for the given FLTK color index.

- static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)
  Returns the RGB value(s) for the given FLTK color index.

- static const char * get_font (Fl_Font)
  Gets the string for this face.

- static const char * get_font_name (Fl_Font, int *attributes=0)
  Gets 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 *)
  Selects the window to grab.

• static void grab (Fl_Window &win)
  See grab(Fl_Window *)

• static int h ()
  Returns the height in pixels of the main screen work area.

• static int handle (int, Fl_Window *)
  Handle events from the window system.

• static int handle_ (int, Fl_Window *)
  Handle events from the window system.

• static int has_check (Fl_Timeout_Handler, void * = 0)
  Returns 1 if the check exists and has not been called yet, 0 otherwise.

• static int has_idle (Fl_Idle_Handler cb, void * data=0)
  Returns true if the specified idle callback is currently installed.

• static int has_timeout (Fl_Timeout_Handler, void * = 0)
  Returns true if the timeout exists and has not been called yet.

• static void insertion_point_location (int x, int y, int height)
  Sets window coordinates and height of insertion point.

• static int is_scheme (const char * name)
  Returns whether the current scheme is the given name.

• static void keyboard_screen_scaling (int value)
  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.

• static int lock ()
  The lock() method blocks the current thread until it can safely access FLTK widgets and data.

• static int menu_linespacing ()
  Gets the default line spacing used by menus.

• static void menu_linespacing (int H)
  Sets the default line spacing used by menus.

• static Fl_Window * modal ()
  Returns the top-most modal() window currently shown.

• static Fl_Window * next_window (const Fl_Window *)
  Returns the next top-level window in the list of shown() windows.

• static bool option (Fl_Option opt)
  FLTK library options management.

• static void option (Fl_Option opt, bool val)
  Override an option while the application is running.

• static void own_colormap ()
  Makes FLTK use its own colormap.

• static void paste (Fl_Widget &receiver, 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 **paste** (Fl_Widget &receiver)
  
  *Backward compatibility only.*

• static int **program** _**should**_ **quit** ()

  *Returns non-zero when a request for program termination was received and accepted.*

• static void **program** _**should**_ **quit** (int should_i)

  *Indicate to the FLTK library whether a program termination request was received and accepted.*

• static Fl_Widget * **pushed** ()

  *Gets the widget that is being pushed.*

• static void **pushed** (Fl_Widget *)

  *Sets the widget that is being pushed.*

• static Fl_Widget * **readqueue** ()

  *Reads the default callback queue and returns the first widget.*

• static int **ready** ()

  *This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.*

• static void **redraw** ()

  *Redraws all widgets.*

• static void **release** ()

  *Releases the current grabbed window, equals grab(0).*

• static void **release** _**widget**_ **pointer** (Fl_Widget *&w)

  *Releases a widget pointer from the watch list.*

• static int **reload** _**scheme**_ ()

  *Called by scheme according to scheme name.*

• static void **remove** _**check**_ (Fl_Timeout_Handler, void *=0)

  *Removes a check callback.*

• static void **remove** _**clipboard**_ **notify** (Fl_Clipboard_Notify_Handler h)

  *Stop calling the specified callback when there are changes to the selection buffer or the clipboard.*

• static void **remove** _fd_ (int, int when)

  *Removes a file descriptor handler.*

• static void **remove** _fd_ (int)

  *Removes a file descriptor handler.*

• static void **remove** _**handler**_ (Fl_Event_Handler h)

  *Removes a previously added event handler.*

• static void **remove** _**idle**_ (Fl_Idle_Handler cb, void +data=0)

  *Removes the specified idle callback, if it is installed.*

• static void **remove** _**system**_ **handler** (Fl_System_Handler h)

  *Removes a previously added system event handler.*

• static void **remove** _timeout_ (Fl_Timeout_Handler, void *=0)

  *Removes a timeout callback.*

• static void **repeat** _**timeout**_ (double t, Fl_Timeout_Handler, void *=0)

  *Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.*

• static void **reset** _marked**_ **text** ()

  *Resets marked text.*

• static int **run** ()

  *Calls Fl::wait() repeatedly as long as any windows are displayed.*

• static void **run** _checks_ ()

• static int **scheme** (const char *name)

  *Sets the current widget scheme.*

• static const char * **scheme** ()

  *See void scheme(const char *name)*

• 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 Fl_Screen_Driver * `screen_driver`
  Returns a pointer to the unique Fl_Screen_Driver object of the platform.

- static int `screen_num` (int x, int y)
  Gets the screen number of a screen that contains the specified screen position x, y.

- static int `screen_num` (int x, int y, int w, int h)
  Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.

- static float `screen_scale` (int n)
  Current value of the GUI scaling factor for screen number n.

- static void `screen_scale` (int n, float factor)
  Set the value of the GUI scaling factor for screen number n.

- static int `screen_scaling_supported`
  See if scaling factors are supported by this platform.

- static void `screen_work_area` (int &X, int &Y, int &W, int &H, 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_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_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 n)
  Gets the screen bounding rect for the given screen.

- 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 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 *text, int len)
  Changes the current selection.

- static Fl_Widget * `selection_owner`
  back-compatibility only: Gets the widget owning the current selection

- static void `selection_owner` (Fl_Widget *)
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.

- static 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, uchar, uchar, uchar)
Sets an entry in the fl_color index table.

- static void set_color (FL_Color i, unsigned c)
  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)
  FTK 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 FL_System_Driver * system_driver ()
  Returns a pointer to the unique FL_System_Driver object of the platform.

- static int test_shortcut (FL_Shortcut)
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in FL_Button).

- static void * thread_message ()
  The thread_message() method returns the last message that was sent from a child by the awake() method.

- static void unlock ()
  The unlock() method releases the lock that was set using the lock() method.

- 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 int use_high_res_GL ()
  Returns 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 void visible_focus (int v)
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- static int visible_focus ()
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- static int visual (int)
  Selects a visual so that your graphics are drawn correctly.

- static int w ()
  Returns the width in pixels of the main screen work area.

- static int wait ()
  Waits until "something happens" and then returns.

- static double wait (double time)
  Waits a maximum of time_to_wait seconds or until "something happens".

- static void watch_widget_pointer (FL_Widget * &w)
  Adds a widget pointer to the widget watch list.

- static int x ()
  Returns the leftmost x coordinate of the main screen work area.

- static int y ()
  Returns the topmost y coordinate of the main screen work area.
Static Public Attributes

- static void(atclose)(Fl_Window *, void *)
  Back compatibility: default window callback handler.
- static bool cfg_gfx_cairo = 0
  Cairo rendering available, available on many platforms.
- static bool cfg_gfx_directx = 0
  DirectX rendering available, usually on Windows systems.
- static bool cfg_gfx_gdi = 0
  GDI rendering available, usually on Windows systems.
- static bool cfg_gfx_opengl = 0
  OpenGL rendering available, available on many platforms.
- static bool cfg_gfx_quartz = 0
  Quartz rendering available, usually on OS X systems.
- static bool cfg_gfx_xlib = 0
  X11 Xlib rendering available, usually on Linux systems.
- static bool cfg_prn_gdi = 0
  GDI rendering available, usually on Windows systems.
- static bool cfg_prn_ps = 0
  PostScript rendering available, usually on Linux systems.
- static bool cfg_prn_quartz = 0
  Quartz rendering available, usually on OS X systems.
- static bool cfg_sys_posix = 0
  Posix system available, usually on Linux and OS X systems, but also Cygwin.
- static bool cfg_sys_win32 = 0
  Windows system available, on Windows.
- static bool cfg_win_cocoa = 0
  Cocoa window management available, usually on OS X systems.
- static bool cfg_win_x11 = 0
  X11 window management available, usually on Linux systems.
- 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(idle)()
  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
- static void(error)(const char *,...) = Fl_System_Driver::error
  FLTK calls Fl::error() to output a normal error message.
- static void(fatal)(const char *,...) = Fl_System_Driver::fatal
  FLTK calls Fl::fatal() to output a fatal error message.
- static const char *const help = helpmsg+13
  Usage string displayed if Fl::args() detects an invalid argument.
- static void(idle)()
  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
- static void(warning)(const char *,...) = Fl_System_Driver::warning
  FLTK calls Fl::warning() to output a warning message.

31.2.1 Detailed Description

The Fl is the FLTK global (static) class containing state information and global methods for the current application.
31.2.2 Member Enumeration Documentation

31.2.2.1 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)</td>
</tr>
<tr>
<td></td>
<td>When switched off (default), the cursor will stop at the end of the text. Pressing Tab or Ctrl-Tab will advance the keyboard focus.</td>
</tr>
<tr>
<td></td>
<td>See also: Fl_Input_::tab_nav()</td>
</tr>
<tr>
<td>OPTION_VISIBLE_FOCUS</td>
<td>If visible focus is switched on (default), FLTK will draw a dotted rectangle 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_TOOLTIPS</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_PRINTER_USES_GTK</td>
<td>When switched on (default), Fl_Printer runs the GTK printer dialog if the GTK library is available on the platform (linux/unix only). When switched off, the GTK printer dialog isn't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_SHOW_SCALING</td>
<td>When switched on (default), the library shows in a transient yellow window the zoom factor value. When switched off, no such window gets displayed.</td>
</tr>
<tr>
<td>OPTION_LAST</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

31.2.3 Member Function Documentation
31.2.3.1 abi_check()

static int Fl::abi_check (  
    const int \texttt{val} = \texttt{FL_ABI_VERSION} )  \texttt{[inline]}, \texttt{[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 \texttt{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 \texttt{val} argument is \texttt{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()

31.2.3.2 abi_version()

int Fl::abi_version ( )  \texttt{[static]}

Returns the compiled-in value of the FL_ABI_VERSION constant.

This is useful for checking the version of a shared library.

31.2.3.3 add_awake_handler()

int Fl::add_awake_handler_ (  
    \texttt{Fl\_Awake\_Handler} func,  
    \texttt{void} * data )  \texttt{[static]}

Adds an awake handler for use in awake().
31.2 Fl Class Reference

31.2.3.4 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();
}
```

31.2.3.5 add_fd() [1/2]

```cpp
void Fl::add_fd (  
    int fd,  
    int when,  
    Fl_FD_Handler cb,  
    void * d = 0 ) [static]
```

Adds file descriptor fd to listen to.

When the fd becomes ready for reading Fl::wait() will call the callback and then return. The callback is passed the fd and the arbitrary void* argument.

This version takes a when bitfield, with the bits FL_READ, FL_WRITE, and FL_EXCEPT defined, to indicate when the callback should be done.

There can only be one callback of each type for a file descriptor. Fl::remove_fd() gets rid of all the callbacks for a given file descriptor.

Under UNIX/Linux/MacOS any file descriptor can be monitored (files, devices, pipes, sockets, etc.). Due to limitations in Microsoft Windows, Windows applications can only monitor sockets.
31.2.3.6 add_fd() [2/2]

```c
void Fl::add_fd (  
    int fd,  
    Fl_FD_Handler cb,  
    void * d = 0 )  [static]
```

Adds file descriptor fd to listen to.

See Fl::add_fd(int fd, int when, Fl_FD_Handler cb, void* = 0) for details.

31.2.3.7 add_idle()

```c
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.

31.2.3.8 add_timeout()

```c
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:

```c
#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();  
}
```
31.2.3.9 api_version()

```
int Fl::api_version ( ) [static]
```

Returns the compiled-in value of the FL_API_VERSION constant.
This is useful for checking the version of a shared library.

31.2.3.10 arg()

```
int Fl::arg ( int argc, char ** argv, int & i ) [static]
```

Parse a single switch from argv, starting at word i.

Returns the number of words eaten (1 or 2, or 0 if it is not recognized) and adds the same value to i.

This is the default argument handler used internally by Fl::args(...), but you can use this function if you prefer to step through the standard FLTK switches yourself.

All standard FLTK switches except -bg2 may be abbreviated to just one letter and case is ignored:

- `-bg color` or `-background color`
  Sets the background color using Fl::background().

- `-bg2 color` or `-background2 color`
  Sets the secondary background color using Fl::background2().

- `-display host:n.n`
  Sets the X display to use; this option is silently ignored under Windows and MacOS.

- `-dnd` and `-nodnd`
  Enables or disables drag and drop text operations using Fl::dnd_text_ops().

- `-fg color` or `-foreground color`
  Sets the foreground color using Fl::foreground().

- `-geometry WxH+X+Y`
  Sets the initial window position and size according to the standard X geometry string.

- `-iconic`
  Iconifies the window using Fl_Window::iconize().

- `-kbd` and `-nokbd`
  Enables or disables visible keyboard focus for non-text widgets using Fl::visible_focus().

- `-name string`
  Sets the window class using Fl_Window::xclass().

- `-scheme string`
  Sets the widget scheme using Fl::scheme().

- `-title string`
  Sets the window title using Fl_Window::label().

- `-tooltips` and `-notooltips`
  Enables or disables tooltips using Fl_Tooltip::enable().

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.
31.2.3.11  args() [1/2]

```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 cb lets you define your own switches. It is called with the same argc and argv, and with i set to the index of the switch to be processed. The cb handler should return zero if the switch is unrecognized, and not change i. It should return non-zero to indicate the number of words processed if the switch is recognized, i.e. 1 for just the switch, and more than 1 for the switch plus associated parameters. i should be incremented by the same amount.

The cb handler is called before any other tests, so you can also override any standard FLTK switch (this is why FLTK can use very short switches instead of the long ones all other toolkits force you to use). See `Fl::arg()` for descriptions of the standard switches.

On return i is set to the index of the first non-switch. This is either:

- The first word that does not start with '-'.
- The word '-' (used by many programs to name stdin as a file)
- The first unrecognized switch (return value is 0).
- argc

The return value is i unless an unrecognized switch is found, in which case it is zero. If your program takes no arguments other than switches you should produce an error if the return value is less than argc.

A usage string is displayed if `Fl::args()` detects an invalid argument on the command-line. You can change the message by setting the `Fl::help` pointer.

A very simple command line parser can be found in `examples/howto-parse-args.cxx`

The simpler `Fl::args(int argc, char **argv)` form is useful if your program does not have command line switches of its own.

31.2.3.12  args() [2/2]

```c
void Fl::args ( int argc,
                char ** argv ) [static]
```

Parse all command line switches matching standard FLTK options only.

It parses all the switches, and if any are not recognized it calls `Fl::abort(Fl::help)`, i.e. unlike the long form, an unrecognized switch generates an error message and causes the program to exit.
31.2.3.13  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.

31.2.3.14  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 Fl_Input and other text widgets.

This call may change fl_color(FL_FOREGROUND_COLOR) if it does not provide sufficient contrast to FL_BACKGROUN2_COLOR.

31.2.3.15  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 c otherwise.

31.2.3.16  box_dh()

```cpp
int Fl::box_dh (Fl_Boxtype t) [static]
```

Returns the height offset for the given boxtype.

See also

   box_dy().
31.2.3.17 box_dw()

```cpp
int Fl::box_dw ( 
    Fl_Boxtype t ) [static]
```

Returns the width offset for the given boxtype.

See also

`box_dy()`.

31.2.3.18 box_dx()

```cpp
int Fl::box_dx ( 
    Fl_Boxtype t ) [static]
```

Returns the X offset for the given boxtype.

See also

`box_dy()`

31.2.3.19 box_dy()

```cpp
int Fl::box_dy ( 
    Fl_Boxtype t ) [static]
```

Returns the Y offset for the given boxtype.

These functions return the offset values necessary for a given boxtype, useful for computing the area inside a box’s borders, to prevent 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():

```cpp
int X = yourwidget->x() + Fl::box_dx(yourwidget->box());
int Y = yourwidget->y() + Fl::box_dy(yourwidget->box());
int W = yourwidget->w() - Fl::box_dw(yourwidget->box());
int H = yourwidget->h() - Fl::box_dh(yourwidget->box());
```

These functions are mainly useful in the draw() code for deriving custom widgets, where one wants to avoid drawing over the widget’s own border box().
31.2.3.20 check()

int Fl::check ( ) [static]

Same as Fl::wait(0).

Calling this during a big calculation will keep the screen up to date and the interface responsive:

```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).

31.2.3.21 damage()

static int Fl::damage ( ) [inline], [static]

If true then flush() will do something.

31.2.3.22 display()

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.

This does nothing on other platforms.

31.2.3.23 dnd_text_ops()[1/2]

static void Fl::dnd_text_ops (int v ) [inline], [static]

Sets whether drag and drop text operations are supported.

This specifically affects whether selected text can be dragged from text fields or dragged within a text field as a cut/paste shortcut.

31.2.3.24 dnd_text_ops()[2/2]

static int Fl::dnd_text_ops ( ) [inline], [static]

Gets whether drag and drop text operations are supported.

This returns whether selected text can be dragged from text fields or dragged within a text field as a cut/paste shortcut.
31.2.3.25 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

```cpp
Fl::box_color(Fl_Color c)
```

31.2.3.26 draw_GL_text_with_textures() [1/2]

```cpp
static void Fl::draw_GL_text_with_textures ( int val ) [inline], [static]
```

sets whether OpenGL uses textures to draw all text.

By default, FLTK draws OpenGL text using textures, if the necessary hardware support is available. Call `Fl::draw_GL_text_with_textures(0)` once in your program before the first call to `gl_font()` to have FLTK draw instead OpenGL text using a legacy, platform-dependent procedure. It’s recommended not to deactivate textures under the MacOS platform because the MacOS legacy procedure is extremely rudimentary.

Parameters

| val | use 0 to prevent FLTK from drawing GL text with textures |

See also

```cpp
gl_texture_pile_height(int max)
```

Version

1.4.0

31.2.3.27 draw_GL_text_with_textures() [2/2]

```cpp
static int Fl::draw_GL_text_with_textures ( ) [inline], [static]
```

returns whether OpenGL uses textures to draw all text.

Default is yes.

See also

```cpp
draw_GL_text_with_textures(int val)
```

Version

1.4.0
31.2.3.28 flush()

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

Causes all the windows that need it to be redrawn and graphics forced out through the pipes.

This is what `wait()` does before looking for events.

Note: in multi-threaded applications you should only call `Fl::flush()` from the main thread. If a child thread needs to trigger a redraw event, it should instead call `Fl::awake()` to get the main thread to process the event queue.

31.2.3.29 foreground()

`void Fl::foreground ( uchar r, uchar g, uchar b ) [static]`

Changes `fl_color(FL_FOREGROUND_COLOR)`.

31.2.3.30 get_awake_handler()

`int Fl::get_awake_handler_ ( Fl_Awake_Handler & func, void ** & data ) [static]`

Gets the last stored awake handler for use in `awake()`.

31.2.3.31 get_boxtype()

`Fl_Box_Draw_F * Fl::get_boxtype ( Fl_Boxtype t ) [static]`

Gets the current box drawing function for the specified box type.

31.2.3.32 get_system_colors()

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

Read the user preference colors from the system and use them to call `Fl::foreground()`, `Fl::background()`, and `Fl::background2()`.

This is done by `Fl_Window::show(argc,argv)` before applying the `-fg` and `-bg` switches.

On X this reads some common values from the Xdefaults database. KDE users can set these values by running the “krdb” program, and newer versions of KDE set this automatically if you check the “apply style to other X programs” switch in their control panel.
31.2.3.33  gl_visual()

static int Fl::gl_visual (  
  int ,  
  int * alist = 0 ) [static]

This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.

This must be done if you want to draw in normal windows with OpenGL with gl_start() and gl_end(). It may be useful to call this so your X windows use the same visual as an Fl_Gl_Window, which on some servers will reduce colormap flashing.

See Fl_Gl_Window for a list of additional values for the argument.

31.2.3.34  insertion_point_location()

void Fl::insertion_point_location (  
  int x,  
  int y,  
  int height ) [static]

Sets window coordinates and height of insertion point.

See also
  Fl::compose(int& del) for a detailed description.

31.2.3.35  is_scheme()

static int Fl::is_scheme (  
  const char * name ) [inline], [static]

Returns whether the current scheme is the given name.

This is a fast inline convenience function to support scheme-specific code in widgets, e.g. in their draw() methods, if required.

Use a valid scheme name, not NULL (although NULL is allowed, this is not a useful argument - see below).

If Fl::scheme() has not been set or has been set to the default scheme ("none" or "base"), then this will always return 0 regardless of the argument, because Fl::scheme() is NULL in this case.

Note
  The stored scheme name is always lowercase, and this method will do a case-sensitive compare, so you must provide a lowercase string to return the correct value. This is intentional for performance reasons.

Example:

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)`

31.2.3.36 menu_linespacing()

`int Fl::menu_linespacing ( ) [static]`

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

31.2.3.37 menu_linespacing()

`void Fl::menu_linespacing ( int H ) [static]`

Sets the default line spacing used by menus.

Default is 4.

Parameters

| in | H | The new default line spacing between menu items, in pixels. |

31.2.3.38 option()

`bool Fl::option ( Fl_Option opt ) [static]`

FLTK library options management.

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>opt</th>
<th>which option</th>
</tr>
</thead>
</table>

### Returns

true or false

### See also

```c
enum Fl::Fl_Option
  Fl::option(Fl::Option, bool)
```

### Since

FLTK 1.3.0

#### 31.2.3.39 option() [2/2]

```c
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:

```c
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)
```

### 31.2.3.40 own_colormap()

```cpp
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.

### 31.2.3.41 program_should_quit() [1/2]

```cpp
static int Fl::program_should_quit ( ) [inline], [static]
```

Returns non-zero when a request for program termination was received and accepted.

On the MacOS platform, the "Quit xxx" item of the application menu is such a request, that is considered accepted when all windows are closed. On other platforms, this function returns 0 until `Fl::program_should_quit(1)` is called.

Version

1.4.0

### 31.2.3.42 program_should_quit() [2/2]

```cpp
static void Fl::program_should_quit ( int should_i ) [inline], [static]
```

Indicate to the FLTK library whether a program termination request was received and accepted.

A program may set this to 1, for example, while performing a platform-independent command asking the program to cleanly terminate, similarly to the "Quit xxx" item of the application menu under MacOS.

Version

1.4.0
### 31.2.3.43 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 * cb, void * data)` or one of its variants.

See also

- `FL_Widget::callback()`
- `FL_Widget::callback(Fl_Callback * cb, void * data)`
- `FL_Widget::default_callback()`

### 31.2.3.44 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()`).

```cpp
while (!calculation_done()) {
    calculate();
    if (Fl::ready()) {
        do_expensive_cleanup();
        Fl::check();
        if (user_hit_abort_button()) break;
    }
}
```

### 31.2.3.45 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 *)`
31.2.3.46 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)

31.2.3.47 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.

31.2.3.48 remove_fd() [1/2]

void Fl::remove_fd ( int fd,
                    int when ) [static]

Removes a file descriptor handler.

31.2.3.49 remove_fd() [2/2]

void Fl::remove_fd ( int fd ) [static]

Removes a file descriptor handler.

31.2.3.50 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.
31.2.3.51 repeat_timeout()

```c
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 of the same timer or at least a closely related timer, otherwise the timing accuracy can't be improved and the behavior is undefined.

The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```c
void callback(void*) {
    puts("TICK");
    Fl::repeat_timeout(1.0, callback);
}

int main() {
    Fl::add_timeout(1.0, callback);
    return Fl::run();
}
```

31.2.3.52 reset_marked_text()

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

Resets marked text.

In many languages, typing a character can involve multiple keystrokes. For example, the Ä can be composed of two dots (¨) on top of the character, followed by the letter A (on a Mac with U.S. keyboard, you’d type Alt-U, Shift-A. To inform the user that the dots may be followed by another character, the ¨ is underlined).

Call this function if character composition needs to be aborted for some reason. One such example would be the text input widget losing focus.

31.2.3.53 run()

```c
int Fl::run () [static]
```

Calls Fl::wait() repeatedly as long as any windows are displayed.

When all the windows are closed it returns zero (supposedly it would return non-zero on any errors, but FLTK calls exit directly for these). A normal program will end main() with return Fl::run();.

Note

Fl::run() and Fl::wait() (but not Fl::wait(double)) both return when all FLTK windows are closed. Therefore, a MacOS FLTK application possessing Fl_Sys_Menu_Bar items able to create new windows and expected to keep running without any open window cannot use these two functions. One solution is to run the event loop as follows:

```c
while (!Fl::program_should_quit()) Fl::wait(1e20);
```
### 31.2.3.54 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()

### 31.2.3.55 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.

### 31.2.3.56 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 widgets.
The new default size for widget scrollbars, in pixels.

31.2.3.57 set_box_color()

```cpp
type Fl::set_box_color ( Fl_Color c ) [static]
```

Sets the drawing color for the box that is currently drawn.

This method sets the current drawing color `fl_color()` depending on the widget's state to either `c` or `fl_inactive(c)`.

It should be used whenever a box background is drawn in the box (type) drawing code instead of calling `fl_color(← Fl_Color bg)` with the background color `bg`, usually `Fl_Widget::color()`.

This method is only useful inside box drawing code. Whenever a box is drawn with one of the standard box drawing methods, a static variable is set depending on the widget's current state - if the widget is inactive_r() then the internal variable is false (0), otherwise it is true (1). This is faster than calling `Fl_Widget::active_r()` because the state is cached.

See also

- `Fl::draw_box_active()`
- `Fl::box_color(Fl_Color)`

31.2.3.58 set_boxtype() [1/2]

```cpp
type Fl::set_boxtype ( Fl_Boxtype t, Fl_Box_Draw_F * f, uchar a, uchar b, uchar c, uchar d ) [static]
```

Sets the function to call to draw a specific boxtype.

31.2.3.59 set_boxtype() [2/2]

```cpp
type Fl::set_boxtype ( Fl_Boxtype to, Fl_Boxtype from ) [static]
```

Copies the from boxtype.
31.2.3.60  set_idle()

static void Fl::set_idle (  
   Fl_Old_Idle_Handler cb ) [inline], [static]

Sets an idle callback.

**Deprecated** This method is obsolete - use the add_idle() method instead.

31.2.3.61  set_labeltype() [1/2]

void Fl::set_labeltype {  
   Fl_Labeltype t,  
   Fl_Label_Draw_F * f,  
   Fl_Label_Measure_F * m ) [static]

Sets the functions to call to draw and measure a specific labeltype.

31.2.3.62  set_labeltype() [2/2]

static void Fl::set_labeltype {  
   Fl_Labeltype ,  
   Fl_Labeltype from ) [static]

Sets the functions to call to draw and measure a specific labeltype.

31.2.3.63  use_high_res_GL() [1/2]

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
31.2.3.64  use_high_res_GL() [2/2]

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

31.2.3.65  version()

double Fl::version ( ) [static]

Returns the compiled-in value of the FL_VERSION constant.

This is useful for checking the version of a shared library.

Deprecated  Use int Fl::api_version() instead.

31.2.3.66  visible_focus() [1/2]

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.

31.2.3.67  visible_focus() [2/2]

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.
31.2.3.68  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).

31.2.3.69  wait() [1/2]

    int Fl::wait ( ) [static]

Waits until "something happens" and then returns.

Call this repeatedly to "run" your program. You can also check what happened each time after this returns, which is
quite useful for managing program state.

What this really does is call all idle callbacks, all elapsed timeouts, call Fl::flush() to get the screen to update, and
then wait some time (zero if there are idle callbacks, the shortest of all pending timeouts, or infinity), for any events
from the user or any Fl::add_fd() callbacks. It then handles the events and calls the callbacks and then returns.

Returns

    non-zero if there are any visible windows - this may change in future versions of FLTK.
31.2.3.70 wait() [2/2]

double Fl::wait (  
    double time_to_wait  )  [static]

Waits a maximum of \texttt{time\_to\_wait} seconds or until "something happens".
See \texttt{Fl::wait()} for the description of operations performed when "something happens".

Returns

Always 1 on Windows. Otherwise, it is positive if an event or fd happens before the time elapsed. It is zero if nothing happens. It is negative if an error occurs (this will happen on X11 if a signal happens).

31.2.4 Member Data Documentation

31.2.4.1 help

const char *const Fl::help = helpmsg+13  [static]

Usage string displayed if \texttt{Fl::args()} detects an invalid argument.
This may be changed to point to customized text at run-time.

31.2.4.2 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

\texttt{add\_idle()}, \texttt{remove\_idle()}

The documentation for this class was generated from the following files:

- \texttt{Fl.H}
- \texttt{Fl.cxx}
- \texttt{Fl_abort.cxx}
- \texttt{Fl_add_idle.cxx}
- \texttt{Fl_arg.cxx}
- \texttt{fl_boxtype.cxx}
- \texttt{fl_color.cxx}
- \texttt{FlCompose.cxx}
- \texttt{Fl_display.cxx}
- \texttt{Fl_get_system_colors.cxx}
- \texttt{Fl_grab.cxx}
- \texttt{fl_labeltype.cxx}
- \texttt{Fl_lock.cxx}
- \texttt{Fl_own_colormap.cxx}
- \texttt{fl_set_font.cxx}
- \texttt{flShortcut.cxx}
- \texttt{Fl_visual.cxx}
- \texttt{Fl_Widget.cxx}
- \texttt{Fl_Window.cxx}
- \texttt{screen_xywh.cxx}
- \texttt{Fl_Cairo.cxx}
31.3 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.
- void soft (int s)
  If "soft" is turned on, the user is allowed to drag the value outside the range.
- int soft () const
  If "soft" is turned on, the user is allowed to drag the value outside the range.

Protected Member Functions

- void draw ()
  Draws the widget.
- int handle (int)
  Handles the specified event.
- void value_damage ()
  Asks for partial redraw.

Additional Inherited Members

31.3.1 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.

### 31.3.2 Constructor & Destructor Documentation

#### 31.3.2.1 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.

### 31.3.3 Member Function Documentation

#### 31.3.3.1 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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

#### 31.3.3.2 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.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<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.

### 31.3.3.3 soft() [1/2]

```cpp
def 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.

### 31.3.3.4 soft() [2/2]

```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.

The documentation for this class was generated from the following files:

- Fl_Adjuster.H
- Fl_Adjuster.cxx

### 31.4 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
```

Generated by Doxygen
Public Member Functions

• virtual Fl_Image * copy (int W, int H)
  Creates a resized copy of the specified image.

• Fl_Image * copy ()

• virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  Draws the image to the current drawing surface with a bounding box.

• void draw (int X, int Y)

• Fl_Bitmap (const uchar * bits, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.

• Fl_Bitmap (const char * bits, int W, int H)
  The constructors create a new bitmap from the specified bitmap data.

• 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 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 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 Attributes

• int alloc_array
  Non-zero if array points to bitmap data allocated internally.

• const uchar * array
  Pointer to raw bitmap data

Friends

• class Fl_Graphics_Driver

Additional Inherited Members

31.4.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.

Images are drawn using the current color.

31.4.2 Constructor & Destructor Documentation
31.4 Fl_Bitmap Class Reference

31.4.2.1 Fl_Bitmap() [1/2]

Fl_Bitmap::Fl_Bitmap (  
    const uchar * bits,  
    int W,  
    int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.

31.4.2.2 Fl_Bitmap() [2/2]

Fl_Bitmap::Fl_Bitmap (  
    const char * array,  
    int W,  
    int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.

31.4.3 Member Function Documentation

31.4.3.1 copy()

Fl_Image * Fl_Bitmap::copy (  
    int W,  
    int H ) [virtual]

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.

Parameters

\begin{verbatim}
W,H  width and height of the returned copied image
\end{verbatim}

Reimplemented from Fl_Image.

31.4.3.2 draw()

void Fl_Bitmap::draw (  
    int X,  
    int Y,  
    int W,
int H,  
int cx = 0,  
int cy = 0 ) [virtual]

Draws the image to the current drawing surface with a bounding box.

Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.

In other words: fl_push_clip(X, Y, W, H) is applied, the image is drawn with its upper-left corner at X-cx, Y-cy and its own width and height, fl_pop_clip() is applied.

Reimplemented from Fl_Image.

31.4.3.3 label() [1/2]

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.

31.4.3.4 label() [2/2]

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.

31.4.3.5 uncache()

void Fl_Bitmap::uncache ( ) [virtual]

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented from Fl_Image.

The documentation for this class was generated from the following files:

- Fl_Bitmap.H
- Fl_Bitmapcxx
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      V
Fl_RGB_Image
         |      |
        V      V
Fl_BMP_Image
```

Public Member Functions

- `Fl_BMP_Image (const char *filename)`
  
  This constructor loads the named BMP image from the given BMP filename.

- `Fl_BMP_Image (const char *imagename, const unsigned char *data)`
  
  Read a BMP image from memory.

Protected Member Functions

- void `load_bmp_(class Fl_Image_Reader &rdr)`

Additional Inherited Members

31.5.1 Detailed Description

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

31.5.2 Constructor & Destructor Documentation

31.5.2.1 Fl_BMP_Image() [1/2]

```
Fl_BMP_Image::Fl_BMP_Image ( const char * filename )
```

This constructor loads the named BMP image from the given BMP filename.

The destructor frees all memory and server resources that are used by the image.

Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>filename</strong></td>
<td>a full path and name pointing to a valid BMP file.</td>
</tr>
</tbody>
</table>

See also

```cpp
Fl_BMP_Image::Fl_BMP_Image(const char * imagename, const unsigned char * data)
```

### 31.5.2.2 Fl_BMP_Image()

```cpp
Fl_BMP_Image::Fl_BMP_Image (  
    const char * imagename,  
    const unsigned char * data  
)
```

Read a BMP image from memory.

Construct an image from a block of memory inside the application. Fluid offers “binary data” chunks as a great way to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

Use `Fl_Image::fail()` to check if `Fl_BMP_Image` failed to load. `fail()` returns `ERR_FILE_ACCESS` if the image could not be read from memory, `ERR_FORMAT` if the BMP format could not be decoded, and `ERR_NO_IMAGE` if the image could not be loaded for another reason.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>imagename</strong></td>
<td>A name given to this image or NULL</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>data</strong></td>
<td>Pointer to the start of the BMP image in memory. This code will not check for buffer overruns.</td>
</tr>
</tbody>
</table>

See also

```cpp
Fl_BMP_Image::Fl_BMP_Image(const char * filename)
Fl_Shared_Image
```

The documentation for this class was generated from the following files:

- `Fl_BMP_Image.H`
- `Fl_BMP_Image.cxx`

### 31.6 Fl_Box Class Reference

This widget simply draws its box, and possibly its label.

```cpp
#include <Fl_Box.H>
```

Inheritance diagram for Fl_Box:
Public Member Functions

- `Fl_Box (int X, int Y, int W, int H, const char *l=0)`
- `Fl_Box (Fl_Boxtype b, int X, int Y, int W, int H, const char *l)`
  
  See `Fl_Box::Fl_Box(int x, int y, int w, int h, const char * = 0)`
- virtual int handle (int)

Handle the specified event.

Protected Member Functions

- void draw ()

Draws the widget.

Additional Inherited Members

31.6.1 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.

31.6.2 Constructor & Destructor Documentation

31.6.2.1 `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.
31.6.3 Member Function Documentation

31.6.3.1 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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

31.6.3.2 handle()

```cpp
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

<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>
31.7 Fl_Browser Class Reference

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.

#include <Fl_Browser.H>

Inheritance diagram for Fl_Browser:

```
        Fl_Widget
          |        
          |        
        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 Member Functions

- void add (const char *newtext, void *d=0)  
  Adds a new line to the end of the browser.
- void bottomline (int line)  
  Scrolls the browser so the bottom item in the browser is showing the specified line.
- void clear ()  
  Removes all the lines in the browser.
- char column_char () const  
  Gets the current column separator character.
- void column_char (char c)  
  Sets the column separator to c.
- const int * column_widths () const
Gets the current column width array.
• void column_widths (const int *arr)
  Sets the current array to arr.
• void *data (int line) const
  Returns the user data() for specified line.
• void data (int line, void *d)
  Sets the user data for specified line to d.
• void display (int line, int val=1)
  For back compatibility.
• int displayed (int line) const
  Returns non-zero if line has been scrolled to a position where it is being displayed.
• Fl_Browser (int X, int Y, int W, int H, const char *L=0)
  The constructor makes an empty browser.
• char format_char () const
  Gets the current format code prefix character, which by default is '@'.
• void format_char (char c)
  Sets the current format code prefix character to c.
• void hide (int line)
  Makes line invisible, preventing selection by the user.
• void hide ()
  Hides the entire Fl_Browser widget – opposite of show().
• void icon (int line, Fl_Image *icon)
  Set the image icon for line to the value icon.
• Fl_Image *icon (int line) const
  Returns the icon currently defined for line.
• 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 (int line)
  Makes line visible, and available for selection by user.
• void show ()
  Shows the entire Fl_Browser widget – opposite of hide().
31.7 Fl_Browser Class Reference

- **int size () const**
  
  Returns how many lines are in the browser.

- **void size (int W, int H)**
  
  Swaps two browser lines a and b.

- **const char * text (int line) const**
  
  Returns the label text for the specified line.

- **void 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.

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_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.*

### Additional Inherited Members

#### 31.7.1 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.

![Figure 31.2 Fl_Hold_Browser](image)

**Figure 31.2 Fl_Hold_Browser**

![Figure 31.3 Fl_Multi_Browser](image)

**Figure 31.3 Fl_Multi_Browser**

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 \#t, 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)`.

### 31.7.2 Constructor & Destructor Documentation

#### 31.7.2.1 Fl_Browser()

```cpp
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>

### 31.7.3 Member Function Documentation

#### 31.7.3.1 _remove()

```cpp
FL_BLINE ∗ Fl_Browser::_remove ( int iLine ) [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()

31.7.3.2 add()

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()

31.7.3.3 bottomline()

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()`

### 31.7.3.4 clear()

```cpp
void Fl_Browser::clear ()
```

Removes all the lines in the browser.

See also

`add(), insert(), remove(), swap(int,int), clear()`

### 31.7.3.5 column_char()[1/2]

```cpp
char Fl_Browser::column_char () const [inline]
```

Gets the current column separator character.

The default is `'	'` (tab).

See also

`column_char(), column_widths()`

### 31.7.3.6 column_char()[2/2]

```cpp
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 `'	'` (tab).

See also

`column_char(), column_widths()`
31.7.3.7 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:

```cpp
Fl_Browser *b = new Fl_Browser(...);
static int widths[] = { 50, 50, 50, 70, 70, 40, 40, 70, 70, 50, 0 }; // widths for each column
b->column_widths(widths); // assign array to widget
b->column_char(','); // use tab as the column character
b->add("root\t2888\t0.0\t0.0\t1352\ttty3\tSW\tAug15\t0:00\t/sbin/mingetty tty3");
```

See also

`column_char(), column_widths()`

31.7.3.8 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()`

31.7.3.9 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)

31.7.3.10 data()[2/2]

```cpp
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) |

31.7.3.11 display()

```cpp
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()

31.7.3.12 displayed()

```cpp
int Fl_Browser::displayed (  
    int line ) const  
```

Returns non-zero if line has been scrolled to a position where it is being displayed.

Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide()/show() status of the widget or item.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line to be checked</th>
</tr>
</thead>
</table>

Returns

1 if visible, 0 if not visible.

See also

toline(), middleline(), bottomline(), displayed(), lineposition()

31.7.3.13  find_line()

`FL_BLINE * Fl_Browser::find_line ( int line ) const [protected]`

Returns the item for specified `line`.

Note: This call is slow. It's fine for e.g. responding to user clicks, but slow if called often, such as in a tight sorting loop. Finding an item 'by line' involves a linear lookup on the internal linked list. The performance hit can be significant if the browser's contents is large, and the method is called often (e.g. during a sort). If you're writing a subclass, use the protected methods item_first(), item_next(), etc. to access the internal linked list more efficiently.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line number of the item to return. (1 based)</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>item</th>
<th>that was found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if line is out of range.</td>
</tr>
</tbody>
</table>

See also

item_at(), find_line(), lineno()

31.7.3.14  format_char() [1/2]

`char Fl_Browser::format_char ( ) const [inline]`

Gets the current format code prefix character, which by default is '@'.

A string of formatting codes at the start of each column are stripped off and used to modify how the rest of the line is printed:
• '@.' Print rest of line, don’t look for more '@' signs

• '@@' Doubling the format character prints the format character once, followed by the rest of line

• '@l' Use a LARGE (24 point) font

• '@m' Use a medium large (18 point) font

• '@s' Use a small (11 point) font

• '@b' Use a bold font (adds FL_BOLD to font)

• '@i' Use an italic font (adds FL_ITALIC to font)

• '@f' or '@t' Use a fixed-pitch font (sets font to FL_COURIER)

• '@c' Center the line horizontally

• '@r' Right-justify the text

• '@N' Use fl_inactive_color() to draw the text

• '@B0', '@B1', ... '@B255' Fill the background with fl_color(n)

• '@C0', '@C1', ... '@C255' Use fl_color(n) to draw the text

• '@F0', '@F1', ... Use fl_font(n) to draw the text

• '@S1', '@S2', ... Use point size n to draw the text

• '@u' or '@_' Underline the text.

• '@-' draw an engraved line through the middle.

Notice that the '@.' command can be used to reliably terminate the parsing. To print a random string in a random color, use sprintf("@%d@.%s", color, string) and it will work even if the string starts with a digit or has the format character in it.

31.7.3.15 format_char() [2/2]

void Fl_Browser::format_char ( char c ) [inline]

Sets the current format code prefix character to c.

The default prefix is '@'. Set the prefix to 0 to disable formatting.

See also

format_char() for list of '@' codes
31.7.3.16  full_height()

```c
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_`.

31.7.3.17  hide() [1/2]

```c
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()`

31.7.3.18  hide() [2/2]

```c
void Fl_Browser::hide ( ) [inline], [virtual]
```

Hides the entire `Fl_Browser` widget – opposite of `show()`.

Reimplemented from `Fl_Widget`. 
31.7.3.19 icon() [1/2]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in line</td>
<td>The line to be modified. If out of range, nothing is done.</td>
</tr>
<tr>
<td>in icon</td>
<td>The image icon to be assigned to the line. If NULL, any previous icon is removed.</td>
</tr>
</tbody>
</table>

31.7.3.20 icon() [2/2]

Fl_Image * Fl_Browser::icon (  
    int line ) const

Returns the icon currently defined for line.

If no icon is defined, NULL is returned.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in line</td>
<td>The line whose icon is returned.</td>
</tr>
</tbody>
</table>

Returns

The icon defined, or NULL if none.

31.7.3.21 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_.
31.7.3.22 insert() [1/2]

```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

| in  | line | The new line will be inserted above this line (1 based). |
|     | item | The item to be added. |

31.7.3.23 insert() [2/2]

```cpp
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.

Parameters

| in  | line | Line position for insert. (1 based) |
|     |      | If `line > size()`, the entry will be added at the end. |
| in  | newtext | The label text for the new line. |
| in  | d     | Optional pointer to user data to be associated with the new line. |

31.7.3.24 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>in</th>
<th>line</th>
<th>The line of the item to return. (1 based)</th>
</tr>
</thead>
</table>

Returns

The item, or NULL if line out of range.

See also

item_at(), find_line(), lineno()

Reimplemented from Fl_Browser_.

31.7.3.25 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>in</th>
<th>item</th>
<th>The item to be drawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>position and size.</td>
</tr>
</tbody>
</table>

Implements Fl_Browser_.

31.7.3.26 item_first()

```cpp
void * Fl_Browser::item_first ( ) const [protected], [virtual]
```

Returns the very first item in the list.

Example of use:

```cpp
// Walk the browser from beginning to end
for ( void *i=item_first(); i; i=item_next(i) ) {  
    printf("item label='\$s'\n", item_text(i));  
}
```
Returns

The first item, or NULL if list is empty.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

31.7.3.27 item_height()

```cpp
int Fl_Browser::item_height ( 
    void * item ) const [protected], [virtual]
```

Returns height of item in pixels.

This takes into account embedded @ codes within the text() label.

Parameters

| in | item | The item whose height is returned. |

Returns

The height of the item in pixels.

See also

item_height(), item_width(), incr_height(), full_height()

Implements Fl_Browser_.

31.7.3.28 item_last()

```cpp
void * Fl_Browser::item_last ( ) const [protected], [virtual]
```

Returns the very last item in the list.

Example of use:

```c
// 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_.

31.7.3.29 item_next()

void * Fl_Browser::item_next ( void * item ) const [protected], [virtual]

Returns the next item after item.

Parameters

| in  | item | The 'current' item |

Returns

The next item after item, or NULL if there are none after this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

31.7.3.30 item_prev()

void * Fl_Browser::item_prev ( void * item ) const [protected], [virtual]

Returns the previous item before item.

Parameters

| in | item | The 'current' item |

Returns

The previous item before item, or NULL if there are none before this one.
See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

### 31.7.3.31 item_select()

```c
void Fl_Browser::item_select ( void * item,
                                int val ) [protected], [virtual]
```

Change the selection state of item to the value val.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be changed.</td>
</tr>
<tr>
<td>in</td>
<td>val</td>
<td>The new selection state: 1 selects, 0 de-selects.</td>
</tr>
</tbody>
</table>

See also

select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

### 31.7.3.32 item_selected()

```c
int Fl_Browser::item_selected ( void * item ) const [protected], [virtual]
```

See if item is selected.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item whose selection state is to be checked.</td>
</tr>
</tbody>
</table>

Returns

1 if selected, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

Generated by Doxygen
### 31.7.3.33 item_swap()

```cpp
void Fl_Browser::item_swap (  
    void * a,  
    void * b ) [inline], [protected], [virtual]
```

Swap the items `a` and `b`.

You must call `redraw()` to make any changes visible.

**Parameters**

| in  | a, b | the items to be swapped. |

**See also**

`swap(int,int), item_swap()`

Reimplemented from `Fl_Browser_`.

### 31.7.3.34 item_text()

```cpp
const char * Fl_Browser::item_text (  
    void * item ) const [protected], [virtual]
```

Returns the label text for `item`.

**Parameters**

| in | item | The item whose label text is returned. |

Returns

The item's text string. (Can be NULL)

Reimplemented from `Fl_Browser_`.

### 31.7.3.35 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_.

31.7.3.36  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()

31.7.3.37  lineposition()  

```cpp
void Fl_Browser::lineposition ( int line,
                                Fl_Line_Position pos )
```

Updates the browser so that line is shown at position pos.
31.7 Fl_Browser Class Reference

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>line number. (1 based)</td>
</tr>
<tr>
<td>in</td>
<td>pos</td>
<td>position.</td>
</tr>
</tbody>
</table>

See also
toline(), middleline(), bottomline()

31.7.3.38 load()

```
int Fl_Browser::load (const char * filename)
```

Clears the browser and reads the file, adding each line from the file to the browser.

If the filename is NULL or a zero-length string then this just clears the browser. This returns zero if there was any error in opening or reading the file, in which case errno is set to the system error. The data() of each line is set to NULL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>filename</td>
<td>The filename to load</td>
</tr>
</tbody>
</table>

Returns

1 if OK, 0 on error (errno has reason)

See also
add()

31.7.3.39 make_visible()

```
void Fl_Browser::make_visible (int line) [inline]
```

Make the item at the specified line visible().

Functionally similar to show(int line). If line is out of range, redisplay top or bottom of list as appropriate.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>The line to be made visible.</td>
</tr>
</tbody>
</table>
See also

show(int), hide(int), display(), visible(), make_visible()

31.7.3.40 middleline()

void Fl_Browser::middleline (  
  int line ) [inline]

Scrolls the browser so the middle item in the browser is showing the specified line.

Parameters

| in  | line | The line to be displayed in the middle. |

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

31.7.3.41 move()

void Fl_Browser::move (  
  int to,  
  int from )

Line from is removed and reinserted at to.

Note: to is calculated after line from gets removed.

Parameters

| in  | to   | Destination line number (calculated after line from is removed) |
| in  | from | Line number of item to be moved |

31.7.3.42 remove()

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()

31.7.3.43 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

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line whose icon is to be removed.</th>
</tr>
</thead>
</table>

31.7.3.44 replace()

```cpp
void Fl_Browser::replace ( int a, const char ∗ b ) [inline]
```

For back compatibility only.

31.7.3.45 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></td>
<td></td>
<td></td>
</tr>
<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()

31.7.3.46 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()

31.7.3.47 show() [1/2]

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()
31.7.3.48 show() [2/2]

void Fl_Browser::show ( ) [inline], [virtual]

Shows the entire Fl_Browser widget – opposite of hide().

Reimplemented from Fl_Widget.

31.7.3.49 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.

31.7.3.50 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()

31.7.3.51 swap() [2/2]

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()`

31.7.3.52  text() [1/2]

```cpp
class Fl_Browser {  
    const char * 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)
```

31.7.3.53  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. |
31.7.3.54 текстsize()

```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.

31.7.3.55 topline() [1/2]

```cpp
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.

31.7.3.56 topline() [2/2]

```cpp
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()
31.7.3.57 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()

31.7.3.58 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()

31.7.3.59 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:
31.8 Fl_Browser_ Class Reference

This is the base class for browsers.

```cpp
#include <Fl_Browser_.H>
```

Inheritance diagram for Fl_Browser_:

```
Fl_Widget
    Fl_Group
        Fl_Browser_
            Fl_Browser Fl_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 `has_scrollbar()`.

Public Member Functions

- `int deselect (int docallbacks=0)`
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- `void display (void *item)`
  Displays the item, scrolling the list as necessary.
- `int handle (int event)`
  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`

Generated by Doxygen
Gets the vertical scroll position of the list as a pixel position \texttt{pos}.

• void \texttt{position} (int \texttt{pos})
  
  Sets the vertical scroll position of the list to pixel position \texttt{pos}.

• void \texttt{resize} (int \texttt{X}, int \texttt{Y}, int \texttt{W}, int \texttt{H})
  
  Repositions and/or resizes the browser.

• void \texttt{scrollbar\_left} ()
  
  Moves the vertical scrollbar to the lefthand side of the list.

• void \texttt{scrollbar\_right} ()
  
  Moves the vertical scrollbar to the righthand side of the list.

• int \texttt{scrollbar\_size} () const
  
  Gets the current size of the scrollbars' troughs, in pixels.

• void \texttt{scrollbar\_size} (int \texttt{newSize})
  
  Sets the pixel size of the scrollbars' troughs to \texttt{newSize}, in pixels.

• int \texttt{scrollbar\_width} () const
  
  Returns the global value \texttt{Fl::scrollbar\_size()}. Sets the global \texttt{Fl::scrollbar\_size()}, and forces this instance of the widget to use it.

• void \texttt{scrollbar\_width} (int \texttt{width})
  
  Sets the selection state of \texttt{item} to \texttt{val}, and returns 1 if the state changed or 0 if it did not.

• int \texttt{select\_only} (void \texttt{+item}, int \texttt{docallbacks}=0)
  
  Selects \texttt{item} and returns 1 if the state changed or 0 if it did not.

• void \texttt{sort} (int \texttt{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.

• void \texttt{textcolor} (\texttt{Fl\_Color \texttt{col}})
  
  Sets the default text color for the lines in the browser to \texttt{color \texttt{col}}.

• \texttt{Fl\_Font textfont} () const
  
  Gets the default text font for the lines in the browser.

• void \texttt{textfont} (\texttt{Fl\_Font \texttt{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.

• void \texttt{textsize} (\texttt{Fl\_Fontsize \texttt{newSize}})
  
  Sets the default text size (in pixels) for the lines in the browser to \texttt{size}.

Public Attributes

• \texttt{Fl\_Scrollbar hscrollbar}
  
  Horizontal scrollbar.

• \texttt{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.

Additional Inherited Members

31.8.1 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.

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.

31.8.2 Member Enumeration Documentation
31.82.1 anonymous enum

anonymous enum

Values for has_scrollbar().

Anonymous enum bit flags for has_scrollbar().

- bit 0: horizontal
- bit 1: vertical
- bit 2: ‘always’ (to be combined with bits 0 and 1)
- bit 3-31: reserved for future use

Enumerator

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HORIZONTAL</td>
<td>Only show horizontal scrollbar.</td>
</tr>
<tr>
<td>VERTICAL</td>
<td>Only show vertical scrollbar.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Show both scrollbars. (default)</td>
</tr>
<tr>
<td>ALWAYS_ON</td>
<td>Specified scrollbar(s) should ‘always’ be shown (to be used with HORIZONTAL/VERTICAL)</td>
</tr>
<tr>
<td>HORIZONTAL_ALWAYS</td>
<td>Horizontal scrollbar always on.</td>
</tr>
<tr>
<td>VERTICAL_ALWAYS</td>
<td>Vertical scrollbar always on.</td>
</tr>
<tr>
<td>BOTH_ALWAYS</td>
<td>Both scrollbars always on.</td>
</tr>
</tbody>
</table>

31.83 Constructor & Destructor Documentation

31.83.1 Fl_Browser()

Fl_Browser_::Fl_Browser_ (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * L = 0 ) [protected]

The constructor makes an empty browser.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in X, Y, W, H</td>
<td>position and size.</td>
</tr>
<tr>
<td>in L</td>
<td>The label string, may be NULL.</td>
</tr>
</tbody>
</table>
31.8.4 Member Function Documentation

31.8.4.1 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) |

31.8.4.2 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. |

31.8.4.3 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 non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default). |

31.8.4.4 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()

31.8.4.5 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()
31.8.4.6 find_item()

```cpp
void * Fl_Browser_::find_item ( 
    int ypos ) [protected]
```

This method returns the item under mouse y position ypos.

NULL is returned if no item is displayed at that position.

**Parameters**

- ypos

**Returns**

- The item, or NULL if not found

---

31.8.4.7 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.

---

31.8.4.8 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.

---

31.8.4.9 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.

31.8.4.10 has_scrollbar()

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.

31.8.4.11 hposition()[1/2]

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()

31.8.4.12 hposition()[2/2]

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()

31.8.13 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.

31.8.14 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 |
| in | b | The new item being inserted |

31.8.15 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.
Parameters

| in  | index  | The index of the item to be returned |

Returns

The item at the specified index.

Reimplemented in Fl_Browser, and Fl_Check_Browser.

31.8.4.16 item_first()

virtual void* Fl_Browser_::item_first ( ) const [protected], [pure virtual]

This method must be provided by the subclass to return the first item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Browser, and Fl_Check_Browser.

31.8.4.17 item_height()

virtual int Fl_Browser_::item_height ( void * item ) const [protected], [pure virtual]

This method must be provided by the subclass to return the height of item in pixels.

Allow for two additional pixels for the list selection box.

Parameters

| in  | item  | The item whose height is returned. |

Returns

The height of the specified item in pixels.

See also

item_height(), item_width(), item_quick_height()

Implemented in Fl_Browser, and Fl_Check_Browser.
31.8.4.18 item_last()

virtual void Fl_Browser_::item_last ( ) const [inline], [protected], [virtual]

This method must be provided by the subclass to return the last item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Reimplemented in Fl_Browser.

31.8.4.19 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, and Fl_Check_Browser.

31.8.4.20 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, and Fl_Check_Browser.

31.8.4.21 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.

31.8.4.22 item_select()

```cpp
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 \( \text{val} \) for the \text{item}.

Sets the selection state for \text{item}, where optional \text{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 \text{Fl_Browser}, and \text{Fl_Check_Browser}.

31.8.4.23 item_selected()

```cpp
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 \text{item}.

The method should return 1 if \text{item} is selected, or 0 otherwise.

Parameters

| in | item | The item to test. |

Reimplemented in \text{Fl_Browser}, and \text{Fl_Check_Browser}.  

---

Generated by Doxygen
31.8.4.24 item_swap()

```cpp
virtual void Fl_Browser_::item_swap (
    void * a,
    void * b ) [inline], [protected], [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items a and b, such as for sorting.

**Parameters**

| in | a, b | The two items to be swapped. |

Reimplemented in `Fl_Browser`, and `Fl_Check_Browser`.

31.8.4.25 item_text()

```cpp
virtual const char* Fl_Browser_::item_text (
    void * item ) const [inline], [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

**Parameters**

| in | item | The item whose label text is returned. |

**Returns**

The item's text label. (Can be NULL if blank)

Reimplemented in `Fl_Browser`, and `Fl_Check_Browser`.

31.8.4.26 item_width()

```cpp
virtual int Fl_Browser_::item_width (
    void * item ) const [protected], [pure virtual]
```

This method must be provided by the subclass to return the width of the item in pixels.

Allow for two additional pixels for the list selection box.

**Parameters**

| in | item | The item whose width is returned. |

Generated by Doxygen
Returns
The width of the item in pixels.

Implemented in Fl_Browser, and Fl_Check_Browser.

31.8.4.27 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()

31.8.4.28 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.

31.8.4.29 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()

31.8.4.30 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()

31.8.4.31 redraw_line()

```cpp
void Fl_Browser_::redraw_line ( void * item ) [protected]
```

This method should be called when the contents of item has changed, but not its height.

Parameters

| in | item | The item that needs to be redrawn. |

See also

- redraw_lines(), redraw_line()

31.8.4.32 redraw_lines()

```cpp
void Fl_Browser_::redraw_lines ( ) [inline], [protected]
```

This method will cause the entire list to be redrawn.

See also

- redraw_lines(), redraw_line()

31.8.4.33 replacing()

```cpp
void Fl_Browser_::replacing ( void * a, void * b ) [protected]
```

This method should be used when item a is being replaced by item b.

It allows the Fl_Browser_ to update its cache data as needed, schedules a redraw for the item being changed, and tries to maintain the selection. This method does not actually replace the item, but handles the follow up bookkeeping after the item has just been replaced.

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>a</th>
<th>Item being replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b</td>
<td>Item to replace 'a'</td>
</tr>
</tbody>
</table>

31.8.4.34 `resize()`

```c
void Fl_Browser_::resize (  
    int _X,  
    int _Y,  
    int _W,  
    int _H ) [virtual]
```

Repositions and/or resizes the browser.

Parameters

| in | X,Y,W,H | The new position and size for the browser, in pixels. |

Reimplemented from `Fl_Widget`.

31.8.4.35 `scrollbar_left()`

```c
void Fl_Browser_::scrollbar_left ( ) [inline]
```

Moves the vertical scrollbar to the lefthand side of the list.

For back compatibility.

31.8.4.36 `scrollbar_right()`

```c
void Fl_Browser_::scrollbar_right ( ) [inline]
```

Moves the vertical scrollbar to the righthand side of the list.

For back compatibility.

31.8.4.37 `scrollbar_size()` [1/2]

```c
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)`
31.8.4.38 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()`

31.8.4.39 scrollbar_width() [1/2]

int Fl_Browser_::scrollbar_width ( ) const [inline]

Returns the global value `Fl::scrollbar_size()`.

**Deprecated** Use `scrollbar_size()` instead.

**Todo** This method should eventually be removed in 1.4+

31.8.4.40 scrollbar_width() [2/2]

void Fl_Browser_::scrollbar_width (  
    int width ) [inline]

Sets the global `Fl::scrollbar_size()`, and forces this instance of the widget to use it.

**Deprecated** Use `scrollbar_size()` instead.

**Todo** This method should eventually be removed in 1.4+
31.8.4.41  select()

int Fl_Browser_::select (  
    void * item,  
    int val = 1,  
    int docallbacks = 0 )

Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

If docallbacks is non-zero, select tries to call the callback function for the widget.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item whose selection state is to be changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>The new selection state (1=select, 0=de-select)</td>
</tr>
<tr>
<td>in</td>
<td>docallbacks</td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>

Returns

1 if state was changed, 0 if not.

31.8.4.42  select_only()

int Fl_Browser_::select_only (  
    void * item,  
    int docallbacks = 0 )

Selects item and returns 1 if the state changed or 0 if it did not.

Any other items in the list are deselected.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to select.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallbacks</td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>

31.8.4.43  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.
### 31.8.4.44 sort()

```cpp
void Fl_Browser_::sort {
    int flags = 0
}
```

Sort the items in the browser based on flags.

*item_swap(void*, void*) and item_text(void*) must be implemented for this call.*

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FL_SORT_ASCENDING – sort in ascending order</td>
</tr>
<tr>
<td></td>
<td>FL_SORT_DESCENDING – sort in descending order</td>
</tr>
<tr>
<td></td>
<td>Values other than the above will cause undefined behavior</td>
</tr>
<tr>
<td></td>
<td>Other flags may appear in the future.</td>
</tr>
</tbody>
</table>

**Todo** Add a flag to ignore case

### 31.8.4.45 swapping()

```cpp
void Fl_Browser_::swapping {
    void * a,
    void * b ) [protected]
```

This method should be used when two items `a` and `b` are being swapped.

It allows the `Fl_Browser_` to update its cache data as needed, schedules a redraw for the two items, and tries to maintain the current selection. This method does not actually swap items, but handles the follow up bookkeeping after items have been swapped.

**Parameters**

| in | a,b | Items being swapped. |

### 31.8.4.46 textfont()

```cpp
Fl_Font Fl_Browser_::textfont ( ) const [inline]
```

Gets the default text font for the lines in the browser.

**See also**

- `textfont()`, `textsize()`, `textcolor()`
31.8.5  Member Data Documentation

31.8.5.1  hscrollbar

**Fl_Scrollbar** Fl_Browser_::hscrollbar

Horizontal scrollbar.

Public, so that it can be accessed directly.

31.8.5.2  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

31.9  Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

```cpp
#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 (int s)
  
  *Sets the shortcut key to s.*

- void shortcut (const char ∗s)
  
  (for backwards compatibility)

- int value (int v)
  
  *Sets the current value of the button.*

- char value () const
  
  *Returns the current value of the button (0 or 1).*

Protected Member Functions

- virtual void draw ()
  
  *Draws the widget.*

- void simulate_key_action ()

Static Protected Member Functions

- static void key_release_timeout (void ∗)

Static Protected Attributes

- static Fl_Widget_Tracker ∗ key_release_tracker = 0
31.9.1 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 before it. The user will have to hold down the `Alt` key so that the input field does not eat the event first as an `FL_KEYBOARD` event.

See also

```cpp
Fl_Widget::shortcut_label(int)
```

**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).

31.9.2 Constructor & Destructor Documentation

31.9.2.1 `Fl_Button()`

```cpp
Fl_Button::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>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>

### 31.9.3 Member Function Documentation

#### 31.9.3.1 clear()

```cpp
to int Fl_Button::clear () [inline]
```

Same as `value(0)`.

See also

- `value(int v)`

#### 31.9.3.2 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`

#### 31.9.3.3 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.
31.9.3.4 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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

Reimplemented in `Fl_Light_Button`, and `Fl_Return_Button`.

31.9.3.5 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
```

Generated by Doxygen
31.9.3.6 set()

```c
int Fl_Button::set () [inline]
```

Same as `value(1)`.

See also

`value(int v)`

31.9.3.7 shortcut() [1/2]

```c
int Fl_Button::shortcut () const [inline]
```

Returns the current shortcut key for the button.

Return values

<table>
<thead>
<tr>
<th>int</th>
</tr>
</thead>
</table>

31.9.3.8 shortcut() [2/2]

```c
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).
31.9.3.9 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v</code></td>
<td>button value</td>
</tr>
</tbody>
</table>

See also

- `set()`, `clear()`

The documentation for this class was generated from the following files:

- `Fl_Button.H`
- `Fl_Button.cxx`

### 31.10 Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.

```cpp
#include <Fl_Cairo.H>
```

#### Public Member Functions

- `bool autolink () const`
  
  *Gets the autolink option. See Fl::cairo_autolink_context(bool)*

- `void autolink (bool b)`
  
  *Sets the autolink option, only available with –enable-cairoext.*

- `cairo_t * cc () const`
  
  *Gets the current cairo context.*

- `void cc (cairo_t *c, bool own=true)`
  
  *Sets the current cairo context.*

- `void gc (void *c)`
  
  *Sets the gc c to keep track on.*

- `void * gc () const`
  
  *Gets the last gc attached to a cc.*

- `void window (void *w)`
  
  *Sets the window w to keep track on.*

- `void * window () const`
  
  *Gets the last window attached to a cc.*
31.10.1 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

31.10.2 Member Function Documentation

31.10.2.1 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

31.11 Fl_Cairo_Window Class Reference

This defines a FLTK window with cairo support.

#include <Fl_Cairo_Window.H>

Inheritance diagram for Fl_Cairo_Window:

```
Fl_Widget
     |            |
     v            v
Fl_Group
     |            |
     v            v
Fl_Window
     |            |
     v            v
Fl_Double_Window
     |            |
     v            v
Fl_Cairo_Window
```

Generated by Doxygen
Public Types

- typedef void (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, const char *L=0)
- Fl_Cairo_Window (int X, int Y, int W, int H, const char *L=0)
- void set_draw_cb (cairo_draw_cb cb)
  
  You must provide a draw callback which will implement your cairo rendering.

Protected Member Functions

- void draw ()
  
  Overloaded to provide cairo callback support.

Additional Inherited Members

31.11.1 Detailed Description

This defines a FLTK window with cairo support.

This class overloads the virtual draw() method for you, so that the only thing you have to do is to provide your cairo code. All cairo context handling is achieved transparently.

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()

31.11.2 Member Function Documentation

31.11.2.1 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
31.12 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.
Protected Member Functions

- void draw()
  
  Draws the widget.

Additional Inherited Members

31.12.1 Detailed Description

Fl_Chart displays simple charts.

It is provided for Forms compatibility.

![Figure 31.4 Fl_Chart](image)

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.

31.12.2 Constructor & Destructor Documentation

31.12.2.1 Fl_Chart()

Fl_Chart::Fl_Chart (  
        int X,  
        int Y,  
        int W,  
        int H,  
        const char * L = 0 )

Create a new Fl_Chart widget using the given position, size and label string.

The default boxstyle is FL_NO_BOX.
31.12.3 Member Function Documentation

31.12.3.1 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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

31.12.3.2 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.

31.12.3.3 autosize() [2/2]

```cpp
void Fl_Chart::autosize (  
    uchar n ) [inline]
```

Set whether the chart will automatically adjust the bounds of the chart.
### 31.12.3.4 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**

| in | n | non-zero to enable automatic resizing, zero to disable |

### 31.12.3.5 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 |

### 31.12.3.6 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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`. 

---

Generated by Doxygen
31.12.3.7 insert()

```cpp
void Fl_Chart::insert ( int ind, double val, const char * str = 0, unsigned col = 0 )
```

Inserts a data value \( v \) at the given position \( \text{ind} \).

Position 1 is the first data value.

Parameters

<table>
<thead>
<tr>
<th>in ( \text{ind} )</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v )</td>
<td>data value</td>
</tr>
<tr>
<td>( \text{str} )</td>
<td>optional data label</td>
</tr>
<tr>
<td>( \text{col} )</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

31.12.3.8 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. |

31.12.3.9 replace()

```cpp
void Fl_Chart::replace ( int ind, double val, const char * str = 0, unsigned col = 0 )
```

Replace a data value \( v \) at the given position \( \text{ind} \).

Position 1 is the first data value.

Parameters

<table>
<thead>
<tr>
<th>in ( \text{ind} )</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v )</td>
<td>data value</td>
</tr>
<tr>
<td>( \text{str} )</td>
<td>optional data label</td>
</tr>
<tr>
<td>( \text{col} )</td>
<td>optional data color</td>
</tr>
</tbody>
</table>
31.12.3.10  textcolor()

```cpp
void Fl_Chart::textcolor ( 
    Fl_Color n ) [inline]
```

gets the chart's text color to \texttt{n}.

31.12.3.11  textfont()

```cpp
void Fl_Chart::textfont ( 
    Fl_Font s ) [inline]
```

Sets the chart's text font to \texttt{s}.

31.12.3.12  textsize()

```cpp
void Fl_Chart::textsize ( 
    Fl_Fontsize s ) [inline]
```

gets the chart's text size to \texttt{s}.

The documentation for this class was generated from the following files:

- Fl_Chart.H
- Fl_Chart.cxx

31.13  FL_CHART_ENTRY Struct Reference

For internal use only.

```c
#include <Fl_Chart.H>
```

Public Attributes

- unsigned \texttt{col}  
  \textit{For internal use only.}
- char \texttt{str [FL_CHART_LABEL_MAX+1]}  
  \textit{For internal use only.}
- float \texttt{val}  
  \textit{For internal use only.}
31.14 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
```

Generated by Doxygen
Public Member Functions

- **int add (char *s)**
  
  Add a new unchecked line to the end of the browser.

- **int add (char *s, int b)**
  
  Add a new line to the end of the browser.

- **int add (const char *s)**
  
  See int Fl_Check_Browser::add(char *s)

- **int add (const char *s, int b)**
  
  See int Fl_Check_Browser::add(char *s)

- **void check_all ()**
  
  Sets all the items checked.

- **void check_none ()**
  
  Sets all the items unchecked.

- **int checked (int item) const**
  
  Gets the current status of item item.

- **void checked (int item, int b)**
  
  Sets the check status of item item to b.

- **void clear ()**
  
  Remove every item from the browser.

- **Fl_Check_Browser (int x, int y, int w, int h, const char *l=0)**
  
  The constructor makes an empty browser.

- **void * item_at (int index) const**
  
  This method must be provided by the subclass to return the item for the specified index.

- **void item_swap (int ia, int ib)**
  
  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.

- **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.

Protected Member Functions

- **int handle (int)**
  
  Handles the event within the normal widget bounding box.

- **void item_draw (void *, int, int, int, int) const**
  
  This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.

- **void * item_first () const**
This method must be provided by the subclass to return the first item in the list.
• int item_height (void *) const
  This method must be provided by the subclass to return the height of item in pixels.
• void * item_next (void *) const
  This method must be provided by the subclass to return the item in the list after item.
• void * item_prev (void *) const
  This method must be provided by the subclass to return the item in the list before item.
• void item_select (void *, int)
  This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.
• int item_selected (void *) const
  This method must be implemented by the subclass if it supports multiple selections; returns the selection state for item.
• const char * item_text (void *item) const
  This optional method returns a string (label) that may be used for sorting.
• int item_width (void *) const
  This method must be provided by the subclass to return the width of the item in pixels.

Additional Inherited Members

31.14.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

31.14.2 Constructor & Destructor Documentation

31.14.2.1 Fl_Check_Browser()

Fl_Check_Browser::Fl_Check_Browser {
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )

The constructor makes an empty browser.

31.14.2.2 ~Fl_Check_Browser()

Fl_Check_Browser::~Fl_Check_Browser ( ) [inline]

The destructor deletes all list items and destroys the browser.
31.14.3 Member Function Documentation

31.14.3.1 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)
```

31.14.3.2 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.

31.14.3.3 check_all()

```cpp
void Fl_Check_Browser::check_all ()
```

Sets all the items checked.

31.14.3.4 check_none()

```cpp
void Fl_Check_Browser::check_none ()
```

Sets all the items unchecked.

31.14.3.5 checked() [1/2]

```cpp
int Fl_Check_Browser::checked (int i) const
```

Gets the current status of item \( i \).
31.14.3.6 checked() [2/2]

```c
void Fl_Check_Browser::checked (  
    int i,  
    int b  )
```

Sets the check status of item item to b.

31.14.3.7 clear()

```c
void Fl_Check_Browser::clear ( )
```

Remove every item from the browser.

31.14.3.8 handle()

```c
int Fl_Check_Browser::handle (  
    int event ) [protected], [virtual]
```

Handles the event within the normal widget bounding box.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>The event to process.</th>
</tr>
</thead>
</table>

**Returns**

1 if event was processed, 0 if not.

Reimplemented from Fl_Browser__.

31.14.3.9 item_at()

```c
void * Fl_Check_Browser::item_at (  
    int index ) const [virtual]
```

This method must be provided by the subclass to return the item for the specified index.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>The index of the item to be returned</th>
</tr>
</thead>
</table>
Returns

The item at the specified index.

Reimplemented from Fl_Browser_.

31.14.3.10 item_first()

```cpp
void * Fl_Check_Browser::item_first ( ) const [protected], [virtual]
```

This method must be provided by the subclass to return the first item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

31.14.3.11 item_height()

```cpp
int Fl_Check_Browser::item_height ( void * item ) const [protected], [virtual]
```

This method must be provided by the subclass to return the height of item in pixels.

Allow for two additional pixels for the list selection box.

Parameters

- **in item** The item whose height is returned.

Returns

The height of the specified item in pixels.

See also

item_height(), item_width(), item_quick_height()

Implements Fl_Browser_.

Generated by Doxygen
31.14.3.12 item_next()

```cpp
void * Fl_Check_Browser::item_next (void * item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list after `item`.

See also
- `item_first()`, `item_next()`, `item_last()`, `item_prev()`

Implements `Fl_Browser_`.

31.14.3.13 item_prev()

```cpp
void * Fl_Check_Browser::item_prev (void * item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list before `item`.

See also
- `item_first()`, `item_next()`, `item_last()`, `item_prev()`

Implements `Fl_Browser_`.

31.14.3.14 item_select()

```cpp
void Fl_Check_Browser::item_select (void * item, int val) [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to `val` for the `item`.

Sets the selection state for `item`, where optional `val` is 1 (select, the default) or 0 (de-select).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to be selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>The optional selection state; 1=select, 0=de-select. The default is to select the item (1).</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Browser_`.
31.14.3.15  item_selected()

```c
text Fl_Check_Browser::item_selected ( 
    void * item ) const [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`.

The method should return 1 if `item` is selected, or 0 otherwise.

**Parameters**

- **in** `item` The item to test.

Reimplemented from `Fl_Browser__`.

31.14.3.16  item_swap()

```c
void Fl_Check_Browser::item_swap ( 
    void * a, 
    void * b ) [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.

**Parameters**

- **in** `a,b` The two items to be swapped.

Reimplemented from `Fl_Browser__`.

31.14.3.17  item_text()

```c
const char * Fl_Check_Browser::item_text ( 
    void * item ) const [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

**Parameters**

- **in** `item` The item whose label text is returned.

**Returns**

- The item's text label. (Can be NULL if blank)
31.14.3.18  item_width()

    int Fl_Check_Browser::item_width (    
      void * item ) const [protected], [virtual]

This method must be provided by the subclass to return the width of the item in pixels.
Allow for two additional pixels for the list selection box.

Parameters

- **item**: The item whose width is returned.

Returns

The width of the item in pixels.

Implements Fl_Browser_.

31.14.3.19  nchecked()

    int Fl_Check_Browser::nchecked ( ) const [inline]

Returns how many items are currently checked.

31.14.3.20  nitems()

    int Fl_Check_Browser::nitems ( ) const [inline]

Returns how many lines are in the browser.
The last line number is equal to this.

31.14.3.21  remove()

    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.

Implements Fl_Browser_.
31.14.3.22 set_checked()

void Fl_Check_Browser::set_checked (  
    int item ) [inline]

Equivalent to Fl_Check_Browser::checked(item, 1).

31.14.3.23 text()

char * Fl_Check_Browser::text (  
    int i ) const

Return a pointer to an internal buffer holding item i's text.

31.14.3.24 value()

int Fl_Check_Browser::value ( ) const

Returns the index of the currently selected item.

The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

31.15 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.
Additional Inherited Members

31.15.1 Detailed Description

A button with a "checkmark" to show its status.

![Fl_Check_Button](image)

Figure 31.5 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.

31.15.2 Constructor & Destructor Documentation

31.15.2.1 Fl_Check_Button()

```c
Fl_Check_Button::Fl_Check_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * L = 0 )
```

Creates a new **Fl_Check_Button** widget using the given position, size, and label string.

The default box type is `FL_NO_BOX`, which draws the label w/o a box right of the checkmark.

The `selection_color()` sets the color of the checkmark. Default is `FL_FOREGROUND_COLOR` (usually black).

You can use `down_box()` to change the box type of the checkmark. Default is `FL_DOWN_BOX`.

Parameters

<table>
<thead>
<tr>
<th>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_Check_Button.H
- Fl_Check_Button.cxx

Generated by Doxygen
A button that is used to pop up a menu.

```c
#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 (int v)**
  
  Sets the currently selected value using the index into the menu item array.
- **int value (const Fl_Menu_Item ∗v)**
  
  Sets the currently selected value using a pointer to menu item.

### Protected Member Functions

- **void draw ()**
  
  Draws the widget.

### Additional Inherited Members

#### 31.16.1 Detailed Description

A button that is used to pop up a menu.

![Fl_Choice](image)

*Figure 31.6 Fl_Choice*
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.

Simple example:

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

int main() {
  Fl_Window *win = new Fl_Window(300,200);
  Fl_Choice *choice = new Fl_Choice(100,10,100,25,"Choice:");
  choice->add("Zero");
  choice->add("One");
  choice->add("Two");
  choice->add("Three");
  choice->value(2); // make "Two" selected by default (zero based!)
  win->end();
  win->show();
  return Fl::run();
}
```
31.16.2 Constructor & Destructor Documentation

31.16.2.1 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 |

31.16.3 Member Function Documentation

31.16.3.1 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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
31.16.3.2 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**

<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.

31.16.3.3 value() [1/3]

```cpp
int Fl_Choice::value ( ) const [inline]
```

Gets the index of the last item chosen by the user.

The index is -1 initially.

31.16.3.4 value() [2/3]

```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.

31.16.3.5 value() [3/3]

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.

The documentation for this class was generated from the following files:

- Fl_Choice.H
- Fl_Choice.cxx

31.17 Fl_Clock Class Reference

This widget provides a round analog clock display.

#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 clock type t, position, size, and label string.

- **int handle (int)**
  
  Handles the specified event.

- **~Fl_Clock ()**
  
  The destructor removes the clock.

Additional Inherited Members

31.17.1 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(). Please see **Fl_Clock_Output** widget for applicable values.

![Figure 31.7 FL_SQUARE_CLOCK type](image)

Figure 31.7 FL_SQUARE_CLOCK type

![Figure 31.8 FL_ROUND_CLOCK type](image)

Figure 31.8 FL_ROUND_CLOCK type

See also

class Fl_Clock_Output

Generated by Doxygen
31.17.2 Constructor & Destructor Documentation

31.17.2.1 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 clock type is FL_SQUARE_CLOCK and the default boxtype is FL_UP_BOX.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

31.17.2.2 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 clock type t, position, size, and label string.

The default clock type t is FL_SQUARE_CLOCK. You can set the clock type to FL_ROUND_CLOCK or any other valid clock type. See Fl_Clock_Output widget for applicable values.

The default boxtype is FL_UP_BOX for FL_SQUARE_CLOCK and FL_NO_BOX for FL_ROUND_CLOCK, if set by the constructor. If you change the clock type with type() later you should also set the boxtype with box().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>type of clock: FL_ROUND_CLOCK or FL_SQUARE_CLOCK (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>position and size of the widget</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>
31.17.3 Member Function Documentation

31.17.3.1 handle()

```cpp
int Fl_Clock::handle (  
    int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.

**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_Widget.

The documentation for this class was generated from the following files:

- Fl_Clock.H
- Fl_Clock.cxx

31.18 Fl_Clock_Output Class Reference

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

```cpp
#include <Fl_Clock.H>
```
Inheritance diagram for Fl_Clock_Output:

```
Fl_Widget
    Fl_Clock_Output
    Fl_Clock
    Fl_Round_Clock
```

Public Member Functions

- **Fl_Clock_Output** (int X, int Y, int W, int H, const char ∗L=0)
  
  Create a new Fl_Clock_Output widget with the given position, size and label.

- **int hour () const**
  
  Returns the displayed hour (0 to 23).

- **int minute () const**
  
  Returns the displayed minute (0 to 59).

- **int second () const**
  
  Returns the displayed second (0 to 60, 60=leap second).

- **int shadow () const**
  
  Returns the shadow drawing mode of the hands.

- **void shadow (int mode)**
  
  Sets the shadow drawing mode of the hands.

- **void value (ulong v)**
  
  Set the displayed time.

- **void value (int H, int m, int s)**
  
  Set the displayed time.

- **ulong value () const**
  
  Returns the displayed time.

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.

Additional Inherited Members

31.18.1 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.
Values for clock `type()` (#include `<FL/Clock.H>`):

```
#define FL_SQUARE_CLOCK 0 // Square Clock variant
#define FL_ROUND_CLOCK 1 // Round Clock variant
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK // An analog clock is square
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK // Not yet implemented
```

### 31.18.2 Constructor & Destructor Documentation

#### 31.18.2.1 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 clock type is `FL_SQUARE_CLOCK` and the default boxtype is `FL_UP_BOX`.

**Parameters**

- `in X, Y, W, H` position and size of the widget
- `in L` widget label, default is no label

Generated by Doxygen
31.18.3 Member Function Documentation

31.18.3.1 draw()

```cpp
void Fl_Clock_Output::draw (  
    int X,  
    int Y,  
    int W,  
    int H ) [protected]
```

Draw clock with the given position and size.

Parameters

| in  | $X,Y,W,H$ | position and size |

31.18.3.2 hour()

```cpp
int Fl_Clock_Output::hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).

See also

- `value()`, `minute()`, `second()`

31.18.3.3 minute()

```cpp
int Fl_Clock_Output::minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).

See also

- `value()`, `hour()`, `second()`
31.18.3.4 second()

int Fl_Clock_Output::second ( ) const [inline]

Returns the displayed second (0 to 60, 60=leap second).

See also
    value(), hour(), minute()

31.18.3.5 shadow() [1/2]

int Fl_Clock_Output::shadow ( ) const [inline]

Returns the shadow drawing mode of the hands.

Returns
    shadow drawing mode of the hands

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no shadows</td>
</tr>
<tr>
<td>1</td>
<td>draw shadows of hands (default)</td>
</tr>
</tbody>
</table>

31.18.3.6 shadow() [2/2]

void Fl_Clock_Output::shadow ( int mode ) [inline]

Sets the shadow drawing mode of the hands.

Enables (1) or disables (0) drawing the hands with shadows.

Values except 0 and 1 are reserved for future extensions and yield undefined behavior.

The default is to draw the shadows (1).

Parameters

| in | mode | 1 = shadows (default), 0 = no shadows |
31.18.3.7 value() [1/3]

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()

31.18.3.8 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()

31.18.3.9 value() [3/3]

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)

The documentation for this class was generated from the following files:

- Fl_Clock.H
- Fl_Clock.cxx
31.19  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:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Color_Chooser
```

Public Member Functions

- **double b () const**
  Returns the current blue value.

- **Fl_Color_Chooser (int X, int Y, int W, int H, const char *L=0)**
  Creates a new Fl_Color_Chooser widget using the given position, size, and label string.

- **double g () const**
  Returns the current green value.

- **int handle (int e)**
  Handles all events received by this widget.

- **int hsv (double H, double S, double V)**
  Set the hsv values.

- **double hue () const**
  Returns the current hue.

- **int mode ()**
  Returns which Fl_Color_Chooser variant is currently active.

- **void mode (int newMode)**
  Set which Fl_Color_Chooser variant is currently active.

- **double r () const**
  Returns the current red value.

- **int rgb (double R, double G, double B)**
  Sets the current rgb color values.

- **double saturation () const**
  Returns the saturation.

- **double value () const**
  Returns the value/brightness.

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.
Related Functions

(Note that these are not member functions.)

- `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

31.19.1 Detailed Description

The Fl_Color_Chooser widget provides a standard RGB color chooser.

![Fl_Color_Chooser](image)

Figure 31.11 fl_color_chooser()

You can place any number of the widgets into a panel of your own design. The diagram shows the widget as part of a color chooser dialog created by the fl_color_chooser() function. The Fl_Color_Chooser widget contains the hue box, value slider, and rgb input fields from the above diagram (it does not have the color chips or the Cancel or OK buttons). The callback is done every time the user changes the rgb value. It is not done if they move the hue control in a way that produces the same rgb value, such as when saturation or value is zero.

The fl_color_chooser() function pops up a window to let the user pick an arbitrary RGB color. They can pick the hue and saturation in the "hue box" on the left (hold down CTRL to just change the saturation), and the brightness using the vertical slider. Or they can type the 8-bit numbers into the RGB Fl_Value_Input fields, or drag the mouse across them to adjust them. The pull-down menu lets the user set the input fields to show RGB, HSV, or 8-bit RGB (0 to 255).

The user can press CTRL-C to copy the currently selected color value as text in RGB hex format with leading zeroes to the clipboard, for instance FL_GREEN would be '00FF00' (since FLTK 1.4.0).

fl_color_chooser() returns non-zero if the user picks ok, and updates the RGB values. If the user picks cancel or closes the window this returns zero and leaves RGB unchanged.

If you use the color chooser on an 8-bit screen, it will allocate all the available colors, leaving you no space to exactly represent the color the user picks! You can however use fl_rectf() to fill a region with a simulated color using dithering.
31.19 Fl_Color_Chooser Class Reference

31.19.2 Constructor & Destructor Documentation

31.19.2.1 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

<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>

31.19.3 Member Function Documentation

31.19.3.1 b()

```cpp
double Fl_Color_Chooser::b ( ) const [inline]
```

Returns the current blue value.

\[ 0 \leq b \leq 1. \]

31.19.3.2 g()

```cpp
double Fl_Color_Chooser::g ( ) const [inline]
```

Returns the current green value.

\[ 0 \leq g \leq 1. \]
31.19.3.3 handle()

```cpp
int Fl_Color_Chooser::handle ( int e ) [virtual]
```

Handles all events received by this widget.

This specific `handle()` method processes the standard 'copy' function as seen in other input widgets. It copies the current color value to the clipboard as a string in RGB format ('RRGGBB').

This format is independent of the `Fl_Color_Chooser` display format setting. No other formats are supplied.

The keyboard events handled are:

- ctrl-c
- ctrl-x
- ctrl-Insert

All other events are processed by the parent class `Fl_Group`.

This enables the user to choose a color value, press `ctrl-c` to copy the value to the clipboard and paste it into a color selection widget in another application window or any other text input (e.g. a preferences dialog or an editor).

**Note**

Keyboard event handling by the current focus widget has priority, hence moving the focus to one of the buttons or selecting text in one of the input widgets effectively disables this special method.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>e</th>
<th>current event</th>
</tr>
</thead>
</table>

**Returns**

1 if event has been handled, 0 otherwise

See also

- `Fl_Group::handle(int)`

Reimplemented from `Fl_Widget`.

31.19.3.4 hsv()

```cpp
int Fl_Color_Chooser::hsv ( double H, double S, double V )
```

Set the hsv values.

The passed values are clamped (or for hue, modulus 6 is used) to get legal values. Does not do the callback.
Parameters

| in  | H,S,V | color components |

Returns

1 if a new hsv value was set, 0 if the hsv value was the previous one.

31.19.3.5 hsv2rgb()

```cpp
void Fl_Color_Chooser::hsv2rgb ( 
  double H,
  double S,
  double V,
  double & R,
  double & G,
  double & B ) [static]
```

This static method converts HSV colors to RGB colorspace.

Parameters

<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>

31.19.3.6 hue()

```cpp
double Fl_Color_Chooser::hue ( ) const [inline]
```

Returns the current hue.

0 <= hue < 6. Zero is red, one is yellow, two is green, etc. This value is convenient for the internal calculations - some other systems consider hue to run from zero to one, or from 0 to 360.

31.19.3.7 mode() [1/2]

```cpp
int Fl_Color_Chooser::mode ( ) [inline]
```

Returns which Fl_Color_Chooser variant is currently active.

Returns

color modes are rgb(0), byte(1), hex(2), or hsv(3)
31.19.3.8 mode() [2/2]

```cpp
void Fl_Color_Chooser::mode (  
    int newMode  )
```

Set which Fl_Color_Chooser variant is currently active.

**Parameters**

| in | newMode | color modes are rgb(0), byte(1), hex(2), or hsv(3) |

31.19.3.9 r()

```cpp
double Fl_Color_Chooser::r ( ) const [inline]
```

Returns the current red value.

$$0 \leq r \leq 1.$$  

31.19.3.10 rgb()

```cpp
int Fl_Color_Chooser::rgb (  
    double R,  
    double G,  
    double B  )
```

Sets the current rgb color values.

Does not do the callback. Does not clamp (but out of range values will produce psychedelic effects in the hue selector).

**Parameters**

| in | R,G,B | color components. |

Returns

1 if a new rgb value was set, 0 if the rgb value was the previous one.

31.19.3.11 rgb2hsv()

```cpp
void Fl_Color_Chooser::rgb2hsv (  
    double R,  
    double G,  
```
This static method converts RGB colors to HSV colorspace.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>R,G,B</th>
<th>color components</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>H,S,V</td>
<td>color components</td>
</tr>
</tbody>
</table>

31.19.3.12 saturation()

double Fl_Color_Chooser::saturation ( ) const [inline]

Returns the saturation.

0 \leq saturation \leq 1.

31.19.3.13 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

31.20 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

#include <Fl_Copy_Surface.H>

Inheritance diagram for Fl_Copy_Surface:
Public Member Functions

- virtual void draw_decorated_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)
  Draws a window with its title bar and frame if any.
- Fl_Copy_Surface (int w, int h)
  the constructor
- int h ()
  Returns the pixel height of the copy surface.
- virtual bool is_current ()
  Is this surface the current drawing surface?
- void origin (int ∗x, int ∗y)
  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y)
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int ∗w, int ∗h)
  Computes the width and height of the drawable area of the drawing surface.
- void set_current ()
  Make this surface the current drawing surface.
- int w ()
  Returns the pixel width of the copy surface.
- ~Fl_Copy_Surface ()
  the destructor

Protected Member Functions

- void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.
- void untranslate ()
  Undoes the effect of a previous translate() call.

Additional Inherited Members

31.20.1 Detailed Description

Supports copying of graphical data to the clipboard.

After creation of an Fl_Copy_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the clipboard. It's possible to draw widgets (using Fl_Copy_Surface::draw() ) or to use any of the Drawing functions or the Color & Font functions. Finally, delete the Fl_Copy_Surface object to load the clipboard with the graphical data.

Usage example:

```c
Fl_Widget ∗w = ...; // a widget you want to copy to the clipboard
Fl_Copy_Surface ∗copy_surf = new Fl_Copy_Surface(w->w(), w->h()); // create an Fl_Copy_Surface object
Fl_Surface_Device::push_current(copy_surf); // direct graphics requests to the clipboard
fl_color(Fl_WHITE); fl_rectf(0, 0, w->w(), w->h()); // draw a white background
copy_surf->draw(w); // draw the w widget in the clipboard
Fl_Surface_Device::pop_current(); // direct graphics requests back to their previous destination
delete copy_surf; // after this, the clipboard is loaded
```
Platform details:

- **Windows**: Transparent RGB images copy without transparency. The graphical data are copied to the clipboard in two formats: 1) as an 'enhanced metafile'; 2) as a color bitmap. Applications to which the clipboard content can be pasted can use the format that suits them best.

- **Mac OS**: The graphical data are copied to the clipboard (a.k.a. pasteboard) in two 'flavors': 1) in vectorial form as PDF data; 2) in bitmap form as a TIFF image. Applications to which the clipboard content is pasted can use the flavor that suits them best.

- **X11**: the graphical data are copied to the clipboard as an image in BMP format.

### 31.20.2 Constructor & Destructor Documentation

#### 31.20.2.1 Fl_Copy_Surface()

Fl_Copy_Surface::Fl_Copy_Surface (  
    int w,  
    int h  )

the constructor

Parameters

| w,h | Width and height of the drawing surface in FLTK units |

### 31.20.3 Member Function Documentation

#### 31.20.3.1 draw_decorated_window()

void Fl_Copy_Surface::draw_decorated_window (  
    Fl_Window * win,  
    int win_offset_x = 0,  
    int win_offset_y = 0  )  [virtual]

Draws a window with its title bar and frame if any.

`win_offset_x` and `win_offset_y` are optional coordinates of where to position the window top left. Equivalent to `draw()` if `win` is a subwindow or has no border. Use `Fl_Window::decorated_w()` and `Fl_Window::decorated_h()` to get the size of the framed window.

Reimplemented from `Fl_Widget_Surface`.

---

Generated by Doxygen
31.20.3.2 origin() [1/2]

```cpp
void Fl_Copy_Surface::origin (  
    int * x,  
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

- **out x,y**: If non-null, `x` and `y` are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from `Fl_Widget_Surface`.

31.20.3.3 origin() [2/2]

```cpp
void Fl_Copy_Surface::origin (  
    int x,  
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters

- **in x,y**: Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from `Fl_Widget_Surface`.

31.20.3.4 printable_rect()

```cpp
int Fl_Copy_Surface::printable_rect (  
    int * w,  
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

Returns

- `0` if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`. 

Generated by Doxygen
31.20.3.5 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. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

Reimplemented from `Fl_Surface_Device`.

31.20.3.6 translate()

```cpp
void Fl_Copy_Surface::translate ( int x, int y ) [protected], [virtual]
```

Translates the current graphics origin accounting for the current rotation.

Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following files:

- `Fl_Copy_Surface.H`
- `Fl_Copy_Surface.cxx`

31.21 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.

```cpp
#include <Fl_Counter.H>
```

Inheritance diagram for `Fl_Counter`:

```
Fl_Widget
   ` ` Fl_Valuator
       ` ` Fl_Counter
           ` ` Fl_Simple_Counter
```

Generated by Doxygen
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.

- **void step** (double a, double b)
  
  Sets the increments for the normal and large step buttons.

- **void step** (double a)
  
  Sets the increment for the normal step buttons.

- **double step () const**
  
  Returns the increment for normal 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.

Protected Member Functions

- **void draw ()**
  
  Draws the widget.

Additional Inherited Members

31.21.1 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.

![Figure 31.12 Fl_Counter](image)

**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.

### 31.21.2 Constructor & Destructor Documentation

#### 31.21.2.1 `Fl_Counter()`

```cpp
Fl_Counter::Fl_Counter (int X, int Y, int W, int H, const char ∗L = 0)
```

Creates a new `Fl_Counter` widget using the given position, size, and label string.

The default type is `FL_NORMAL_COUNTER`.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>X,Y,W,H</code></td>
<td>position and size of the widget</td>
</tr>
<tr>
<td>in <code>L</code></td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 31.21.3 Member Function Documentation
31.21.3.1 draw()

```cpp
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 `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

31.21.3.2 handle()

```cpp
int Fl_Counter::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><code>int event</code></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`. 
31.21.3.3  lstep()

void Fl_Counter::lstep (double a) [inline]

Sets the increment for the large step buttons.

The default value is 1.0.

Parameters

| in | a | large step increment. |

31.21.3.4  step() [1/2]

void Fl_Counter::step (double a, double b) [inline]

Sets the increments for the normal and large step buttons.

Parameters

| in | a,b | normal and large step increments. |

31.21.3.5  step() [2/2]

void Fl_Counter::step (double a) [inline]

Sets the increment for the normal step buttons.

Parameters

| in | a | normal step increment. |

The documentation for this class was generated from the following files:

- Fl_Counter.H
- Fl_Counter.cxx

31.22  Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.
#include <Fl_Device.H>

Inheritance diagram for Fl_Device_Plugin:

```
 Fl_Device_Plugin
    |          |
    |          | Fl_Plugin
```

Public Member Functions

- **Fl_Device_Plugin (const char *pluginName)**
  
  The constructor.

- **virtual const char * klass ()**
  
  Returns the class name.

- **virtual const char * name ()=0**
  
  Returns the plugin name.

- **virtual int print (Fl_Widget *w)=0**
  
  Prints a widget.

- **virtual Fl_RGB_Image * rectangle_capture (Fl_Widget *widget, int x, int y, int w, int h)=0**
  
  Captures a rectangle of a widget as an image.

Static Public Member Functions

- **static Fl_Device_Plugin * opengl_plugin ()**
  
  Returns the OpenGL plugin.

### 31.22.1 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.

### 31.22.2 Member Function Documentation
31.22.2.1 rectangle_capture()

```cpp
virtual Fl_RGB_Image* Fl_Device_Plugin::rectangle_capture (
    Fl_Widget * widget,
    int x,                  
    int y,                   
    int w,                   
    int h )                  
    { pure virtual }
```

Captures a rectangle of a widget as an image.

Returns

The captured pixels as an RGB image

The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

31.23 Fl_Dial Class Reference

The **Fl_Dial** widget provides a circular dial to control a single floating point value.

```c
#include <Fl_Dial.H>
```

Inheritance diagram for Fl_Dial:

```
Fl_Widget

Fl_Valuator

Fl_Dial

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.*

Generated by Doxygen
Protected Member Functions

- void **draw** (int X, int Y, int W, int H)
  
  Draws dial at given position and size.
- void **draw** ()
  
  Draws dial at current 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.

Additional Inherited Members

31.23.1 Detailed Description

The Fl_Dial widget provides a circular dial to control a single floating point value.

![Figure 31.13 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.

31.23.2 Constructor & Destructor Documentation

31.23.2.1 Fl_Dial()

```cpp
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.
31.23.3 Member Function Documentation

31.23.3.1 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.

31.23.3.2 draw()

```
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
```

31.23.3.3 handle()

```
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.
```

The documentation for this class was generated from the following files:

- Fl_Dial.H
- Fl_Dial.cxx
### 31.24  **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_Surface_Device
  Fl_Display_Device
```

#### Static Public Member Functions

- `static Fl_Display_Device * display_device ()`
  
  Returns a pointer to the unique display device.

#### Additional Inherited Members

### 31.24.1  Detailed Description

A display to which the computer can draw.

When the program begins running, an object of class `Fl_Display_Device` has been created and made the current drawing surface.

The documentation for this class was generated from the following files:

- `Fl_Device.H`
- `Fl_Device.cxx`

### 31.25  **Fl_Double_Window Class Reference**

The `Fl_Double_Window` provides a double-buffered window.

```cpp
#include <Fl_Double_Window.H>
```

Inheritance diagram for Fl_Double_Window:

```
Fl_Widget
  Fl_Group
    Fl_Window
      Fl_Double_Window
        Fl_Cairo_Window
        Fl_Overlay_Window
```

---

Generated by Doxygen
Public Member Functions

- virtual Fl_Double_Window * as_double_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Double_Window (int W, int H, const char * l=0)
  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- Fl_Double_Window (int X, int Y, int W, int H, const char * l=0)
  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char *label = 0)
- void flush ()
  Forces the window to be drawn, this window is also made current and calls draw().
- 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.

Additional Inherited Members

31.25.1 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:

```cpp
Fl::visual (FL_DOUBLE|FL_INDEX)
```

This makes sure you can use Xdbe on servers where double buffering does not exist for every visual.

31.25.2 Constructor & Destructor Documentation

31.25.2.1 ~Fl_Double_Window()

```cpp
Fl_Double_Window::~Fl_Double_Window ()
```

The destructor also deletes all the children.

This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.
31.25.3 Member Function Documentation

31.25.3.1 flush()

```cpp
void Fl_Double_Window::flush ( ) [virtual]
```
Forces the window to be drawn, this window is also made current and calls `draw()`.

Reimplemented from `Fl_Window`.

Reimplemented in `Fl_Overlay_Window`.

31.25.3.2 hide()

```cpp
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 `Fl_Window`.

Reimplemented in `Fl_Overlay_Window`.

31.25.3.3 resize()

```cpp
void Fl_Double_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_Window`.

Reimplemented in `Fl_Overlay_Window`. 

---

Generated by Doxygen
31.25.3.4 show()

```
void Fl_Double_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`.

Reimplemented in `Fl_Overlay_Window`.

The documentation for this class was generated from the following files:

- `Fl_Double_Window.H`
- `Fl_Double_Window.cxx`

### 31.26 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()`
31.26.1 Detailed Description

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

```c
class MyClass {
    Fl_Group group;
    Fl_Button button_in_group;
    Fl_End end;
    Fl_Button button_outside_group;
    MyClass();
}
MyClass::MyClass() :
    group(10,10,100,100),
    button_in_group(20,20,60,30),
    end(),
    button_outside_group(10,120,60,30) {
   [..ctor code..]
}
```

The documentation for this class was generated from the following file:

- Fl_Group.H

31.27 Fl_EPS_File_Surface Class Reference

Encapsulated PostScript drawing surface.

```c
#include <Fl_PostScript.H>
```

Inheritance diagram for Fl_EPS_File_Surface:

```
Fl_Surface_Device
  Fl_Widget_Surface
    Fl_EPS_File_Surface
```

Public Member Functions

- int close ()
  Closes using fclose() the underlying FILE pointer.
- FILE * file ()
  Returns the underlying FILE pointer.
- Fl_EPS_File_Surface (int width, int height, FILE *eps, Fl_Color background=FL_WHITE)
  Constructor.
- virtual void origin (int x, int y)
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- virtual void origin (int *px, int *py)
  Computes the coordinates of the current origin of graphics functions.
- virtual int printable_rect (int *w, int *h)
  Computes the width and height of the drawable area of the drawing surface.
- virtual void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.
- virtual void untranslate ()
  Undoes the effect of a previous translate() call.
- Fl_EPS_File_Surface ()
  Destructor.
Protected Member Functions

- Fl_PostScript_Graphics_Driver * driver ()
  
  Returns the PostScript driver of this drawing surface.

Additional Inherited Members

31.27.1 Detailed Description

Encapsulated PostScript drawing surface.

This drawing surface allows to store any FLTK graphics in vectorial form in an "Encapsulated PostScript" file.

Usage example:

```c
Fl_Window *win = ...; // Window to draw to an .eps file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *eps = fl_fopen("/path/to/mywindow.eps", "w");
if (eps) {
  Fl_EPS_File_Surface *surface = new Fl_EPS_File_Surface(ww, wh, eps, win->color());
  Fl_Surface_Device::push_current(surface);
  surface->draw_decorated_window(win);
  Fl_Surface_Device::pop_current();
  delete surface; // the .eps file is not complete until the destructor was run
  fclose(eps);
}
```

31.27.2 Constructor & Destructor Documentation

31.27.2.1 Fl_EPS_File_Surface()

```c
Fl_EPS_File_Surface::Fl_EPS_File_Surface (  
  int width,
  int height,
  FILE * eps,
  Fl_Color background = FL_WHITE )
```

Constructor.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width, height</td>
<td>Width and height of the EPS drawing area</td>
</tr>
<tr>
<td>eps</td>
<td>A writable FILE pointer where the Encapsulated PostScript data will be sent</td>
</tr>
<tr>
<td>background</td>
<td>Color expected to cover the background of the EPS drawing area. This parameter affects only the drawing of transparent Fl_RGB_Image objects: transparent areas of RGB images are blended with the background color.</td>
</tr>
</tbody>
</table>
### 31.27.2.2 ∼Fl_EPS_File_Surface()

Fl_EPS_File_Surface::∼Fl_EPS_File_Surface ( )

Destructor.

The underlying FILE pointer remains open after destruction of the `Fl_EPS_File_Surface` object unless `close()` was called.

### 31.27.3 Member Function Documentation

#### 31.27.3.1 close()

```cpp
int Fl_EPS_File_Surface::close ()
```

Closes using fclose() the underlying FILE pointer.

The only operation possible with the `Fl_EPS_File_Surface` object after calling `close()` is its destruction.

#### 31.27.3.2 driver()

```cpp
Fl_PostScript_Graphics_Driver* Fl_EPS_File_Surface::driver () [inline], [protected]
```

Returns the PostScript driver of this drawing surface.

#### 31.27.3.3 origin() [1/2]

```cpp
virtual void Fl_EPS_File_Surface::origin ( int x, int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters:

| in | x, y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from `Fl_Widget_Surface`. 

Generated by Doxygen
31.27.3.4 origin() [2/2]

virtual void Fl_EPS_File_Surface::origin(
    int * x,
    int * y) [virtual]

Computes the coordinates of the current origin of graphics functions.

Parameters

out x, y
If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

31.27.3.5 printable_rect()

virtual int Fl_EPS_File_Surface::printable_rect(
    int * w,
    int * h) [virtual]

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

31.27.3.6 translate()

virtual void Fl_EPS_File_Surface::translate(
    int x,
    int y) [virtual]

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following file:

- Fl_PostScript.H

Generated by Doxygen
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_Browser
|   |
|   Fl_File_Browser
```

Public Types
- enum { FILE, DIRECTORY }

Public Member Functions
- int filetype () const
  Sets or gets the file browser type, FILE or DIRECTORY.
- void filetype (int t)
  Sets or gets the file browser type, FILE or DIRECTORY.
- void filter (const char *pattern)
  Sets or gets the filename filter.
- const char * filter () const
  Sets or gets the filename filter.
- Fl_File_Browser (int, int, int, int, const char *=0)
  The constructor creates the Fl_File_Browser widget at the specified position and size.
- uchar iconsize () const
  Sets or gets the size of the icons.
- void iconsize (uchar s)
  Sets or gets the size of the icons.
- int load (const char *directory, Fl_File_Sort_F *sort=fl_numericsort)
  Loads the specified directory into the browser.
- Fl_Fontsize textsize () const
- void textsize (Fl_Fontsize s)

Additional Inherited Members

31.28.1 Detailed Description

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.
31.28.2 Constructor & Destructor Documentation

31.28.2.1 Fl_File_Browser()

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.

31.28.3 Member Function Documentation

31.28.3.1 filetype()[1/2]

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.

31.28.3.2 filetype()[2/2]

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.

31.28.3.3 filter()[1/2]

void Fl_File_Browser::filter (  
   const char * pattern 
)

Sets or gets the filename filter.

The pattern matching uses the fl_filename_match() function in FLTK.
Sets or gets the filename filter.

The pattern matching uses the `fl_filename_match()` function in FLTK.

Sets or gets the size of the icons.

The default size is 20 pixels.

Sets or gets the size of the icons.

The default size is 20 pixels.

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`

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 an extra widget at the bottom of the 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.
- void color (Fl_Color c)
  Sets the background color of the Fl_File_Browser list.
- Fl_Color color ()
  Gets the background color of the Fl_File_Browser list.
- int count ()
  Returns the number of selected files.
- void directory (const char *d)
  Sets the current directory.
- char * directory ()
  Gets the current directory.
- void filter (const char *p)
  Sets the current filename filter patterns.
- const char * filter ()
  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.
- void iconsize (uchar s)
  Sets the size of the icons in the Fl_File_Browser.
- uchar iconsize ()
  Gets the size of the icons in the Fl_File_Browser.
- void label (const char *l)
  Sets the title bar text for the Fl_File_Chooser.
- const char * label ()
  Gets the title bar text for the Fl_File_Chooser.
- void ok_label (const char *l)
  Sets the label for the "ok" button in the Fl_File_Chooser.
- const char * ok_label ()
  Gets the label for the "ok" button in the Fl_File_Chooser.
- void preview (int e)
  Enable or disable the preview tile.
- int preview () const
  Returns the current state of the preview box.
- 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**().
- void **textcolor** (Fl_Color c)
  
  Sets the current **Fl_File_Browser** text color.
- Fl_Color **textcolor** ()
  
  Gets the current **Fl_File_Browser** text color.
- void **textfont** (Fl_Font f)
  
  Sets the current **Fl_File_Browser** text font.
- Fl_Font **textfont** ()
  
  Gets the current **Fl_File_Browser** text font.
- void **textsize** (Fl_Fontsize s)
  
  Sets the current **Fl_File_Browser** text size.
- Fl_Fontsize **textsize** ()
  
  Gets the current **Fl_File_Browser** text size.
- void **type** (int t)
  
  Sets the current type of **Fl_File_Chooser**.
- int **type** ()
  
  Gets 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.
- const char **∗value** (int f=1)
  
  Gets the current value of the selected file(s).
- void **value** (const char **∗filename**)
  
  Sets the current value of the selected file.
- 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 = Fl::system_driver()->filesystems_label()
  [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 Functions

(Note that these are not member functions.)

- char ∗ fl_dir_chooser (const char ∗message, const char ∗fname, int relative)
  Shows a file chooser dialog and gets a directory.
- char ∗ fl_file_chooser (const char ∗message, const char ∗pat, const char ∗fname, int relative)
  Shows a file chooser dialog and gets a filename.
- void fl_file_chooser_callback (void (∗cb)(const char ∗))
  Set the file chooser callback.
- void fl_file_chooser_ok_label (const char ∗l)
  Set the "OK" button label.
31.29.1 Detailed Description

The **Fl_File_Chooser** widget displays a standard file selection dialog that supports various selection modes.

![Figure 31.14 Fl_File_Chooser](image)

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 Windows platform to open files with a non-ASCII name).

The **Fl_File_Chooser** class also exports several static values that may be used to localize or customize the appearance of all file chooser dialogs:

<table>
<thead>
<tr>
<th>Member</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_favorites_label</td>
<td>&quot;Add to Favorites&quot;</td>
</tr>
<tr>
<td>all_files_label</td>
<td>&quot;All Files (+)*&quot;</td>
</tr>
<tr>
<td>custom_filter_label</td>
<td>&quot;Custom Filter&quot;</td>
</tr>
<tr>
<td>existing_file_label</td>
<td>&quot;Please choose an existing file!&quot;</td>
</tr>
<tr>
<td>favorites_label</td>
<td>&quot;Favorites&quot;</td>
</tr>
<tr>
<td>filename_label</td>
<td>&quot;Filename:&quot;</td>
</tr>
<tr>
<td>filesystems_label</td>
<td>&quot;My Computer&quot; (Windows) &quot;File Systems&quot; (all others)</td>
</tr>
<tr>
<td>hidden_label</td>
<td>&quot;Show hidden files:&quot;</td>
</tr>
<tr>
<td>manage_favorites_label</td>
<td>&quot;Manage Favorites&quot;</td>
</tr>
<tr>
<td>new_directory_label</td>
<td>&quot;New Directory?&quot;</td>
</tr>
<tr>
<td>new_directory_tooltip</td>
<td>&quot;Create a new directory.&quot;</td>
</tr>
<tr>
<td>preview_label</td>
<td>&quot;Preview&quot;</td>
</tr>
<tr>
<td>save_label</td>
<td>&quot;Save&quot;</td>
</tr>
<tr>
<td>show_label</td>
<td>&quot;Show:&quot;</td>
</tr>
<tr>
<td>sort</td>
<td>fl_numericsort</td>
</tr>
</tbody>
</table>
The Fl_File_Chooser::sort member specifies the sort function that is used when loading the contents of a directory and can be customized at run-time.

The Fl_File_Chooser class also exports the Fl_File_Chooser::newButton and Fl_File_Chooser::previewButton widgets so that application developers can control their appearance and use.

### 31.29.2 Constructor & Destructor Documentation

#### 31.29.2.1 Fl_File_Chooser()

```c++
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.

#### 31.29.2.2 ~Fl_File_Chooser()

```c++
Fl_File_Chooser::~Fl_File_Chooser ( )
```

Destroys the widget and frees all memory used by it.
31.29.3 Member Function Documentation

31.29.3.1 add_extra()

`Fl_Widget * Fl_File_Chooser::add_extra ( Fl_Widget * extra )`

Adds an extra widget at the bottom of the `Fl_File_Chooser` window.

You can use any `Fl_Widget` or `Fl_Group`. If you use an `Fl_Group`, set its (x, y) coordinates to (0, 0) and position its children relative to (0, 0) inside the `Fl_Group` container widget. Make sure that all child widgets of the `Fl_Group` are entirely included inside the bounding box of their parents, i.e. the `Fl_Group` widget, and the `Fl_File_Chooser` window, respectively.

**Note**

The width of the `Fl_File_Chooser` window is an undocumented implementation detail and may change in the future.

If `extra` is NULL any previous extra widget is removed.

**Parameters**

*in* `extra` Custom widget or group to be added to the `Fl_File_Chooser` window.

**Returns**

Pointer to previous extra widget or NULL if not set previously.

**Note**

`Fl_File_Chooser` does **not** delete the extra widget in its destructor! The extra widget is removed from the `Fl_File_Chooser` window before the `Fl_File_Chooser` widget gets destroyed. To prevent memory leakage, don’t forget to delete unused extra widgets.

31.29.3.2 color() [1/2]

`void Fl_File_Chooser::color ( Fl_Color c )`

Sets the background color of the `Fl_File_Browser` list.
### 31.29.3.3 color()

```cpp
Fl_Color Fl_File_Chooser::color()
```

Gets the background color of the Fl_File_Browser list.

### 31.29.3.4 count()

```cpp
int Fl_File_Chooser::count()
```

Returns the number of selected files.

### 31.29.3.5 directory()

```cpp
void Fl_File_Chooser::directory(const char * d)
```

Sets the current directory.

### 31.29.3.6 directory()

```cpp
const char * Fl_File_Chooser::directory()
```

Gets the current directory.

### 31.29.3.7 filter()

```cpp
void Fl_File_Chooser::filter(const char * p)
```

Sets the current filename filter patterns.

The filter patterns use `fl_filename_match()`. Multiple patterns can be used by separating them with tabs, like `"*.jpg	*.png	*.gif	*"`. In addition, you can provide human-readable labels with the patterns inside parenthesis, like `"JPEG Files (*.jpg)	PNG Files (*.png)	GIF Files (*.gif)	All Files (*)"`. Use `filter(NULL)` to show all files.

---

Generated by Doxygen
31.29.3.8 filter() [2/2]

const char * Fl_File_Chooser::filter( )

Gets the current filename filter patterns.

31.29.3.9 filter_value() [1/2]

int Fl_File_Chooser::filter_value( )

Gets the current filename filter selection.

31.29.3.10 filter_value() [2/2]

void Fl_File_Chooser::filter_value( int f )

Sets the current filename filter selection.

31.29.3.11 hide()

void Fl_File_Chooser::hide( )

Hides the Fl_File_Chooser window.

31.29.3.12 iconsize() [1/2]

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().

31.29.3.13 iconsize() [2/2]

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().
### 31.29.3.14 label() [1/2]

```cpp
void Fl_File_Chooser::label (const char * l)
```

Sets the title bar text for the Fl_File_Chooser.

### 31.29.3.15 label() [2/2]

```cpp
const char * Fl_File_Chooser::label ()
```

Gets the title bar text for the Fl_File_Chooser.

### 31.29.3.16 preview() [1/2]

```cpp
void Fl_File_Chooser::preview (int e)
```

Enable or disable the preview tile.
1 = enable preview, 0 = disable preview.

### 31.29.3.17 preview() [2/2]

```cpp
int Fl_File_Chooser::preview () const [inline]
```

Returns the current state of the preview box.

### 31.29.3.18 rescan()

```cpp
void Fl_File_Chooser::rescan ()
```

Reloads the current directory in the Fl_File_Browser.

### 31.29.3.19 show()

```cpp
void Fl_File_Chooser::show ()
```

Shows the Fl_File_Chooser window.
shown()

```cpp
int Fl_File_Chooser::shown()
```

Returns non-zero if the file chooser main window `show()` has been called, but not `hide()`.

See also:
- `Fl_Window::shown()`


textcolor()[1/2]

```cpp
void Fl_File_Chooser::textcolor(Fl_Color c)
```

Sets the current `Fl_File_Browser` text color.


textcolor()[2/2]

```cpp
Fl_Color Fl_File_Chooser::textcolor()
```

Gets the current `Fl_File_Browser` text color.


textfont()[1/2]

```cpp
void Fl_File_Chooser::textfont(Fl_Font f)
```

Sets the current `Fl_File_Browser` text font.


textfont()[2/2]

```cpp
Fl_Font Fl_File_Chooser::textfont()
```

Gets the current `Fl_File_Browser` text font.
31.29.3.25  **textsize()** [1/2]

```cpp
void Fl_File_Chooser::textsize (Fl_Fontsize s)
```

Sets the current **Fl_File_Browser** text size.

31.29.3.26  **textsize()** [2/2]

```cpp
Fl_Fontsize Fl_File_Chooser::textsize ( )
```

Gets the current **Fl_File_Browser** text size.

31.29.3.27  **type()** [1/2]

```cpp
void Fl_File_Chooser::type (int t)
```

Sets the current type of **Fl_File_Chooser**.

31.29.3.28  **type()** [2/2]

```cpp
int Fl_File_Chooser::type ( )
```

Gets the current type of **Fl_File_Chooser**.

31.29.3.29  **user_data()**

```cpp
void * Fl_File_Chooser::user_data ( ) const
```

Gets the file chooser user data.
31.29.3.30 value()

```cpp
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:

```cpp
// Get list of filenames user selected from a MULTI chooser
for ( int t=1; t<chooser->count(); t++ ) {
    const char *filename = chooser->value(t);
    ...
}
```

31.29.3.31 visible()

```cpp
int Fl_File_Chooser::visible ( )
```

Returns 1 if the *Fl_File_Chooser* window is visible.

31.29.4 Member Data Documentation

31.29.4.1 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*

31.30 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.

```cpp
#include <Fl_File_Icon.H>
```
Public Types

- enum {
  ANY, PLAIN, FIFO, DEVICE,
  LINK, DIRECTORY
}
- enum {
  END, COLOR, LINE, CLOSEDLINE,
  POLYGON, OUTLINEPOLYGON, VERTEX
}

Public Member Functions

- short * add (short d)
  Adds a keyword value to the icon array, returning a pointer to it.
- short * add_color (Fl_Color c)
  Adds a color value to the icon array, returning a pointer to it.
- short * add_vertex (int x, int y)
  Adds a vertex 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.
- 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)
  
  Draws the icon label.

- static void load_system_icons (void)
  
  Loads all system-defined icons.

31.30.1 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.

31.30.2 Constructor & Destructor Documentation

31.30.2.1 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

| in  | p         | filename pattern |
|     | t         | file type        |
| in  | nd        | number of data values |
| in  | d         | data values      |

31.30.3 Member Function Documentation

31.30.3.1 add()

short * Fl_File_Icon::add (
  short d)

Adds a keyword value to the icon array, returning a pointer to it.
### 31.30.3.2 add_color()

```c
short* Fl_File_Icon::add_color ( Fl_Color c ) [inline]
```

Adds a color value to the icon array, returning a pointer to it.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>d</th>
<th>data value</th>
</tr>
</thead>
</table>

### 31.30.3.3 add_vertex() [1/2]

```c
short* Fl_File_Icon::add_vertex ( int x, int y ) [inline]
```

Adds a vertex value to the icon array, returning a pointer to it.

The integer version accepts coordinates from 0 to 10000. The origin (0.0) is in the lower-lefthand corner of the icon.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>vertex coordinates</th>
</tr>
</thead>
</table>

### 31.30.3.4 add_vertex() [2/2]

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>vertex coordinates</th>
</tr>
</thead>
</table>
31.30.3.5 clear()

void Fl_File_Icon::clear ( ) [inline]

Clears all icon data from the icon.

31.30.3.6 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>

31.30.3.7 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
31.30.3.8 first()

```cpp
static Fl_File_Icon* Fl_File::first() [inline], [static]
```

Returns a pointer to the first icon in the list.

31.30.3.9 label()

```cpp
void Fl_File::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 |

31.30.3.10 labeltype()

```cpp
void Fl_File::labeltype (const Fl_Label* o, int x, int y, int w, int h, Fl_Align a) [static]
```

Draw the icon label.

**Parameters**

| in | o | label data |
| in | x,y,w,h | position and size of label |
| in | a | label alignment [not used] |

31.30.3.11 load()

```cpp
void Fl_File::load (const char* f)
```

Loads the specified icon image.

The format is deduced from the filename.
31.30.3.12 load_fti()

```c
int Fl_File_Icon::load_fti (const char * fti)
```

Loads an SGI icon file.

Parameters

- in fti icon filename

Returns

0 on success, non-zero on error

31.30.3.13 load_image()

```c
int Fl_File_Icon::load_image (const char * ifile)
```

Load an image icon file from an image filename.

Parameters

- in ifile image filename

Returns

0 on success, non-zero on error

31.30.3.14 load_system_icons()

```c
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:

```c
Fl_File_Icon::load_system_icons();
```
31.30.3.15 next()

Fl_File_Icon* Fl_File_Icon::next() [inline]

Returns next file icon object.
See Fl_File_Icon::first()

31.30.3.16 pattern()

const char* Fl_File_Icon::pattern() [inline]

Returns the filename matching pattern for the icon.

31.30.3.17 size()

int Fl_File_Icon::size() [inline]

Returns the number of words of data used by the icon.

31.30.3.18 type()

int Fl_File_Icon::type() [inline]

Returns the filetype associated with the icon, which can be one of the following:

- Fl_File_Icon::ANY, any kind of file.
- Fl_File_Icon::PLAIN, plain files.
- Fl_File_Icon::FIFO, named pipes.
- Fl_File_Icon::DEVICE, character and block devices.
- Fl_File_Icon::LINK, symbolic links.
- Fl_File_Icon::DIRECTORY, directories.

31.30.3.19 value()

short* Fl_File_Icon::value() [inline]

Returns the data array for the icon.

The documentation for this class was generated from the following files:

- Fl_File_Icon.H
- Fl_File_Icon.cxx
- Fl_File_Icon2.cxx

Generated by Doxygen
31.31 Fl_File_Input Class Reference

This widget displays a pathname in a text input field.

```
#include <Fl_File_Input.H>
```

Inheritance diagram for Fl_File_Input:

```
Fl_Widget
    ↓
Fl_Input_
    ↓
Fl_Input
    ↓
Fl_File_Input
```

Public Member Functions

- **Fl_Boxtype down_box () const**
  
  Gets the box type used for the navigation bar.

- **void down_box (Fl_Boxtype b)**
  
  Sets the box type to use for the navigation bar.

- **Fl_Color errorcolor () const**
  
  Gets the current error color.

- **void errorcolor (Fl_Color c)**
  
  Sets the current error color to c.

- **Fl_File_Input (int X, int Y, int W, int H, const char *L=0)**
  
  Creates a new Fl_File_Input widget using the given position, size, and label string.

- **virtual int handle (int event)**
  
  Handle events in the widget.

- **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.

- **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.

Protected Member Functions

- **virtual void draw ()**
  
  Draws the file input widget.
Additional Inherited Members

31.31.1 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 31.15 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.

31.31.2 Constructor & Destructor Documentation

31.31.2.1 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

| in | X,Y,W,H | position and size of the widget |
| in | L       | widget label, default is no label |

31.31.3 Member Function Documentation

Generated by Doxygen
31.31.3.1  down_box() [1/2]

```cpp
Fl_Boxtype Fl_File_Input::down_box ( ) const [inline]
```

Gets the box type used for the navigation bar.

31.31.3.2  down_box() [2/2]

```cpp
void Fl_File_Input::down_box ( Fl_Boxtype b ) [inline]
```

Sets the box type to use for the navigation bar.

31.31.3.3  errorcolor() [1/2]

```cpp
Fl_Color Fl_File_Input::errorcolor ( ) const [inline]
```

Gets the current error color.

Returns \texttt{FL\_RED} since FLTK 1.4.0 (default in 1.3.x). Retained for backwards compatibility.

\textbf{Deprecated} Will be removed in FLTK 1.5.0 or higher.

\textbf{Todo} Remove \texttt{Fl\_File\_Input::errorcolor()} in FLTK 1.5.0 or higher.

31.31.3.4  errorcolor() [2/2]

```cpp
void Fl_File_Input::errorcolor ( Fl_Color c ) [inline]
```

Sets the current error color to \texttt{c}.

Does nothing since FLTK 1.4.0. Retained for backwards compatibility.

\textbf{Deprecated} Will be removed in FLTK 1.5.0 or higher.

\textbf{Todo} Remove \texttt{Fl\_File\_Input::errorcolor(Fl\_Color)} in FLTK 1.5.0 or higher.

31.31.3.5  handle()

```cpp
int Fl_File_Input::handle ( int event ) [virtual]
```

Handle events in the widget.

Return non zero if event is handled.
Parameters

Reimplemented from Fl_Widget.

31.31.3.6 value() [1/2]

int Fl_File_Input::value (const char * str)

Sets the value of the widget given a new string value.

Returns non 0 on success.

Parameters

31.31.3.7 value() [2/2]

int Fl_File_Input::value (const char * str, int len)

Sets the value of the widget given a new string value and its length.

Returns non 0 on success.

Parameters

The documentation for this class was generated from the following files:

- Fl_File_Input.H
- Fl_File_Input.cxx

31.32 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`.

Additional Inherited Members

31.32.1 Detailed Description

Draws a dial with a filled arc.

31.32.2 Constructor & Destructor Documentation

31.32.2.1 Fl_Fill_Dial()

```
Fl_Fill_Dial::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`.

The documentation for this class was generated from the following files:

- Fl_Fill_Dial.H
- Fl_Dial.cxx
31.33  Fl_Fill_Slider Class Reference

Widget that draws a filled horizontal slider, useful as a progress or value meter.

```
#include <Fl_Fill_Slider.H>
```

Inheritance diagram for Fl_Fill_Slider:

```
Fl_Widget
    Fl_Valuator
        Fl_Slider
            Fl_Fill_Slider
```

Public Member Functions

- **Fl_Fill_Slider**(int X, int Y, int W, int H, const char *L=0)
  
  Creates the slider from its position, size and optional title.

Additional Inherited Members

31.33.1  Detailed Description

Widget that draws a filled horizontal slider, useful as a progress or value meter.

31.33.2  Constructor & Destructor Documentation

31.33.2.1  Fl_Fill_Slider()

```
Fl_Fill_Slider::Fl_Fill_Slider (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * L = 0 )
```

Creates the slider from its position, size and optional title.

The documentation for this class was generated from the following files:

- Fl_Fill_Slider.H
- Fl_Slider.cxx
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).

```c
#include <Fl_Float_Input.H>
```

Inheritance diagram for **Fl_Float_Input**:

```
Fl_Widget
   `-- Fl_Float_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.

### Additional Inherited Members

#### 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).

#### Constructor & Destructor Documentation

##### 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**
### 31.35 Fl_FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.

```cpp
#include <Fl_FormsBitmap.H>
```

Inheritance diagram for Fl_FormsBitmap:

```
Fl_Widget
   Fl_FormsBitmap
```

#### Public Member Functions

- `void bitmap (Fl_Bitmap *B)`
  
  Sets a new bitmap.

- `Fl_Bitmap * bitmap () const`
  
  Gets a the current associated `Fl_Bitmap` objects.

- `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.

#### Protected Member Functions

- `void draw ()`
  
  Draws the bitmap and its associated box.

#### Additional Inherited Members

#### 31.35.1 Detailed Description

Forms compatibility Bitmap Image Widget.

#### 31.35.2 Member Function Documentation

#### 31.35.2.1 bitmap() [1/2]

```cpp
void Fl_FormsBitmap::bitmap ( Fl_Bitmap * B ) [inline]
```

Sets a new bitmap.
31.35.2.2 bitmap() [2/2]

```cpp
Fl_Bitmap* Fl_FormsBitmap::bitmap ( ) const [inline]
```

Gets a the current associated Fl_Bitmap objects.

31.35.2.3 draw()

```cpp
void Fl_FormsBitmap::draw ( void ) [protected], [virtual]
```

Draws the bitmap and its associated box.

Implements Fl_Widget.

31.35.2.4 set()

```cpp
void Fl_FormsBitmap::set ( int W, int H, const uchar * bits )
```

Sets a new bitmap bits with size W,H.

Deletes the previous one.

The documentation for this class was generated from the following files:

- Fl_FormsBitmap.H
- forms_bitmap.cxx

31.36 Fl_FormsPixmap Class Reference

Forms pixmap drawing routines.

```cpp
#include <Fl_FormsPixmap.H>
```

Inheritance diagram for Fl_FormsPixmap:

```
 Fl_Widget
  
 Fl_FormsPixmap
```

Generated by Doxygen
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.

- `void Pixmap (Fl_Pixmap *B)`
  Set the internal pixmap pointer to an existing pixmap.

- `Fl_Pixmap * Pixmap () const`
  Get the internal pixmap pointer.

- `void set (char *const *bits)`
  Set/create the internal pixmap using raw data.

Protected Member Functions

- `void draw ()`
  Draws the widget.

Additional Inherited Members

31.36.1 Detailed Description

Forms pixmap drawing routines.

31.36.2 Constructor & Destructor Documentation

31.36.2.1 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>
31.36.3 Member Function Documentation

31.36.3.1 draw()

```cpp
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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

31.36.3.2 Pixmap() [1/2]

```cpp
void Fl_FormsPixmap::Pixmap(Fl_Pixmap *B) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

**Parameters**

- `in B` existing pixmap

31.36.3.3 Pixmap() [2/2]

```cpp
Fl_Pixmap* Fl_FormsPixmap::Pixmap() const [inline]
```

Get the internal pixmap pointer.

31.36.3.4 set()

```cpp
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
- formsPixmap.cxx

31.37 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)**

Protected Member Functions

- void **draw()**
  
  *Draws the widget.*

Additional Inherited Members

31.37.1 Member Function Documentation

31.37.1.1 **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:

```
FI_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

*Implements Fl_Widget.*

The documentation for this class was generated from the following file:

- forms.H
31.38 Fl_Free Class Reference

Emulation of the Forms "free" widget.

#include <Fl_Free.H>

Inheritance diagram for Fl_Free:

```
Fl_Widget
   |
   +----------+----------+
   |          |          |
Fl_Free    |
```

Public Member Functions

- **Fl_Free** (uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl)
  
  Create a new Fl_Free widget with type, position, size, label and handler.

- **int handle** (int e)
  
  Handles the specified event.

- **Fl_Free** ()
  
  The destructor will call the handle function with the event FL_FREE_MEM.

Protected Member Functions

- **void draw** ()
  
  Draws the widget.

Additional Inherited Members

31.38.1 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.
31.38.2 Constructor & Destructor Documentation

31.38.2.1 Fl_Free()

Fl_Free::Fl_Free (uchar t, int X, int Y, int W, int H, const char * L, FL_HANDLEPTR hdl)

Create a new Fl_Free widget with type, position, size, label and handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>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</td>
</tr>
<tr>
<td>in</td>
<td>hdl</td>
<td>handler function</td>
</tr>
</tbody>
</table>

The constructor takes both the type and the handle function. The handle function should be declared as follows:

```
int handle_function(Fl_Widget *w, int event, float event_x, float event_y, char key)
```

This function is called from the handle() method in response to most events, and is called by the draw() method.

The event argument contains the event type:

```
// 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
```
31.38.3.1 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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

31.38.3.2 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

Generated by Doxygen
The `Fl_GIF_Image` class supports loading, caching, and drawing of Compuserve GIF\textsuperscript{SM} images.

```cpp
#include <Fl_GIF_Image.H>
```

Inheritance diagram for `Fl_GIF_Image`:

```
Fl_Image
   Fl_Pixmap
   Fl_GIF_Image
```

### Public Member Functions

- `Fl_GIF_Image` (const char ∗filename)
  
  The constructor loads the named GIF image.

- `Fl_GIF_Image` (const char ∗imagename, const unsigned char ∗data)
  
  The constructor loads a GIF image from memory.

### Protected Member Functions

- void `load_gif_` (class `Fl_Image_Reader` &rdr)

### Additional Inherited Members

31.39.1 Detailed Description

The `Fl_GIF_Image` class supports loading, caching, and drawing of Compuserve GIF\textsuperscript{SM} images.

The class loads the first image and supports transparency.

31.39.2 Constructor & Destructor Documentation

31.39.2.1 `Fl_GIF_Image()` [1/2]

```cpp
Fl_GIF_Image::Fl_GIF_Image (const char ∗filename )
```

The constructor loads the named GIF image.

If a GIF is animated, `Fl_GIF_Image` will only read and display the first frame of the animation.

The destructor frees all memory and server resources that are used by the image.

Use `Fl_Image::fail()` to check if `Fl_GIF_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.
31.39.2.2 Fl_GIF_Image()

Fl_GIF_Image::Fl_GIF_Image (  
    const char * imagename,  
    const unsigned char * data )

The constructor loads a GIF image from memory.

Construct an image from a block of memory inside the application. Fluid offers “binary Data” chunks as a great way to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

If a GIF is animated, Fl_GIF_Image will only read and display the first frame of the animation.

Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>imagename</th>
<th>A name given to this image or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>Pointer to the start of the GIF image in memory. This code will not check for buffer overruns.</td>
</tr>
</tbody>
</table>

See also

Fl_GIF_Image::Fl_GIF_Image(const char *filename)

Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_GIF_Image.H
- Fl_GIF_Image.cxx

31.40 Fl_Gl_Choice Class Reference
31.41 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
      |
      Fl_Glut_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.

- GLContext context () const
  
  Returns a pointer to the GLContext that this window is using.

- void context (GLContext, int destroy_flag=0)
  
  Sets a pointer to the GLContext that this window is using.

- char context_valid () const
  
  Will only be set if the OpenGL context is created or recreated.

- void context_valid (char v)
  
  See char Fl_Gl_Window::context_valid() const.

- Fl_Gl_Window (int W, int H, const char *l=0)

  Creates a new Fl_Gl_Window widget using the given size, and label string.

- Fl_Gl_Window (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Gl_Window widget using the given position, size, and label string.
• void flush ()
  
  Forces the window to be drawn, this window is also made current and calls draw().

• Fl_Gl_Window_Driver * gl_driver ()
  
  Returns a pointer to the window's Fl_Gl_Window_Driver object.

• 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 Windows.

• void invalidate ()
  
  The invalidate() method turns off valid() and is equivalent to calling value(0).

• void make_current ()
  
  The make_current() method selects the OpenGL context for the widget.

• void make_overlay_current ()
  
  Selects the OpenGL context for the widget's overlay.

• Fl_Mode mode () const
  
  Returns the current OpenGL capabilities of the window.

• int mode (int a)
  
  Set or change the OpenGL capabilities of the window.

• int mode (const int * a)
  
  Set the OpenGL capabilities of the window using platform-specific data.

• void ortho ()
  
  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.

• int pixel_h ()
  
  Gives the window height in OpenGL pixels.

• int pixel_w ()
  
  Gives the window width in OpenGL pixels.

• float pixels_per_unit ()
  
  The number of pixels per FLTK unit of length for the window.

• void redraw_overlay ()
  
  Causes draw_overlay() to be called at a later time.

• void resize (int, int, int, int)
  
  Changes the size and position of the window.

• void show ()
  
  Puts the window on the screen.

• void show (int a, char ** b)
  
  Same as Fl_Window::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.
Static Public Member Functions

- static int can_do (int m)
  
  Returns non-zero if the hardware supports the given OpenGL mode.

- static int can_do (const int *m)
  
  Returns non-zero if the hardware supports the given OpenGL mode.

Protected Member Functions

- virtual void draw ()
  
  Draws the Fl_Gl_Window.

Friends

- class Fl_Gl_Window_Driver

Additional Inherited Members

31.41.1 Detailed Description

The Fl_Gl_Window widget sets things up so OpenGL works.

It also keeps an OpenGL "context" for that window, so that changes to the lighting and projection may be reused between redraws. Fl_Gl_Window also flushes the OpenGL streams and swaps buffers after draw() returns.

OpenGL hardware typically provides some overlay bit planes, which are very useful for drawing UI controls atop your 3D graphics. If the overlay hardware is not provided, FLTK tries to simulate the overlay. This works pretty well if your graphics are double buffered, but not very well for single-buffered.

Please note that the FLTK drawing and clipping functions will not work inside an Fl_Gl_Window. All drawing should be done using OpenGL calls exclusively.

See also

OpenGL and support of HighDPI displays

Note

FLTK 1.4 introduces a driver system for graphic calls. It is now possible to add a selection of widgets to an OpenGL window. The widgets will draw on top of any OpenGL rendering. The number of supported widgets will increase as the driver development improves. Program test/cube.cxx illustrates how to do that.
31.41.2.1 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.

31.41.2.2 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.

31.41.3 Member Function Documentation

31.41.3.1 as_gl_window()

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.
31.41.3.2 can_do() [1/3]

static int Fl_Gl_Window::can_do (  
    int m ) [inline], [static]

Returns non-zero if the hardware supports the given OpenGL mode.

31.41.3.3 can_do() [2/3]

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)

31.41.3.4 can_do() [3/3]

int Fl_Gl_Window::can_do ( ) [inline]

Returns non-zero if the hardware supports the current OpenGL mode.

31.41.3.5 can_do_overlay()

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.

31.41.3.6 context() [1/2]

GLContext Fl_Gl_Window::context ( ) const [inline]

Returns a pointer to the GLContext that this window is using.

See also

   void context(GLContext c, int destroy_flag)
31.41.3.7 context() [2/2]

```cpp
void Fl_Gl_Window::context (  
    GLContext v,  
    int destroy_flag = 0 )
```

Sets a pointer to the GLContext that this window is using.

This is a system-dependent structure, but it is portable to copy the context from one window to another. You can also set it to NULL, which will force FLTK to recreate the context the next time make_current() is called, this is useful for getting around bugs in OpenGL implementations.

If `destroy_flag` is true the context will be destroyed by fltk when the window is destroyed, or when the mode() is changed, or the next time context(x) is called.

31.41.3.8 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.

31.41.3.9 draw()

```cpp
void Fl_Gl_Window::draw (  
    void ) [protected], [virtual]
```

Draws the `Fl_Gl_Window`.

You must subclass `Fl_Gl_Window` and provide an implementation for `draw()`. You may also provide an implementation of `draw_overlay()` if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking `valid()` at the start of `draw()` and only doing the initialization if it is false.

The `draw()` method can only use OpenGL calls. Do not attempt to call X, any of the functions in `<FL/fl_draw.H>`, or glX directly. Do not call `gl_start()` or `gl_finish()`.

If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.

The following pseudo-code shows how to use "if (!valid())" to initialize the viewport:

```cpp
void mywindow::draw() {  
    if (!valid()) {  
        glViewport(0,0,pixel_w(),pixel_h());  
        glFrustum(...) or glOrtho(...)  
        ...other initialization...  
    }  
    if (!context_valid()) {  
        ...load textures, etc. ...  
    }  
    // clear screen  
    glClearColor(...);  
    glClear(...);  
    ... draw your geometry here ...  
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```cpp
```
void mywindow::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0, 0, pixel_w(), pixel_h());
        glOrtho(-w(), w(), -h(), h(), -1, 1);
    }
    // Clear screen
    glClear(GL_COLOR_BUFFER_BIT);
    // Draw white 'X'
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
    glBegin(GL_LINE_STRIP); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}

Implements Fl_Widget.

Reimplemented in Fl_Glut_Window.

31.41.3.10 flush()

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.

31.41.3.11 hide_overlay()

void Fl_Gl_Window::hide_overlay ( )

Hides the window if it is not this window, does nothing in Windows.

31.41.3.12 make_current()

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.

31.41.3.13 make_overlay_current()

void Fl_Gl_Window::make_overlay_current ( )

Selects the OpenGL context for the widget's overlay.

This method is called automatically prior to the draw_overlay() method being called and can also be used to implement feedback and/or selection within the handle() method.
31.41.3.14  mode() [1/3]

Fl_Mode Fl_Gl_Window::mode ( ) const [inline]

Returns the current OpenGL capabilities of the window.

Don’t use this if capabilities were set through Fl_Gl_Window::mode(const int *a).

31.41.3.15  mode() [2/3]

int Fl_Gl_Window::mode ( 
    int a ) [inline]

Set or change the OpenGL capabilities of the window.

The value can be any of the following OR'd together:

- FL_RGB - RGB color (not indexed)
- FL_RGB8 - RGB color with at least 8 bits of each color
- FL_INDEX - Indexed mode
- FL_SINGLE - not double buffered
- FL_DOUBLE - double buffered
- FL_ACCUM - accumulation buffer
- FL_ALPHA - alpha channel in color
- FL_DEPTH - depth buffer
- FL_STENCIL - stencil buffer
- FL_MULTISAMPLE - multisample antialiasing
- FL_OPENGL3 - use OpenGL version 3.0 or more.

FL_RGB and FL_SINGLE have a value of zero, so they are "on" unless you give FL_INDEX or FL_DOUBLE.

If the desired combination cannot be done, FLTK will try turning off FL_MULTISAMPLE. If this also fails the show() will call Fl::error() and not show the window.

You can change the mode while the window is displayed. This is most useful for turning double-buffering on and off. Under X this will cause the old X window to be destroyed and a new one to be created. If this is a top-level window this will unfortunately also cause the window to blink, raise to the top, and be de-iconized, and the xid() will change, possibly breaking other code. It is best to make the GL window a child of another window if you wish to do this!

mode() must not be called within draw() since it changes the current context.

The FL_OPENGL3 flag is required to access OpenGL version 3 or more under the X11 and MacOS platforms; it's optional under Windows. See more details in Using OpenGL 3.0 (or higher versions).

Version

the FL_OPENGL3 flag appeared in version 1.3.4

31.41.3.16  mode() [3/3]

int Fl_Gl_Window::mode ( 
    const int * a ) [inline]

Set the OpenGL capabilities of the window using platform-specific data.
Parameters

| a | zero-ending array of platform-specific attributes and attribute values |

**Unix/Linux platform:** attributes are GLX attributes adequate for the 3rd argument of the glXChooseVisual() function (e.g., GLX_DOUBLEBUFFER, defined by including `<GL/glx.h>`).

**Note**

What attributes are adequate here is subject to change. The preferred, stable public API is Fl_Gl_Window::mode(int a).

**Windows platform:** this member function is of no use.

**Mac OS X platform:** attributes belong to the CGLPixelFormatAttribute enumeration (defined by including `<OpenGL/OpenGL.h>`, e.g., kCGLPFADoubleBuffer) and may be followed by adequate attribute values.

### 31.41.3.17 ortho()

```cpp
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.

### 31.41.3.18 pixel_h()

```cpp
int Fl_Gl_Window::pixel_h ( ) [inline]
```

Gives the window height in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::h() which is expressed in FLTK units, may differ from the window height in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use `pixel_h()` rather than `h()`. Method `pixel_h()` detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

**Version**

1.3.4

### 31.41.3.19 pixel_w()

```cpp
int Fl_Gl_Window::pixel_w ( ) [inline]
```

Gives the window width in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::w() which is expressed in FLTK units, may differ from the window width in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use `pixel_w()` rather than `w()`. Method `pixel_w()` detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

**Version**

1.3.4
31.41.3.20 pixels_per_unit()

float Fl_Gl_Window::pixels_per_unit ( )

The number of pixels per FLTK unit of length for the window.

This method dynamically adjusts its value when the GUI is rescaled or when the window is moved to/from displays of distinct resolutions. This method is useful, e.g., to convert, in a window's handle() method, the FLTK units returned by Fl::event_x() and Fl::event_y() to the pixel units used by the OpenGL source code.

Version

1.3.4

31.41.3.21 redraw_overlay()

void Fl_Gl_Window::redraw_overlay ( )

Causes draw_overlay() to be called at a later time.

Initially the overlay is clear. If you want the window to display something in the overlay when it first appears, you must call this immediately after you show() your window.

31.41.3.22 resize()

void Fl_Gl_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_Window.
31.41.3.23 show()

```cpp
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`.

31.41.3.24 swap_buffers()

```cpp
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.

31.41.3.25 valid()

```cpp
char Fl_Gl_Window::valid ( ) const [inline]
```

Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after `draw()` is called.

You can use this inside your `draw()` method to avoid unnecessarily initializing the OpenGL context. Just do this:

```cpp
void mywindow::draw() {
  if (!valid()) {
    glViewport(0,0,pixel_w(),pixel_h());
    glFrustum(...);
    ...other initialization...
  }
  if (!context_valid()) {
    ...load textures, etc. ...
  }
  ... draw your geometry here ...
}
```

You can turn `valid()` on by calling `valid(1)`. You should only do this after fixing the transformation inside a `draw()` or after `make_current()`. This is done automatically after `draw()` returns.

The documentation for this class was generated from the following files:

- Fl_Gl_Window.H
- Fl_Gl_Overlay.cxx
- Fl_Gl_Window.cxx

Generated by Doxygen
31.42 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

31.42.1 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

31.43 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

31.44 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
31.45 **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

31.46 **Fl_Glut_StrokeVertex Struct Reference**

Public Attributes

- GLfloat **X**
- GLfloat **Y**

The documentation for this struct was generated from the following file:

- glut.H

31.47 **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
```

Generated by Doxygen
Public Member Functions

- **Fl_Glut_Window (int w, int h, const char *t=0)**
  
  Creates a glut window, registers to the glut windows list.

- **Fl_Glut_Window (int x, int y, int w, int h, const char *t=0)**
  
  Creates a glut window, registers to the glut windows list.

- **void make_current ()**
- **~Fl_Glut_Window ()**
  
  Destroys the glut window, first unregister it from the glut windows list.

Public 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.

Additional Inherited Members

31.47.1 Detailed Description

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

31.47.2 Constructor & Destructor Documentation
31.47.2.1 Fl_Glut_Window() [1/2]

Fl_Glut_Window::Fl_Glut_Window (  
   int \_W,  
   int \_H,  
   const char * \_t = 0 )

Creates a glut window, registers to the glut windows list.

31.47.2.2 Fl_Glut_Window() [2/2]

Fl_Glut_Window::Fl_Glut_Window (  
   int \_X,  
   int \_Y,  
   int \_W,  
   int \_H,  
   const char * \_t = 0 )

Creates a glut window, registers to the glut windows list.

31.47.3 Member Function Documentation

31.47.3.1 draw()

void Fl_Glut_Window::draw (  
   void ) [protected], [virtual]

Draws the Fl_Gl_Window.

You must subclass Fl_Gl_Window and provide an implementation for draw(). You may also provide an implementation of draw_overlay() if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking valid() at the start of draw() and only doing the initialization if it is false.

The draw() method can only use OpenGL calls. Do not attempt to call X, any of the functions in <FL/fl_draw.H>, or glX directly. Do not call gl_start() or gl_finish().

If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.

The following pseudo-code shows how to use “if (!valid())” to initialize the viewport:

void mywindow::draw() {  
   if (!valid()) {  
      glViewport(0,0,pixel_w(),pixel_h());  
      glFrustum(...) or glOrtho(...)  
      ...other initialization...  
   }  
   if (!context_valid()) {  
      ...load textures, etc. ...  
   }  
   // clear screen  
   glColorColor(...);  
   glClear(...);  
   ... draw your geometry here ...  
}
Actual example code to clear screen to black and draw a 2D white "X":

```cpp
void mywindow::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0, 0, pixel_w(), pixel_h());
        glOrtho(-w(), w(), -h(), h(), -1, 1);
    }
    // Clear screen
    glClear(GL_COLOR_BUFFER_BIT);
    // Draw white 'X'
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
    glBegin(GL_LINE_STRIP); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

Reimplemented from Fl_Gl_Window.

31.47.3.2 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.

The documentation for this class was generated from the following files:

- glut.H
- glut_compatibility.cxx

31.48 Fl_Group Class Reference

The Fl_Group class is the FLTK container widget.

```cpp
#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.
- void clip_children (int c)
Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `unsigned int clip_children ()`
  Returns the current clipping mode.
- `void end ()`
  Exactly the same as `current(this->parent())`.
- `int find (const Fl_Widget *) const`
  Searches the child array for the widget and returns the index.
- `int find (const Fl_Widget &o) const`
  See `int Fl_Group::find(const Fl_Widget *w) const`.
- `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 &o, Fl_Widget *before)`
  This does `insert(w, find(before))`.
- `void remove (int index)`
  Removes the widget at index from the group but does not delete it.
- `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 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.
- `Fl_Widget *resizable () const`
  See `void Fl_Group::resizable(Fl_Widget *box)`
- `void resize (int, int, int)`
  Resizes the `Fl_Group` widget and all of its children.
- `virtual ~Fl_Group ()`
  The destructor also deletes all the children.

Static Public Member Functions

- `static Fl_Group *current ()`
  Returns the currently active group.
- `static void current (Fl_Group *g)`
  Sets the current group.
Protected Member Functions

- Fl_Rect * bounds ()
  Returns the internal array of widget sizes and positions.
- 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.

Additional Inherited Members

31.48.1 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.

31.48.2 Constructor & Destructor Documentation

31.48.2.1 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.
31.48.2.2 \texttt{\textasciitilde Fl\_Group()}

\texttt{Fl\_Group::\textasciitilde 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 \texttt{Fl\_Group} and all of its children are automatic (local) variables, but you must declare the \texttt{Fl\_Group} first, so that it is destroyed last.

If you add static or automatic (local) variables to an \texttt{Fl\_Group}, then it is your responsibility to remove (or delete) all such static or automatic child widgets \textit{before destroying} the group - otherwise the child widgets' destructors would be called twice!

31.48.3 Member Function Documentation

31.48.3.1 \texttt{array()}

\texttt{Fl\_Widget *const * Fl\_Group::array ( ) const}

Returns a pointer to the array of children.

\textbf{Note}

This pointer is only valid until the next time a child is added or removed.

31.48.3.2 \texttt{as\_group()}

\texttt{virtual Fl\_Group* Fl\_Group::as\_group ( ) [inline], [virtual]}

Returns an \texttt{Fl\_Group} pointer if this widget is an \texttt{Fl\_Group}.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from \texttt{Fl\_Group}. If it returns non-NULL, then the widget in question is derived from \texttt{Fl\_Group}, and you can use the returned pointer to access its children or other \texttt{Fl\_Group}-specific methods.

\textbf{Example:}

\begin{verbatim}
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");
}
\end{verbatim}
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.

31.48.3.3 begin()

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!

31.48.3.4 bounds()

Fl_Rect * Fl_Group::bounds ( ) [protected]

Returns the internal array of widget sizes and positions.

If the bounds() array does not exist, it will be allocated and filled with the current widget sizes and positions.

The bounds() array stores the initial positions of widgets as Fl_Rect's. The size of the array is children() + 2.

- The first Fl_Rect is the group,
- the second is the resizable (clipped to the group),
- the rest are the children.

This is a convenient order for the resize algorithm.

If the group and/or the resizable() is a Fl_Window (or subclass) then the x() and y() coordinates of their respective Fl_Rect's are zero.
Note

You should never need to use this protected method directly, unless you have special needs to rearrange the children of a Fl_Group. Fl_Tile uses this to rearrange its widget positions. The returned array should be considered read-only. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call init_sizes().

#include <FL/Fl_Rect.H> if you want to access the bounds() array in your derived class. Fl_Rect.H is intentionally not included by Fl_Group.H to avoid unnecessary dependencies.

Returns

Array of Fl_Rect's with widget positions and sizes. The returned array is only valid until init_sizes() is called or widgets are added to or removed from the group.

See also

init_sizes()

Since

FLTK 1.4.0

31.48.3.5 child()

Fl_Widget* Fl_Group::child ( int n ) const [inline]

Returns array[n].

No range checking is done!

31.48.3.6 clear()

def 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.

The resizable() widget of the Fl_Group is set to the Fl_Group itself.

31.48.3.7 clip_children() [1/2]

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.
31.48.3.8 clip_children() [2/2]

`unsigned int Fl_Group::clip_children ( ) [inline]`

Returns the current clipping mode.

Returns

true, if clipping is enabled, false otherwise.

See also

void Fl_Group::clip_children(int c)

31.48.3.9 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)`.

31.48.3.10 current() [2/2]

`void Fl_Group::current ( Fl_Group * g ) [static]`

Sets the current group.

See also

`Fl_Group::current()`

31.48.3.11 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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.


Generated by Doxygen
31.48.3.12 draw_child()

```cpp
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.

31.48.3.13 draw_children()

```cpp
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.

31.48.3.14 draw_outside_label()

```cpp
void Fl_Group::draw_outside_label (  
    const Fl_Widget & widget ) const [protected]
```

Parents normally call this to draw outside labels of child widgets.

31.48.3.15 end()

```cpp
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.

31.48.3.16 find()

```cpp
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.
31.48.3.17 focus()

```c
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();`

31.48.3.18 handle()

```c
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**

- **in event** the kind of event received

**Return values**

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

See also

`Fl_Event`

Reimplemented from `Fl_Widget`.

Reimplemented in `Fl_Tree`, `Fl_Help_View`, `Fl_Table`, `Fl_Tabs`, `Fl_Window`, `Fl_Text_Display`, `Fl_Scroll`, `Fl_Table_Row`, `Fl_Spinner`, `Fl_Text_Editor`, `Fl_Glut_Window`, and `Fl_Tile`.  

Generated by Doxygen
31.48.3.19 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 resize() occurs. For more information on the contents and structure of the bounds() array see bounds().

See also

bounds()
sizes() (deprecated)

31.48.3.20 insert() [1/2]

void Fl_Group::insert ( Fl_Widget & o, int index )

The widget is removed from its current group (if any) and then inserted into this group. It is put at index n - or at the end, if n >= children(). This can also be used to rearrange the widgets inside a group.

31.48.3.21 insert() [2/2]

void Fl_Group::insert ( Fl_Widget & o, Fl_Widget * before ) [inline]

This does insert(w, find(before)).

This will append the widget if before is not in the group.

31.48.3.22 remove() [1/3]

void Fl_Group::remove ( int index )

Removes the widget at index from the group but does not delete it.

This method does nothing if index is out of bounds.

This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

Since

FLTK 1.3.0
31.48.3.23 remove() [2/3]

```cpp
void Fl_Group::remove (Fl_Widget & o )
```

Removes a widget from the group but does not delete it.

This method does nothing if the widget is not a child of the group.

This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

**Note**

If you have the child's index anyway, use remove(int index) instead, because this doesn't need a child lookup in the group's table of children. This can be much faster, if there are lots of children.

31.48.3.24 remove() [3/3]

```cpp
void Fl_Group::remove (Fl_Widget * o ) [inline]
```

Removes the widget \( o \) from the group.

**See also**

void remove(Fl_Widget&)

31.48.3.25 resizable()

```cpp
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:
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.

31.48.3.26 resize()

```c
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*)
See also

```cpp
Fl_Group::resizable(Fl_Widget*)
Fl_Group::resizable()
Fl_Widget::resize(int,int,int,int)
```

Reimplemented from `Fl_Widget`.

Reimplemented in `Fl_Table`, `Fl_Text_Display`, `Fl_Tree`, `Fl_Help_View`, `Fl_Window`, `Fl_Scroll`, `Fl_Input_Choice`, `Fl_Spinner`, `Fl_Overlay_Window`, and `Fl_Tile`.

### 31.48.3.27 sizes()

```cpp
int * Fl_Group::sizes () [protected]
```

Returns the internal array of widget sizes and positions.

For backward compatibility with FLTK versions before 1.4.

The `sizes()` array stores the initial positions of widgets as (left, right, top, bottom) quads. The first quad is the group, the second is the resizable (clipped to the group), and the rest are the children. If the group and/or the `resizable()` is a `Fl_Window`, then the first (left) and third (top) entries of their respective quads (x,y) are zero.

**Deprecated** Deprecated since 1.4.0. Please use `bounds()` instead.

**Note**

This method will be removed in a future FLTK version (1.5.0 or higher).

Returns

Array of int's with widget positions and sizes. The returned array is only valid until `init_sizes()` is called or widgets are added to or removed from the group.

**Note**

Since FLTK 1.4.0 the returned array is a **read-only** and re-ordered copy of the internal `bounds()` array. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call `init_sizes()`.

See also

```cpp
bounds()
```
31.48.3.28  update_child()

```cpp
void Fl_Group::update_child ( 
    Fl_Widget & widget ) const  [protected]
```

Draws a child only if it needs it.

This draws a child widget, if it is not clipped and if any damage() bits are set. The damage bits are cleared after drawing.

See also

```cpp
Fl_Group::draw_child(Fl_Widget& widget) const
```

The documentation for this class was generated from the following files:

- Fl_Group.H
- Fl_Group.cxx
- forms_compatibility.cxx

### 31.49  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

### 31.50  Fl_Help_Dialog Class Reference

The `Fl_Help_Dialog` widget displays a standard help dialog window using the `Fl_Help_View` widget.
Public Member Functions

- Fl_Help_Dialog ()
  
  The constructor creates the dialog pictured above.

- int h ()

  Returns the position and size of the help dialog.

- void hide ()

  Hides the Fl_Help_Dialog window.

- int load (const char * f)

  Loads the specified HTML file into the Fl_Help_View widget.

- void position (int xx, int yy)

  Set the screen position of the dialog.

- void resize (int xx, int yy, int ww, int hh)

  Change the position and size of the dialog.

- void show ()

  Shows the Fl_Help_Dialog window.

- void show (int argc, char ** argv)

  Shows the main Help Dialog Window Delegates call to encapsulated window_ void Fl_Window::show(int argc, char ** argv) instance method.

- void textsize (Fl_Fontsize s)

  Sets or gets the default text size for the help view.

- Fl_Fontsize textsize ()

  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.

- void value (const char * f)

  The first form sets the current buffer to the string provided and reformats the text.

- const char * value () const

  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.

31.50.1 Detailed Description

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.

The Fl_Help_Dialog class is not part of the FLTK core library, but instead of fltk_images. Use -use-images when compiling with fltk-config.
31.50.2 Constructor & Destructor Documentation

31.50.2.1 Fl_Help_Dialog()

Fl_Help_Dialog::Fl_Help_Dialog ( )

The constructor creates the dialog pictured above.

31.50.3 Member Function Documentation

31.50.3.1 h()

int Fl_Help_Dialog::h ( )

Returns the position and size of the help dialog.

31.50.3.2 hide()

void Fl_Help_Dialog::hide ( )

Hides the Fl_Help_Dialog window.

31.50.3.3 load()

int Fl_Help_Dialog::load ( const char * f )

Loads the specified HTML file into the Fl_Help_View widget.

The filename can also contain a target name ("filename.html#target"). Always use forward slashes as path delimiters, MSWindows-style backslashes are not supported here.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td></td>
<td>the name and path of an HTML file</td>
</tr>
</tbody>
</table>

Returns

0 on success, -1 on error

See also

Fl_Help_View::load(), fl_load_uri()

31.50.3.4 position()

```cpp
void Fl_Help_Dialog::position (int x, int y)
```

Set the screen position of the dialog.

31.50.3.5 resize()

```cpp
void Fl_Help_Dialog::resize (int xx, int yy, int ww, int hh)
```

Change the position and size of the dialog.

31.50.3.6 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.

31.50.3.7 textsize() [1/2]

```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

Generated by Doxygen
31.50.3.8 `textsize()` [2/2]

uchar Fl_Help_Dialog::textsize()

Sets or gets the default text size for the help view.

31.50.3.9 `value()` [1/2]

void Fl_Help_Dialog::value ( const char * v )

The first form sets the current buffer to the string provided and reformats the text. It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

31.50.3.10 `value()` [2/2]

const char * Fl_Help_Dialog::value ( ) const

The first form sets the current buffer to the string provided and reformats the text. It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

31.50.3.11 `visible()`

int Fl_Help_Dialog::visible ( )

Returns 1 if the Fl_Help_Dialog window is visible.

31.50.3.12 `w()`

int Fl_Help_Dialog::w ( )

Returns the position and size of the help dialog.

31.50.3.13 `x()`

int Fl_Help_Dialog::x ( )

Returns the position and size of the help dialog.
31.50.3.14 y()

int Fl_Help_Dialog::y ( )

Returns the position and size of the help dialog.

The documentation for this class was generated from the following files:

- Fl_Help_Dialog.H
- Fl_Help_Dialog.cxx
- Fl_Help_Dialog_Dox.cxx

### 31.51 Fl_Help_Font_Stack Struct Reference

#### Public Member Functions

- `size_t count () const`
  
  Gets the current count of font style elements in the stack.

- `Fl_Help_Font_Stack ()`
  
  Font stack construction, initialize attributes.

- `void init (Fl_Font f, Fl_Fontsize s, Fl_Color c)`

- `void pop (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)`
  
  Pops from the stack the font style triplet and calls `fl_font()` & `fl_color()` adequately.

- `void push (Fl_Font f, Fl_Fontsize s, Fl_Color c)`

  Pushes the font style triplet on the stack, also calls `fl_font()` & `fl_color()` adequately.

- `void top (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)`
  
  Gets the top (current) element on the stack.

#### Protected Attributes

- `Fl_Help_Font_Style elts_ [MAX_FL_HELP_FS_ELTS]`
  
  Font elements

- `size_t nfonts_`
  
  Current number of fonts in stack

### 31.51.1 Constructor & Destructor Documentation

#### 31.51.1.1 Fl_Help_Font_Stack()

Fl_Help_Font_Stack::Fl_Help_Font_Stack ( ) [inline]

Font stack construction, initialize attributes.
31.51.2 Member Function Documentation

31.51.2.1 count()

size_t Fl_Help_Font_Stack::count ( ) const [inline]

Gets the current count of font style elements in the stack.

31.51.2.2 top()

void Fl_Help_Font_Stack::top ( Fl_Font & f, Fl_Fontsize & s, Fl_Color & c ) [inline]

Gets the top (current) element on the stack.

The documentation for this struct was generated from the following file:

• Fl_Help_View.H

31.52 Fl_Help_Font_Style Struct Reference

Fl_Help_View font stack element definition.

#include <Fl_Help_View.H>

Public Member Functions

• Fl_Help_Font_Style ( Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor)
• void get ( Fl_Font &afont, Fl_Fontsize &asize, Fl_Color &acolor)
  Gets current font attributes.
• void set ( Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor)
  Sets current font attributes.

Public Attributes

• Fl_Color c
  Font Color.
• Fl_Font f
  Font.
• Fl_Fontsize s
  Font Size.
31.52.1 Detailed Description

*Fl_Help_View* font stack element definition.

The documentation for this struct was generated from the following file:

- *Fl_Help_View.H*

### 31.53 Fl_Help_Link Struct Reference

Definition of a link for the html viewer.

```c
#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.

### 31.53.1 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*

### 31.54 Fl_Help_Target Struct Reference

*Fl_Help_Target* structure.

```c
#include <Fl_Help_View.H>
```
Public Attributes

- char name [32]
  Target name.
- int y
  Y offset of target.

31.54.1 Detailed Description

Fl_Help_Target structure.

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

31.55 Fl_Help_View Class Reference

The Fl_Help_View widget displays HTML text.

#include <Fl_Help_View.H>

Inheritance diagram for Fl_Help_View:

```
Fl_Widget
   ↓
Fl_Group
   ↓
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.
- void leftline (int)
  Scrolls the text to the indicated position, given a pixel column.
- int leftline () const
Gets the left position in pixels. 

• 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) 

• void textcolor (Fl_Color c) 
  Sets the default text color.

• Fl_Color textcolor () const 
  Returns the current default text color.

• void textfont (Fl_Font f) 
  Sets the default text font.

• Fl_Font textfont () const 
  Returns the current default text font.

• void textsize (Fl_Fontsize s) 
  Sets the default text size.

• Fl_Fontsize textsize () const 
  Gets the default text size.

• const char ∗ title () 
  Returns the current document title, or NULL if there is no title.

• 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.

• int topline () const 
  Returns the current top line in pixels.

• void value (const char ∗val) 
  Sets the current help text buffer to the string provided and reformats the text.

• const char ∗ value () const 
  Returns the current buffer contents.

• ~Fl_Help_View () 
  Destroys the Fl_Help_View widget.

Protected Member Functions

• void draw () 
  Draws the Fl_Help_View widget.
31.55.1 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
- TITLE
- TT
- U
- UL
- VAR
Supported color names:

- black, red, green, yellow, blue, magenta, fuchsia, cyan, aqua, white, gray, grey, lime, maroon, navy, olive, purple, silver, teal.

Supported urls:

- Internal: file:
- External: http: ftp: https: ipp: mailto: news:

Quoted char names:

- Aacute aacute Acirc acirc acute AElig aelig Agrave agrave amp Aring aring Atilde atilde Auml auml
- brvbar bull
- Ccedil ccedil cedil cent copy curren
- dagger deg divide
- Eacute eacute Ecirc ecirc Egrave egrave ETH eth Euml euml euro
- frac12 frac14 frac34
- gt
- Iacute iacute Icirc icirc iexcl Igrave igrave iquest Iuml iuml
- laquo lt
- macr micro middot
- nbsp not Ntilde ntilde
- Oacute oacute Ocirc ocirc Ograve ograve ordf ordm Oslash oslash Otilde otilde Ouml ouml
- para permil plusmn pound
- quot
- raquo reg
- sect shy sup1 sup2 sup3 szlig
- THORN thorn times trade
- Uacute uacute Ucirc ucirc Ugrave ugrave uml Uuml uuml
- Yacute yacute
- yen Yuml yuml

31.55.2 Constructor & Destructor Documentation
31.55.2.1 \texttt{\~\textasciitilde Fl Help View()}

\texttt{Fl Help View::\~\textasciitilde Fl Help View ( )}

Destroys the \texttt{Fl Help View} widget.

The destructor destroys the widget and frees all memory that has been allocated for the current document.

31.55.3 Member Function Documentation

31.55.3.1 \texttt{clear_selection()}

\texttt{void Fl Help View::clear_selection ( )}

Removes the current text selection.

31.55.3.2 \texttt{directory()}

\texttt{const char \textasciitilde Fl Help View::directory ( ) const [inline]}

Returns the current directory for the text in the buffer.

31.55.3.3 \texttt{draw()}

\texttt{void Fl Help View::draw ( void ) [protected], [virtual]}

Draws the \texttt{Fl Help View} widget.

Reimplemented from \texttt{Fl Group}.

31.55.3.4 \texttt{filename()}

\texttt{const char \textasciitilde Fl Help View::filename ( ) const [inline]}

Returns the current filename for the text in the buffer.
31.55.3.5  find()

```cpp
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

31.55.3.6  handle()

```cpp
int Fl_Help_View::handle (int event ) [virtual]
```

Handles events in the widget.

Reimplemented from `Fl_Group`.

31.55.3.7  leftline()[1/2]

```cpp
void Fl_Help_View::leftline (int left )
```

Scrolls the text to the indicated position, given a pixel column.

If the given pixel value `left` is out of range, then the text is scrolled to the left or right side of the document, resp.

Parameters

| in  | left | left column number in pixels (0 = left side) |

31.55.3.8  leftline()[2/2]

```cpp
int Fl_Help_View::leftline ( ) const [inline]
```

Gets the left position in pixels.
31.55.3.9 link()

```cpp
void Fl_Help_View::link (  
    Fl_Help_Func * fn ) [inline]
```

This method assigns a callback function to use when a link is followed or a file is loaded (via `Fl_Help_View::load()`) that requires a different file or path.

The callback function receives a pointer to the `Fl_Help_View` widget and the URI or full pathname for the file in question. It must return a pathname that can be opened as a local file or NULL:

```cpp
const char *fn(Fl_Widget *w, const char *uri);
```

The link function can be used to retrieve remote or virtual documents, returning a temporary file that contains the actual data. If the link function returns NULL, the value of the `Fl_Help_View` widget will remain unchanged.

If the link callback cannot handle the URI scheme, it should return the uri value unchanged or set the `value()` of the widget before returning NULL.

31.55.3.10 load()

```cpp
int Fl_Help_View::load (  
    const char * f )
```

Loads the specified file.

This method loads the specified file or URL. The filename may end in a `#name` style target.

If the URL starts with `ftp`, `http`, `https`, `ipp`, `mailto`, or `news`, followed by a colon, FLTK will use `fl_open_uri()` to show the requested page in an external browser.

In all other cases, the URL is interpreted as a filename. The file is read and displayed in this browser. Note that MSWindows style backslashes are not supported in the file name.

**Parameters**

| in | f | filename or URL |

**Returns**

0 on success, -1 on error

**See also**

`fl_open_uri()`
31.55.3.11 resize()

```cpp
void Fl_Help_View::resize (  
    int xx,  
    int yy,  
    int ww,  
    int hh) [virtual]
```

Resizes the help widget.

Reimplemented from Fl_Group.

31.55.3.12 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)

31.55.3.13 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>
<th>Sets the scrollbar size in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 (default)</td>
<td>scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

Generated by Doxygen
See also

Fl::scrollbar_size()

31.55.3.14  select_all()

void Fl_Help_View::select_all ( )

Selects all the text in the view.

31.55.3.15  size()

int Fl_Help_View::size ( ) const [inline]

Gets the size of the help view.

31.55.3.16  textcolor() [1/2]

void Fl_Help_View::textcolor ( 
          Fl_Color c ) [inline]

Sets the default text color.

31.55.3.17  textcolor() [2/2]

Fl_Color Fl_Help_View::textcolor ( ) const [inline]

Returns the current default text color.

31.55.3.18  textfont() [1/2]

void Fl_Help_View::textfont ( 
          Fl_Font f ) [inline]

Sets the default text font.
### 31.55.3.19 `textfont()` [2/2]

```cpp
Fl_Font Fl_Help_View::textfont ( ) const [inline]
```

Returns the current default text font.

### 31.55.3.20 `textsize()` [1/2]

```cpp
void Fl_Help_View::textsize ( Fl_Fontsize s ) [inline]
```

Sets the default text size.

### 31.55.3.21 `textsize()` [2/2]

```cpp
Fl_Fontsize Fl_Help_View::textsize ( ) const [inline]
```

Gets the default text size.

### 31.55.3.22 `title()`

```cpp
const char* Fl_Help_View::title ( ) [inline]
```

Returns the current document title, or NULL if there is no title.

### 31.55.3.23 `topline()` [1/3]

```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></th>
<th></th>
<th>target name</th>
</tr>
</thead>
</table>

*Generated by Doxygen*
31.55.3.24 topline() [2/3]

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

<table>
<thead>
<tr>
<th></th>
<th>top</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>top line number in pixels (0 = start of document)</td>
</tr>
</tbody>
</table>

31.55.3.25 topline() [3/3]

int Fl_Help_View::topline ( ) const [inline]

Returns the current top line in pixels.

31.55.3.26 value() [1/2]

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.

31.55.3.27 value() [2/2]

const char* Fl_Help_View::value ( ) const [inline]

Returns the current buffer contents.

The documentation for this class was generated from the following files:

- Fl_Help_View.H
- Fl_Help_View.cxx
31.56  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.

```cpp
#include <Fl_Hold_Browser.H>
```

Inheritance diagram for Fl_Hold_Browser:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Browser_
   |
   v
Fl_Browser
   |
   v
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.

Additional Inherited Members

31.56.1  Detailed Description

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

![Figure 31.19 Fl_Hold_Browser](image)

As long as the mouse button is held down the item pointed to by it is highlighted, and this highlighting remains on when the mouse button is released. Normally the callback is done when the user releases the mouse, but you can change this with `when()`.

See Fl_Browser for methods to add and remove lines from the browser.

31.56.2  Constructor & Destructor Documentation

Generated by Doxygen
31.56.2.1 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

31.57 Fl_Hor_Fill_Slider Class Reference

Inheritance diagram for Fl_Hor_Fill_Slider:

```
Fl_Widget
  |
  v
Fl_Valuator
  |
  v
Fl_Slider
  |
  v
Fl_Hor_Fill_Slider
```

Public Member Functions

- **Fl_Hor_Fill_Slider** (int X, int Y, int W, int H, const char ∗L=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Hor_Fill_Slider.H
- Fl_Slider.cxx
31.58 Fl_Hor_Nice_Slider Class Reference

Inheritance diagram for Fl_Hor_Nice_Slider:

```plaintext
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)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Hor_Nice_Slider.H
- Fl_Slider.cxx

31.59 Fl_Hor_Slider Class Reference

Horizontal Slider class.

```cpp
#include <Fl_Hor_Slider.H>
```

Inheritance diagram for Fl_Hor_Slider:

```plaintext
Fl_Widget
    Fl_Valuator
        Fl_Slider
            Fl_Hor_Slider
```

Public Member Functions

- **Fl_Hor_Slider** (int X, int Y, int W, int H, const char ∗I=0)

  Creates a new Fl_Hor_Slider widget using the given position, size, and label string.
Additional Inherited Members

31.59.1 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

31.60 Fl_Hor_Value_Slider Class Reference

Inheritance diagram for Fl_Hor_Value_Slider:

```
+-------------------------------------------+     
| Fl_Widget                                 |     
| +---------------------------------------+     
| | Fl_Valuator                           |     
| +---------------------------------------+     
| | Fl_Slider                             |     
| +---------------------------------------+     
| | Fl_Value_Slider                       |     
| +---------------------------------------+     
| | Fl_Hor_Value_Slider                   |     
+-------------------------------------------+
```

Public Member Functions

- **Fl_Hor_Value_Slider** (int X, int Y, int W, int H, const char *l=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Hor_Value_Slider.H
- Fl_Value_Slider.cxx
31.61 Fl_Image Class Reference

Base class for image caching, scaling and drawing.

```
#include <Fl_Image.H>
```

Inheritance diagram for Fl_Image:

```
Fl_Image
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Fl_Bitmap</td>
<td>Fl_Pixmap</td>
</tr>
<tr>
<td>Fl_RGB_Image</td>
<td>Fl_Shared_Image</td>
</tr>
<tr>
<td>Fl_Tiled_Image</td>
<td></td>
</tr>
<tr>
<td>Fl_XBM_Image</td>
<td>Fl_GIF_Image</td>
</tr>
<tr>
<td>Fl_XPM_Image</td>
<td></td>
</tr>
<tr>
<td>Fl_BMP_Image</td>
<td>Fl_JPEG_Image</td>
</tr>
<tr>
<td>Fl_PNG_Image</td>
<td></td>
</tr>
<tr>
<td>Fl_PNM_Image</td>
<td></td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td></td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `virtual class Fl_Shared_Image ∗ as_shared_image ()`
  Returns whether an image is an Fl_Shared_Image or not.
- `virtual void color_average (Fl_Color c, float i)`
  The color_average() method averages the colors in the image with the FLTK color value c.
- `virtual Fl_Image ∗ copy (int W, int H)`
  Creates a resized copy of the specified image.
- `Fl_Image ∗ copy ()`
  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 image depth.
- `const char ∗ const ∗ data () const`
  Returns a pointer to the current image data array.
- `int data_h () const`
  Returns the height of the image data.
- `int data_w () const`
  Returns the width of the image data.
- `virtual void desaturate ()`
  The desaturate() method converts an image to grayscale.
- `virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)`
  Draws the image to the current drawing surface with a bounding box.
- `void draw (int X, int Y)`
  Draws the image to the current drawing surface.
- `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 drawing height in FLTK units.

- **void inactive ()**
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- **virtual void label (Fl_Widget ∗w)**
  The label() methods are an obsolete way to set the image attribute of a widget or menu item.

- **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.

- **int ld () const**
  Returns the current line data size in bytes.

- **virtual void release ()**
  Releases an Fl_Image - the same as 'delete this'.

- **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

- **virtual void uncache ()**
  If the image has been cached for display, delete the cache data.

- **int w () const**
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.

### Static Public Member Functions

- **static Fl_Labeltype define_FL_IMAGE_LABEL ()**
- **static void RGB_scaling (Fl_RGB_Scaling)**
  Sets the RGB image scaling method used for copy(int, int).
- **static Fl_RGB_Scaling RGB_scaling ()**
  Returns the currently used RGB image scaling method.
- **static void scaling_algorithm (Fl_RGB_Scaling algorithm)**
  Sets what algorithm is used when resizing a source image to draw it.
- **static Fl_RGB_Scaling scaling_algorithm ()**
  Gets what algorithm is used when resizing a source image to draw it.

### Static Public Attributes

- **static const int ERR_FILE_ACCESS = -2**
- **static const int ERR_FORMAT = -3**
- **static const int ERR_NO_IMAGE = -1**
- **static bool register_images_done = false**
  True after fl_register_images() was called, false before.
Protected Member Functions

- **void d (int D)**
  
  Sets the current image depth.

- **void data (const char *const *p, int c)**
  
  Sets the current 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.

- **int draw_scaled (int X, int Y, int W, int H)**
  
  Draw the image to the current drawing surface rescaled to a given width and height.

- **void h (int H)**
  
  Sets the height of the image data.

- **void ld (int LD)**
  
  Sets the current line data size in bytes.

- **void w (int W)**
  
  Sets the width of the image data.

Static Protected Member Functions

- **static void labeltype (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)**

- **static void measure (const Fl_Label *lo, int &lw, int &lh)**

Friends

- **class Fl_Graphics_Driver**

31.61.1 Detailed Description

Base class for image caching, scaling and drawing.

**Fl_Image** is the base class used for caching, scaling and drawing all kinds of images in FLTK. This class keeps track of common image data such as the pixels, colormap, width, height, and depth. Virtual methods are used to provide type-specific image handling.

Each image possesses two (width, height) pairs. 1) The width and height of the image raw data are returned by `data_w()` and `data_h()`. These values are set when the image is created and remain unchanged. 2) The width and height of the area filled by the image when it gets drawn are returned by `w()` and `h()`. The values are equal to `data_w()` and `data_h()` when the image is created, and can be changed by the `scale()` member function.

Since the **Fl_Image** class does not support image drawing by itself, calling the `draw()` method results in a box with an X in it being drawn instead.
31.61.2.1 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.

31.61.3 Member Function Documentation

31.61.3.1 as_shared_image()

virtual class Fl_Shared_Image* Fl_Image::as_shared_image ( ) [inline], [virtual]

Returns whether an image is an Fl_Shared_Image or not.

This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_Image.

Since

  1.4.0

Reimplemented in Fl_Shared_Image.

31.61.3.2 color_average()

void Fl_Image::color_average (  
  Fl_Color c,  
  float i ) [virtual]

The color_average() method averages the colors in the image with the 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_SVG_Image, Fl_Shared_Image, Fl_Pixmap, and Fl_Tiled_Image.

31.61.3.3 copy()[1/2]

Fl_Image * Fl_Image::copy (  
  int W,  
  int H ) [virtual]

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.
Parameters

\[ W, H \]
width and height of the returned copied image

Reimplemented in \texttt{Fl_RGB\_Image}, \texttt{Fl\_SVG\_Image}, \texttt{Fl\_Shared\_Image}, \texttt{Fl\_Pixmap}, \texttt{Fl\_Bitmap}, and \texttt{Fl\_Tiled\_Image}.

31.61.3.4 \texttt{copy()} [2/2]

\texttt{Fl\_Image\* Fl\_Image::copy ( ) \[inline\]}

Creates a copy of the specified image.

The image should be deleted (or in the case of \texttt{Fl\_Shared\_Image}, released) when you are done with it.

31.61.3.5 \texttt{count()}

\texttt{int Fl\_Image::count ( ) const \[inline\]}

The \texttt{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.

31.61.3.6 \texttt{d()}

\texttt{int Fl\_Image::d ( ) const \[inline\]}

Returns the image depth.

The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

31.61.3.7 \texttt{data()}

\texttt{const char* const* Fl\_Image::data ( ) const \[inline\]}

Returns a pointer to the current image data array.

Use the \texttt{count()} method to find the size of the data array.

31.61.3.8 \texttt{desaturate()}

\texttt{void Fl\_Image::desaturate ( ) \[virtual\]}

The \texttt{desaturate()} method converts an image to grayscale.

If the image contains an alpha channel (depth = 4), the alpha channel is preserved.

An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

Reimplemented in \texttt{Fl\_RGB\_Image}, \texttt{Fl\_SVG\_Image}, \texttt{Fl\_Shared\_Image}, \texttt{Fl\_Pixmap}, and \texttt{Fl\_Tiled\_Image}. 

Generated by Doxygen
31.61.3.9 draw() [1/2]

```cpp
void Fl_Image::draw(
    int X,
    int Y,
    int W,
    int H,
    int cx = 0,
    int cy = 0) [virtual]
```

Draws the image to the current drawing surface with a bounding box.

Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.

In other words: `fl_push_clip(X,Y,W,H)` is applied, the image is drawn with its upper-left corner at X-cx, Y-cy and its own width and height, `fl_pop_clip()` is applied.

Reimplemented in Fl_RGB_Image, Fl_SVG_Image, Fl_Shared_Image, Fl_Pixmap, Fl_Bitmap, and Fl_Tiled_← Image.

31.61.3.10 draw() [2/2]

```cpp
void Fl_Image::draw(
    int X,
    int Y) [inline]
```

Draws the image to the current drawing surface.

Parameters

| X, Y | specify the upper-lefthand corner of the image. |

31.61.3.11 draw_empty()

```cpp
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.
31.61.3.12 draw_scaled()

```
int Fl_Image::draw_scaled (  
    int X,  
    int Y,  
    int W,  
    int H ) [protected]
```

Draw the image to the current drawing surface rescaled to a given width and height.

Intended for internal use by the FLTK library.

**Parameters**

<table>
<thead>
<tr>
<th>X, Y</th>
<th>position of the image's top-left</th>
</tr>
</thead>
<tbody>
<tr>
<td>W, H</td>
<td>width and height for the drawn image</td>
</tr>
</tbody>
</table>

**Returns**

1

*Deprecated* Only for API compatibility with FLTK 1.3.4.

31.61.3.13 fail()

```
int Fl_Image::fail ( )
```

Returns a value that is not 0 if there is currently no image available.

**Example use:**

```c
[..]  
Fl_Box box(X,Y,W,H);
Fl_JPEG_Image jpg("/tmp/foo.jpg");
switch ( jpg.fail() ) {  
    case Fl_Image::ERR_NO_IMAGE:  
        fl_alert("/tmp/foo.jpg: %s", strerror(errno)); // shows actual os error to user
        exit(1);  
    case Fl_Image::ERR_FILE_ACCESS:  
        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.
31.61.3.14  h() [1/2]

void Fl_Image::h (  
       int \texttt{H} ) \texttt{[inline]}, \texttt{[protected]}  

Sets the height of the image data.  

This protected function sets both image heights: the height of the image data returned by \texttt{data_h()} and the image drawing height in FLTK units returned by \texttt{h()}.

31.61.3.15  h() [2/2]

\begin{verbatim}  
int Fl_Image::h ( ) \texttt{const \ [inline]}  
\end{verbatim}  

Returns the current image drawing height in FLTK units.  

The values of \texttt{h()} and \texttt{data_h()} are identical unless \texttt{scale()} has been called after which they may differ.

31.61.3.16  inactive()

void Fl_Image::inactive ( ) \texttt{[inline]}  

The \texttt{inactive()} method calls \texttt{color\_average(FL\_BACKGROUND\_COLOR, 0.33f)} to produce an image that appears grayed out.  

An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

31.61.3.17  label() [1/2]

\begin{verbatim}  
void Fl_Image::label (  
       Fl_Widget * \texttt{widget} ) \texttt{[virtual]}  
\end{verbatim}  

The \texttt{label()} methods are an obsolete way to set the image attribute of a widget or menu item.  

Use the \texttt{image()} or \texttt{deimage()} methods of the \texttt{Fl\_Widget} and \texttt{Fl\_Menu\_Item} classes instead.  

Reimplemented in \texttt{Fi\_RGB\_Image}, \texttt{Fi\_Pixmap}, and \texttt{Fi\_Bitmap}.

31.61.3.18  label() [2/2]

\begin{verbatim}  
void Fl_Image::label (  
       Fl_Menu_Item * \texttt{m} ) \texttt{[virtual]}  
\end{verbatim}  

The \texttt{label()} methods are an obsolete way to set the image attribute of a widget or menu item.  

Use the \texttt{image()} or \texttt{deimage()} methods of the \texttt{Fl\_Widget} and \texttt{Fl\_Menu\_Item} classes instead.  

Reimplemented in \texttt{Fi\_RGB\_Image}, \texttt{Fi\_Pixmap}, and \texttt{Fi\_Bitmap}.  

---

Generated by Doxygen
31.61.3.19 ld() [1/2]

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.

31.61.3.20 ld() [2/2]

int Fl_Image::ld () const [inline]

Returns the current line data size in bytes.

See also

    ld(int)

31.61.3.21 release()

virtual void Fl_Image::release () [inline], [virtual]

Releases an Fl_Image - the same as 'delete this'.

This virtual method is for almost all image classes the same as calling

delete image;

where image is an Fl_Image * pointer.

However, for subclass Fl_Shared_Image this virtual method is reimplemented and maintains shared images.

This virtual method makes it possible to delete all image types in the same way by calling

image->release();

Reasoning: If you have an 'Fl_Image *' pointer and don't know if the object is one of the class Fl_Shared_Image or any other subclass of Fl_Image (for instance Fl_RGB_Image) then you can't just use operator delete since this is not appropriate for Fl_Shared_Image objects.

The virtual method release() handles this properly.

Since

    1.4.0 in the base class Fl_Image and virtual in Fl_Shared_Image

Reimplemented in Fl_Shared_Image.
31.61.3.22 RGB_scaling() [1/2]

```cpp
void Fl_Image::RGB_scaling (Fl_RGB_Scaling method) [static]
```

Sets the RGB image scaling method used for `copy(int, int)`.

Applies to all RGB images, defaults to `FL_RGB_SCALING_NEAREST`.

31.61.3.23 RGB_scaling() [2/2]

```cpp
Fl_RGB_Scaling Fl_Image::RGB_scaling () [static]
```

Returns the currently used RGB image scaling method.

31.61.3.24 scale()

```cpp
void Fl_Image::scale (int width, int height, int proportional = 1, int can_expand = 0) [virtual]
```

Sets the drawing size of the image.

This function controls the values returned by member functions `w()` and `h()` which in turn control how the image is drawn: the full image data (whose size is given by `data_w()` and `data_h()`) are drawn scaled to an area of the drawing surface sized at `w() x h()` FLTK units. This can make a difference if the drawing surface has more than 1 pixel per FLTK unit because the image can be drawn at the full resolution of the drawing surface. Examples of such drawing surfaces: HiDPI displays, laser printers, PostScript files, PDF printers.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>width, height</code></td>
<td>maximum values, in FLTK units, that <code>w()</code> and <code>h()</code> should return</td>
</tr>
<tr>
<td><code>proportional</code></td>
<td>if not null, keep the values returned by <code>w()</code> and <code>h()</code> proportional to <code>data_w()</code> and <code>data_h()</code></td>
</tr>
<tr>
<td><code>can_expand</code></td>
<td>if null, the values returned by <code>w()</code> and <code>h()</code> will not be larger than <code>data_w()</code> and <code>data_h()</code>, respectively</td>
</tr>
</tbody>
</table>

**Note**

This function generally changes the values returned by the `w()` and `h()` member functions. In contrast, the values returned by `data_w()` and `data_h()` remain unchanged.

**Version**

1.4 (1.3.4 and FL_ABI_VERSION for `Fl_Shared_Image` only)

**Example code:** scale an image to fit in a box
31.61 Fl_Image Class Reference

```cpp
Fl_Box *b = ... // a box
Fl_Image *img = new Fl_PNG_Image("/path/to/picture.png"); // read a picture file
// set the drawing size of the image to the size of the box keeping its aspect ratio
img->scale(b->w(), b->h());
b->image(img); // use the image as the box image
```

### 31.61.3.25 scaling_algorithm() [1/2]

```cpp
static void Fl_Image::scaling_algorithm {
    Fl_RGB_Scaling algorithm } [inline], [static]
```

Sets what algorithm is used when resizing a source image to draw it.

The default algorithm is FL_RGB_SCALING_BILINEAR. Drawing an Fl_Image is sometimes performed by first resizing the source image and then drawing the resized copy. This occurs, e.g., when drawing to screen under X11 without Xrender support after having called scale(). This function controls what method is used when the image to be resized is an Fl_RGB_Image.

**Version**

1.4

### 31.61.3.26 scaling_algorithm() [2/2]

```cpp
static Fl_RGB_Scaling Fl_Image::scaling_algorithm () [inline], [static]
```

Gets what algorithm is used when resizing a source image to draw it.

### 31.61.3.27 uncache()

```cpp
void Fl_Image::uncache () [virtual]
```

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented in Fl_RGB_Image, Fl_Shared_Image, Fl_Pixmap, and Fl_Bitmap.

### 31.61.3.28 w() [1/2]

```cpp
void Fl_Image::w ( int W ) [inline], [protected]
```

Sets the width of the image data.

This protected function sets both image widths: the width of the image data returned by data_w() and the image drawing width in FLTK units returned by w().

Generated by Doxygen
31.61.3.29  \texttt{w()} [2/2]

\begin{verbatim}
int Fl_Img::w ( ) const [inline]
\end{verbatim}

Returns the current image drawing width in FLTK units.

The values of \texttt{w()} and \texttt{data\_w()} are identical unless \texttt{scale()} has been called after which they may differ.

The documentation for this class was generated from the following files:

- \texttt{Fl\_Image.H}
- \texttt{Fl\_Image.cxx}

31.62  \texttt{Fl\_Image\_Reader} Class Reference

\textbf{Public Member Functions}

- \texttt{const char \* name ()}
- \texttt{int open(const char \* filename)}
- \texttt{int open(const char \* imagename, const unsigned char \* data)}
- \texttt{unsigned char read\_byte ()}
- \texttt{unsigned int read\_dword ()}
- \texttt{int read\_long ()}
- \texttt{unsigned short read\_word ()}
- \texttt{void seek(unsigned int n)}

The documentation for this class was generated from the following files:

- \texttt{Fl\_Image\_Reader.h}
- \texttt{Fl\_Image\_Reader.cxx}

31.63  \texttt{Fl\_Image\_Surface} Class Reference

Directs all graphics requests to a \texttt{Fl\_Image}.

\begin{verbatim}
#include <Fl\_Image\_Surface.H>
\end{verbatim}

Inheritance diagram for \texttt{Fl\_Image\_Surface}:

```
Fl_Surface_Device
   \|-- Fl_Widget_Surface
      \|-- Fl_Image_Surface
```

Generated by Doxygen
Public Member Functions

- **FL_Image_Surface** (int w, int h, int high_res=0, Fl_Offscreen off=0)
  Constructor with optional high resolution.

- **FL_Shared_Image** + highres_image ()
  Returns a possibly high resolution image made of all drawings sent to the FL_Image_Surface object.

- **FL_RGB_Image** + image ()
  Returns an image made of all drawings sent to the FL_Image_Surface object.

- virtual bool is_current ()
  Is this surface the current drawing surface?

- **FL_Offscreen** offscreen ()
  Returns the FL_Offscreen object associated to the image surface.

- void origin (int +x, int +y)
  Computes the coordinates of the current origin of graphics functions.

- void origin (int x, int y)
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- int printable_rect (int +w, int +h)
  Computes the width and height of the drawable area of the drawing surface.

- void rescale ()
  Adapts the FL_Image_Surface object to the new value of the GUI scale factor.

- void set_current ()
  Make this surface the current drawing surface.

- ~FL_Image_Surface ()
  The destructor.

Protected Member Functions

- void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.

- void untranslate ()
  Undoes the effect of a previous translate() call.

Friends

- class FL_Graphics_Driver

Additional Inherited Members

31.63.1 Detailed Description

Directs all graphics requests to an FL_Image.

After creation of an FL_Image_Surface object, make it the current drawing surface calling FL_Surface_Device::<push_current>(), and all subsequent graphics requests will be recorded in the image. It's possible to draw widgets (using FL_Image_Surface::draw()) or to use any of the Drawing functions or the Color & Font functions. Finally, call image() on the object to obtain a newly allocated FL_RGB_Image object. FL_Gl_Window objects can be drawn in the image as well.

Usage example:
// this is the widget that you want to draw into an image
Fl_Widget *g = ...;

// create an Fl_Image_Surface object
Fl_Image_Surface *image_surface = new Fl_Image_Surface(g->w(), g->h());

// direct all further graphics requests to the image
Fl_Surface_Device::push_current(image_surface);

// draw a white background
fl_color(FL_WHITE);
fl_rectf(0, 0, g->w(), g->h());

// draw the g widget in the image
image_surface->draw(g);

// get the resulting image
Fl_RGB_Image *image = image_surface->image();

// direct graphics requests back to their previous destination
Fl_Surface_Device::pop_current();

// delete the image_surface object, but not the image itself
delete image_surface;

31.63.2 Constructor & Destructor Documentation

31.63.2.1 Fl_Image_Surface()

Fl_Image_Surface::Fl_Image_Surface {
    int w,
    int h,
    int high_res = 0,
    Fl_Offscreen off = 0
}

Constructor with optional high resolution.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>set the size of the resulting image. The value of the high_res parameter controls whether w and h are interpreted as pixel or FLTK units.</td>
</tr>
<tr>
<td>h</td>
<td>If zero, the created image surface is sized at w x h pixels. If non-zero, the pixel size of the created image surface depends on the value of the display scale factor (see Fl_Graphics_Driver::scale()): the resulting image has the same number of pixels as an area of the display of size w x h expressed in FLTK units.</td>
</tr>
<tr>
<td>high_res</td>
<td>If not null, the image surface is constructed around a pre-existing Fl_Offscreen. The caller is responsible for both construction and destruction of this Fl_Offscreen object. Is mostly intended for internal use by FLTK.</td>
</tr>
</tbody>
</table>

Version

1.3.4 (1.3.3 without the highres parameter)
31.63.2.2  \texttt{\~F}_{l} \texttt{Image\_Surface()}

\texttt{F}_l \texttt{Image\_Surface::\~F}_l \texttt{Image\_Surface ( )}

The destructor.

31.63.3  Member Function Documentation

31.63.3.1  highres\_image()

\texttt{F}_l \texttt{Shared\_Image} \ast \texttt{F}_l \texttt{Image\_Surface::highres\_image ( )}

Returns a possibly high resolution image made of all drawings sent to the \texttt{F}_l \texttt{Image\_Surface} object.

The \texttt{F}_l \texttt{Image\_Surface} object should have been constructed with \texttt{F}_l \texttt{Image\_Surface(W, H, 1)}. The returned \texttt{F}_l \texttt{Shared\_Image} object is scaled to a size of WxH FLTK units and may have a pixel size larger than these values. The returned object should be deallocated with \texttt{F}_l \texttt{Shared\_Image::release()} after use.

\textbf{Deprecated} Use \texttt{image()} instead.

Version

1.4 (1.3.4 for MacOS platform only)

31.63.3.2  image()

\texttt{F}_l \texttt{RGB\_Image} \ast \texttt{F}_l \texttt{Image\_Surface::image ( )}

Returns an image made of all drawings sent to the \texttt{F}_l \texttt{Image\_Surface} object.

The returned object contains its own copy of the RGB data. The caller is responsible for deleting the image.

31.63.3.3  offscreen()

\texttt{F}_l \texttt{Offscreen} \texttt{F}_l \texttt{Image\_Surface::offscreen ( )}

Returns the \texttt{F}_l \texttt{Offscreen} object associated to the image surface.

The returned \texttt{F}_l \texttt{Offscreen} object is deleted when the \texttt{F}_l \texttt{Image\_Surface} object is deleted, unless the \texttt{F}_l \texttt{Image\_Surface} was constructed with non-null \texttt{F}_l \texttt{Offscreen} argument.

31.63.3.4  origin() [1/2]

\texttt{void F}_l \texttt{Image\_Surface::origin (}
\begin{verbatim}
    int \ast x,
    int \ast y ) [virtual]
\end{verbatim}

Computes the coordinates of the current origin of graphics functions.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>x, y</th>
</tr>
</thead>
</table>

If non-null, *x* and *y* are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from `Fl_Widget_Surface`.

### 31.63.3.5 origin()

```c
void Fl_Image_Surface::origin (int x, int y) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x, y</th>
</tr>
</thead>
</table>

Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from `Fl_Widget_Surface`.

### 31.63.3.6 printable_rect()

```c
int Fl_Image_Surface::printable_rect (int *w, int *h) [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

Returns

0 if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`. 
31.63.3.7 rescale()

void Fl_Image_Surface::rescale ( )

Adapts the Fl_Image_Surface object to the new value of the GUI scale factor.

The Fl_Image_Surface object must not be the current drawing surface. This function is useful only for an object constructed with non-zero high_res parameter.

Version
1.4

31.63.3.8 set_current()

void Fl_Image_Surface::set_current ( ) [virtual]

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Reimplemented from Fl_Surface_Device.

31.63.3.9 translate()

void Fl_Image_Surface::translate ( int x, int y ) [protected], [virtual]

Translates the current graphics origin accounting for the current rotation.

Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following files:

- Fl_Image_Surface.H
- Fl_Image_Surface.hxx

Generated by Doxygen
31.64  **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_          Fl_Widget
                      /
                      /
Fl_File_Input Fl_Float_Input Fl_Int_Input Fl_Multiline_Input Fl_Output Fl_Secret_Input Fl_Spinner::Fl_Spinner_Input
                           /
                           /
Fl_Multiline_Output
```

**Public Member Functions**

- `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.

**Protected Member Functions**

- `void draw()`
  
  Draws the widget.

**Friends**

- `class Fl_Cocoa_Screen_Driver`
- `class Fl_Screen_Driver`

**Additional Inherited Members**

**31.64.1 Detailed Description**

This is the FLTK text input widget.

It displays a single line of text and lets the user edit it. Normally it is drawn with an inset box and a white background. The text may contain any characters, and will correctly display any UTF text, using `\X` notation for unprintable control characters. It assumes the font can draw any characters of the used scripts, which is true for standard fonts under Windows and Mac OS X. Characters can be input using the keyboard or the character palette/map. Character composition is done using dead keys and/or a compose key as defined by the operating system.

**Table 31.189 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</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Table 31.190 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 Line Feed.</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</td>
<td>Command-X, Shift-Delete</td>
<td>Cut.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Copy the selection to the clipboard and delete it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(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>
<tr>
<td>Home</td>
<td>Command-Up, Command-Left</td>
<td>Move to start of line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>End</td>
<td>Command-Down, Command-Right</td>
<td>Move to end of line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-Home</td>
<td>Command-Up, Command-PgUp, Ctrl-Left</td>
<td>Move to top of document/field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In single line input, moves to start of line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In multiline input, moves to start of top line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-End</td>
<td>Command-End, Command-PgDn, Ctrl-Right</td>
<td>Move to bottom of document/field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In single line input, moves to end of line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In multiline input, moves to end of last line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-Left</td>
<td>Alt-Left</td>
<td>Word left.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-Right</td>
<td>Alt-Right</td>
<td>Word right.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-Backspace</td>
<td>Alt-Backspace</td>
<td>Delete word left.</td>
</tr>
<tr>
<td>Ctrl-Delete</td>
<td>Alt-Delete</td>
<td>Delete word right.</td>
</tr>
</tbody>
</table>
31.64.2 Constructor & Destructor Documentation

31.64.2.1 Fl_Input()

Fl_Input::Fl_Input (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0  
)

Creates a new Fl_Input widget using the given position, size, and label string.

The default boxtype is FL_DOWN_BOX.

31.64.3 Member Function Documentation

31.64.3.1 draw()

void Fl_Input::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:

    Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
    s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

31.64.3.2 handle()

int Fl_Input::handle (  
    int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.
Parameters

| in | event | the kind of event received |

Return values

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

Reimplemented in Fl_Spinner::Fl_Spinner_Input, and Fl_Secret_Input.

The documentation for this class was generated from the following files:

- Fl_Input.H
- Fl_Input.cxx

31.65 Fl_Input__ Class Reference

This class provides a low-overhead text input field.

#include <Fl_Input_.H>

Inheritance diagram for Fl_Input__:

```
Fl_Widget
  Fl_Input
    Fl_File_Input Fl_Float_Input Fl_Int_Input Fl_Multiline_Input Fl_Output Fl_Secret_Input Fl_Spinner::Fl_Spinner_Input
```

Public Member Functions

- int append (const char *t, int l=0, char keep_selection=0)
  Append text at the end.

- 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 n)
  Deletes the next n bytes rounded to characters before or after the cursor.
• int cut(int a, int b)
  Deletes all characters between index a and b.
• 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, int m)
  Sets the index for the cursor and mark.
• int position(int p)
  Sets the cursor position 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.
• void tab_nav(int val)
Sets whether the Tab key does focus navigation, or inserts tab characters into \texttt{Fl\_Multiline\_Input}.

- \textbf{int tab\_nav () const}
  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.

- \textbf{Fl\_Color textcolor () const}
  
  Gets the color of the text in the input field.

- \textbf{void textcolor (Fl\_Color n)}
  
  Sets the color of the text in the input field.

- \textbf{Fl\_Font textfont () const}
  
  Gets the font of the text in the input field.

- \textbf{void textfont (Fl\_Font s)}
  
  Sets the font of the text in the input field.

- \textbf{Fl\_Fontsize textsize () const}
  
  Gets the size of the text in the input field.

- \textbf{void textsize (Fl\_Fontsize s)}
  
  Sets the size of the text in the input field.

- \textbf{int undo ()}
  
  Undoes previous changes to the text buffer.

- \textbf{int value (const char* \_)}
  
  Changes the widget text.

- \textbf{int value (const char\* \_; int)}
  
  Changes the widget text.

- \textbf{const char\* value () const}
  
  Returns the text displayed in the widget.

- \textbf{int wrap () const}
  
  Gets the word wrapping state of the input field.

- \textbf{void wrap (int b)}
  
  Sets the word wrapping state of the input field.

- \textbf{\~Fl\_Input\_ ()}
  
  Destroys the widget.

Protected Member Functions

- \textbf{void drawtext (int, int, int, int)}
  
  Draws the text in the passed bounding box.

- \textbf{void handle\_mouse (int, int, int, int, int keepmark=0)}
  
  Handles mouse clicks and mouse moves.

- \textbf{int handle\_text (int e, int, int, int, int)}
  
  Handles all kinds of text field related events.

- \textbf{int line\_end (int i) const}
  
  Finds the end of a line.

- \textbf{int line\_start (int i) const}
  
  Finds the start of a line.

- \textbf{int linesPerPage ()}

- \textbf{void maybe\_do\_callback ()}

- \textbf{int up\_down\_position (int, int keepmark=0)}
  
  Moves the cursor to the column given by \texttt{up\_down\_pos}.

- \textbf{int word\_end (int i) const}
  
  Finds the end of a word.

- \textbf{int word\_start (int i) const}
  
  Finds the start of a word.

- \textbf{int xscroll () const}

- \textbf{int yscroll () const}

- \textbf{void yscroll (int yOffset)}
Additional Inherited Members

31.65.1 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:

```c
#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
Fl_Widget::shortcut_label(int)

31.65.2 Constructor & Destructor Documentation

31.65.2.1 Fl_Input()

```c
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

<table>
<thead>
<tr>
<th>X, Y, W, H</th>
<th>the dimensions of the new widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>an optional label text</td>
</tr>
</tbody>
</table>

31.65.2.2 Fl_Input::~Fl_Input()

```
Fl_Input_::~Fl_Input_ ()
```

Destroys the widget.

The destructor clears all allocated buffers and removes the widget from the parent Fl_Group.

31.65.3 Member Function Documentation

31.65.3.1 append()

```
int Fl_Input_::append (  
    const char * t,  
    int l = 0,  
    char keep_selection = 0 )
```

Append text at the end.

This function appends the string in t to the end of the text. It does not moves the new position or mark.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>text that will be appended</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>l</td>
<td>length of text, or 0 if the string is terminated by null.</td>
</tr>
<tr>
<td>in</td>
<td>keep_selection</td>
<td>if this is 1, the current text selection will remain, if 0, the cursor will move to the end of the inserted text.</td>
</tr>
</tbody>
</table>

Returns

0 if no text was appended

31.65.3.2 copy()

```
int Fl_Input_::copy (  
    int clipboard )
```

Generated by Doxygen
Put the current selection into the clipboard.

This function copies the current selection between mark() and position() into the specified clipboard. This does not replace the old clipboard contents if position() and mark() are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.

Parameters

| clipboard | the clipboard destination 0 or 1 |

Returns

0 if no text is selected, 1 if the selection was copied

See also

Fl::copy(const char *, int, int)

31.65.3.3 copy_cuts()

int Fl_Input_::copy_cuts ( )

Copies the yank buffer to the clipboard.

This method copies all the previous contiguous cuts from the undo information to the clipboard. This function implements the ^K shortcut key.

Returns

0 if the operation did not change the clipboard

See also

copy(int), cut()

31.65.3.4 cursor_color() [1/2]

Fl_Color Fl_Input_::cursor_color ( ) const [inline]

Gets the color of the cursor.

Returns

the current cursor color

31.65.3.5 cursor_color() [2/2]

void Fl_Input_::cursor_color ( Fl_Color n ) [inline]

Sets the color of the cursor.

The default color for the cursor is FL_BLACK.
Parameters

| in | \( n \) | the new cursor color |

31.65.3.6 cut() [1/3]

```c
int Fl_Input_::cut ( ) [inline]
```

Deletes the current selection.

This function deletes the currently selected text \textit{without} storing it in the clipboard. To use the clipboard, you may call \texttt{copy()} first or \texttt{copy_cuts()} after this call.

Returns

0 if no data was copied

31.65.3.7 cut() [2/3]

```c
int Fl_Input_::cut ( 
    int \( n \) ) [inline]
```

Deletes the next \( n \) bytes rounded to characters before \textit{or} after the cursor.

This function deletes the currently selected text \textit{without} storing it in the clipboard. To use the clipboard, you may call \texttt{copy()} first or \texttt{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

31.65.3.8 cut() [3/3]

```c
int Fl_Input_::cut ( 
    int \( a \),
    int \( b \) ) [inline]
```

Deletes all characters between index \( a \) and \( b \).

This function deletes the currently selected text \textit{without} storing it in the clipboard. To use the clipboard, you may call \texttt{copy()} first or \texttt{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

31.65.3.9 \texttt{drawtext()}

\begin{verbatim}
void Fl_Input__::drawtext (  
    int \textit{X},  
    int \textit{Y},  
    int \textit{W},  
    int \textit{H} ) [protected]
\end{verbatim}

Draws the text in the passed bounding box.

If \texttt{damage()} & \texttt{FL\_DAMAGE\_ALL} is true, this assumes the area has already been erased to \texttt{color()}. Otherwise it does minimal update and erases the area itself.

Parameters

\[ X, Y, W, H \] area that must be redrawn

31.65.3.10 \texttt{handle\_mouse()}

\begin{verbatim}
void Fl_Input__::handle\_mouse (  
    int \textit{X},  
    int \textit{Y},  
    int \textit{},  
    int \textit{},  
    int \textit{drag = 0} ) [protected]
\end{verbatim}

Handles mouse clicks and mouse moves.

\textbf{Todo} Add comment and parameters
31.65.3.11 handletext()

```c
int Fl_Input_::handletext ( int event, int X, int Y, int W, int H ) [protected]
```

Handles all kinds of text field related events.

This is called by derived classes.

**Todo** Add comment and parameters

31.65.3.12 index()

```c
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**

| in | i | index into the value field |

Returns

the character at index `i`

31.65.3.13 input_type()[1/2]

```c
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.

Returns

the current input type

31.65.3.14 input_type()[2/2]

```c
void Fl_Input_::input_type ( int t ) [inline]
```

Sets the input field type.

A `redraw()` is required to reformat the input field.
Parameters

| in  | f         | new input type |

31.65.3.15  insert()

```c
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  | f         | text that will be inserted |
| in  | f         | length of text, or 0 if the string is terminated by `nul`. |

Returns

0 if no text was inserted

31.65.3.16  line_end()

```c
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
31.65.3.17  line_start()

```cpp
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

31.65.3.18  mark()[1/2]

```cpp
int Fl_Input_::mark ( ) const [inline]
```

Gets the current selection mark.

Returns

- index into the text

31.65.3.19  mark()[2/2]

```cpp
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)

31.65.3.20  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().

31.65.3.21  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).

31.65.3.22  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)

31.65.3.23  position() [2/3]

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)

---

31.65.3.24 position() [3/3]

```c
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)

---

31.65.3.25 readonly() [1/2]

```c
int Fl_Input_::readonly ( ) const [inline]
```

Gets the read-only state of the input field.

Returns

non-zero if this widget is read-only
31.65.3.26 readonly() [2/2]

void 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 |

31.65.3.27 replace()

int Fl_Input_::replace (  
  int b,  
  int e,  
  const char ∗ text,  
  int ilen = 0 )

Deletes text from b to e and inserts the new string text.

All changes to the text buffer go through this function. It deletes the region between b and e (either one may be less or equal to the other), and then inserts the string text at that point and moves the mark() and position() to the end of the insertion. Does the callback if when() & FL_WHEN_CHANGED and there is a change.

Set b and e equal to not delete anything. Set text to NULL to not insert anything.

ilen can be zero or strlen(text), which saves a tiny bit of time if you happen to already know the length of the insertion, or can be used to insert a portion of a string. If ilen is zero, strlen(text) is used instead.

b and e are clamped to the 0..<size()> range, so it is safe to pass any values. b, e, and ilen are used as numbers of bytes (not characters), where b and e count from 0 to size() (end of buffer).

If b and/or e don't point to a valid UTF-8 character boundary, they are adjusted to the previous (b) or the next (e) valid UTF-8 character boundary, resp..

If the current number of characters in the buffer minus deleted characters plus inserted characters in text would overflow the number of allowed characters (maximum_size()), then only the first characters of the string are inserted, so that maximum_size() is not exceeded.

cut() and insert() are just inline functions that call replace().

Parameters

| 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).

31.65.3.28 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)`

Reimplemented from `Fl_Widget`.

31.65.3.29 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()`
31.65.3.30  shortcut() [2/2]

`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()`

31.65.3.31  size() [1/2]

`int Fl_Input_::size () const` [inline]

Returns the number of bytes in `value()`.
This may be greater than `strlen(value())` if there are null characters in the text.

Returns

number of bytes in the text

31.65.3.32  size() [2/2]

`void Fl_Input_::size (int W, int H)` [inline]

Sets the width and height of this widget.

Parameters

|  in  | W,H | new width and height |

See also

`Fl_Widget::size(int, int)`
31.65.3.33 static_value() [1/2]

int Fl_Input_::static_value (  
    const char * str )

Changes the widget text.

This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.

Parameters

| in | str | the new text |

Returns

non-zero if the new value is different than the current one

31.65.3.34 static_value() [2/2]

int Fl_Input_::static_value (  
    const char * str,  
    int len )

Changes the widget text.

This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.

You can use the len parameter to directly set the length if you know it already or want to put null characters in the text.

Parameters

| 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
31.65.3.35  tab_nav() [1/2]

void Fl_Input_::tab_nav (  
    int val ) [inline]

Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.

By default this flag is enabled to provide the 'normal' behavior most users expect; Tab navigates focus to the next
widget. To inserting an actual Tab character, users can use Ctrl-I or copy/paste.

Disabling this flag gives the old FLTK behavior where Tab inserts a tab character into the text field, in which case
only the mouse can be used to navigate to the next field.

History: This flag was provided for backwards support of FLTK's old 1.1.x behavior where Tab inserts a tab character
instead of navigating focus to the next widget. This behavior was unique to Fl_Multiline_Input. With the advent of
Fl_Text_Editor, this old behavior has been deprecated.

Parameters

| in  | val | If val is 1, Tab advances focus (default).  
|     |     | If val is 0, Tab inserts a tab character (old FLTK behavior). |

See also

    tab_nav(), Fl::OPTION_ARROW_FOCUS.

31.65.3.36  tab_nav() [2/2]

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.
31.65.3.37  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)

31.65.3.38  textcolor() [2/2]

    void Fl_Input_::textcolor ( 
            Fl_Color n ) [inline]

Sets the color of the text in the input field.

The text color defaults to FL_FOREGROUND_COLOR.

Parameters
    in  n  new text color

See also
    textcolor()}

31.65.3.39  textfont() [1/2]

    Fl_Font Fl_Input_::textfont ( ) const [inline]

Gets the font of the text in the input field.

Returns
    the current Fl_Font index

31.65.3.40  textfont() [2/2]

    void Fl_Input_::textfont ( 
            Fl_Font s ) [inline]

Sets the font of the text in the input field.

The text font defaults to FL_HELVETICA.

Generated by Doxygen
**Parameters**

| in  | s     | the new text font |

### 31.65.3.41 `textsize()` [1/2]

```cpp
Fl_Fontsize Fl_Input_::textsize ( ) const [inline]
```

Gets the size of the text in the input field.

**Returns**

the text height in pixels

### 31.65.3.42 `textsize()` [2/2]

```cpp
void Fl_Input_::textsize ( Fl_Fontsize s ) [inline]
```

Sets the size of the text in the input field.

The text height defaults to `FL_NORMAL_SIZE`.

**Parameters**

| in  | s     | the new font height in pixel units |

### 31.65.3.43 `undo()`

```cpp
int Fl_Input_::undo ( )
```

Undoes previous changes to the text buffer.

This call undoes a number of previous calls to `replace()`.

**Returns**

non-zero if any change was made.
### up_down_position()

```cpp
int Fl_Input_::up_down_position ( int i, int keepmark = 0 ) [protected]
```

Moves the cursor to the column given by `up_down_pos`.

This function is helpful when implementing up and down cursor movement. It moves the cursor from the beginning of a line to the column indicated by the global variable `up_down_pos` in pixel units.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>index into the beginning of a line of text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>keepmark</td>
<td>if set, move only the cursor, but not the mark</td>
</tr>
</tbody>
</table>

**Returns**

- index to new cursor position

### value()

```cpp
int Fl_Input_::value ( const char * str )
```

Changes the widget text.

This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal buffer. Passing NULL is the same as " ".

**Parameters**

| in   | str     | the new text |

**Returns**

- non-zero if the new value is different than the current one

**See also**

- `Fl_Input_::value(const char * str, int len)`, `Fl_Input_::value()`
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()

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*)

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

31.65.3.49 word_start()

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 |

Returns

start of the word, or previous word

31.65.3.50 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.

31.65.3.51 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

Generated by Doxygen
31.66  Fl_Input_Choice Class Reference

A combination of the input widget and a menu button.

Inheritance diagram for Fl_Input_Choice:

```
Fl_Widget
   |
   V
Fl_Group
   |
   V
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_Input_Choice widget.
- **void** set_changed ()
  
  Sets the changed() state of both input and menu button widgets to the specified value.
- **FL_Color** textcolor () const
  
  Gets the Fl_Input text field's text color.
- **void** textcolor (FL_Color c)
  
  Sets the Fl_Input text field's text color to 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.

- int update_menubutton ()
  
  Updates the menubutton with the string value in Fl_Input.

- 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.

Additional Inherited Members

31.66.1 Detailed Description

A combination of the input widget and a menu button.

![Figure 31.20 Fl_Input_Choice widget](image)

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;
    printf("*** Choice Callback:\n");
    printf(" widget's text value='\\%s'\n", choice->value()); // normally all you need

    // Access the menu via menubutton()...
    const Fl_Menu_Item *item = choice->menubutton()->mvalue();
    printf(" item label()='\\%s'\n", item ? item->label() : "(No item)" );
    printf(" item value()=%d\n", choice->menubutton()->value() );
    printf(" input value()='\\%s'\n", choice->input()->value() );
    printf(" The user %s\n", choice->menubutton()->changed() ? "picked a menu item" : "typed text");
}

int main() {
    Fl_Double_Window win(200,100,"Input Choice");
    win.begin();
    Fl_Input_Choice choice(10,10,100,30);
    choice.callback(choice_cb, 0);
    choice.add("Red");
    choice.add("Orange");
    choice.add("Yellow");
    //choice.value("Red"); // uncomment to make "Red" default
    win.end();
    win.show();
    return Fl::run();
}
```

31.66.2 Constructor & Destructor Documentation

31.66.2.1 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.

31.66.3 Member Function Documentation
31.66.3.1 add()

void Fl_Input_Choice::add (    
    const char * s ) [inline]  

Adds an item to the menu.

When any item is selected, the Fl_Input_Choice callback() is invoked, which can do something with the selected item.

You can access the more complex Fl_Menu_Button::add() methods (setting item-specific callbacks, userdata, etc), via menubutton(). Example:

```cpp
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Fonts");
Fl_Menu_Button *mb = choice->menubutton(); // use Fl_Input_Choice’s
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);
```

31.66.3.2 changed()

int Fl_Input_Choice::changed ( ) const [inline]

Returns the combined changed() state of the input and menu button widget.

31.66.3.3 clear()

void Fl_Input_Choice::clear ( ) [inline]

Removes all items from the menu.

31.66.3.4 clear_changed()

void Fl_Input_Choice::clear_changed ( )

Clears the changed() state of both input and menu button widgets.

31.66.3.5 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.
31.66.3.6  menu() [1/2]

const Fl_Menu_Item * Fl_Input_Choice::menu() [inline]

Gets the Fl_Menu_Item array used for the menu.

31.66.3.7  menu() [2/2]

void Fl_Input_Choice::menu ( const Fl_Menu_Item * m ) [inline]

Sets the Fl_Menu_Item array used for the menu.

31.66.3.8  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)" );
}
```

31.66.3.9  update_menubutton()

int Fl_Input_Choice::update_menubutton()  

Updates the menubutton with the string value in Fl_Input.

If the string value currently in Fl_Input matches one of the menu items in menubutton(), that menu item will become the current item selected.

Call this method after setting value(const char *) if you need the menubutton() to be synchronized with the Fl_Input field.

```cpp
// Add items
choice->add(".25");
choice->add(".50");
choice->add("1.0");
choice->add("2.0");
choice->add("4.0");
choice->value("1.0");
choice->update_menubutton();  // cause menubutton to reflect this value too
// (returns 1 if match was found, 0 if not)

// Verify menubutton()'s value.
printf("menu button choice index=%d, value=%s\n", choice->menubutton()->value(), // would be -1 if update not done 
       choice->menubutton()->text());  // would be NULL if update not done
```
Returns

1 if a matching menuitem was found and value set, 0 if not.

Version

1.4.0

31.66.3.10 value() [1/2]

```c
void Fl_Input_Choice::value (  
    const char * val ) [inline]
```

Sets the Fl_Input text field's contents to val.

Note it is possible to set the value() to one that is not in the menubutton's list of choices.

Setting the value() does NOT affect the menubutton's selection. If that's needed, call update_menubutton() after setting value().

See also

    void value(int val), update_menubutton()

31.66.3.11 value() [2/2]

```c
void Fl_Input_Choice::value (  
    int val )
```

Chooses item# val in the menu, and sets the Fl_Input text field to that value.

Any previous text is cleared.

See also

    void value(const char *val)

The documentation for this class was generated from the following files:

- Fl_Input_Choice.H
- Fl_Input_Choice.cxx
31.67 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_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.

Additional Inherited Members

31.67.1 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 0xaef).

31.67.2 Constructor & Destructor Documentation

31.67.2.1 Fl_Int_Input()

Fl_Int_Input::Fl_Int_Input (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * l = 0 )

Creates a new Fl_Int_Input widget using the given position, size, and label string.

The default boxtype is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

The documentation for this class was generated from the following files:

- Fl_Int_Input.H
- Fl_Input.cxx
31.68 Fl_JPEG_Image Class Reference

The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.

```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.

Protected Member Functions

- `void load_jpg_(const char *filename, const char *sharename, const unsigned char *data)`

Additional Inherited Members

31.68.1 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.

31.68.2 Constructor & Destructor Documentation

31.68.2.1 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. |

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char *filename)

31.68.2 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.

Parameters

| name | A unique name or NULL |
| data | A pointer to the memory location of the JPEG image |

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char *filename)
Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_JPEG_Image.H
- Fl_JPEG_Image.cxx

31.69 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

31.69.1 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.

31.69.2 Member Function Documentation

31.69.2.1 draw()

```cpp
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.
31.69.2.2 measure()

```cpp
void Fl_Label::measure ( 
    int & W, 
    int & H ) const
```

Measures the size of the label.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W,H</td>
<td>this is the requested size for the label text plus image; on return, this will contain the size needed to fit the label</td>
</tr>
</tbody>
</table>

31.69.3 Member Data Documentation

31.69.3.1 type

```cpp
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`

31.70 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.

Protected Member Functions

- **virtual void draw** ()

  Draws the widget.

Additional Inherited Members

31.70.1 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()*.

![Fl_Light_Button](image)

Figure 31.21 Fl_Light_Button

31.70.2 Constructor & Destructor Documentation

31.70.2.1 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.
31.70.3 Member Function Documentation

31.70.3.1 draw()

```cpp
void Fl_Light_Button::draw () [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Button`.

31.70.3.2 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**

| in | event | the kind of event received |

**Return values**

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |
See also

Fl_Event

Reimplemented from Fl_Button.

The documentation for this class was generated from the following files:

- Fl_Light_Button.H
- Fl_Light_Button.cxx

### 31.71 Fl_Line_Dial Class Reference

Inheritance diagram for Fl_Line_Dial:

```
+-------------+                   +---------------+
| Fl_Widget   |                   | Fl_Valuator   |
|             |                   +---------------+
|             +-----------------+               |
| Fl_Dial     |                   | Fl_Line_Dial  |
|             |                   +---------------+
```

Public Member Functions

- **Fl_Line_Dial** (int X, int Y, int W, int H, const char *L=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Line_Dial.H
- Fl_Dial.cxx

### 31.72 Fl_Mac_App_Menu Class Reference

Static Public Member Functions

- **static void custom_application_menu_items** (const Fl_Menu_Item *m)

  Adds custom menu item(s) to the application menu of the system menu bar.
Static Public Attributes

- static const char * about
  Localizable text for the "About xxx" application menu item.
- static const char * hide
  Localizable text for the "Hide xxx" application menu item.
- static const char * hide_others
  Localizable text for the "Hide Others" application menu item.
- static const char * print
  Localizable text for the "Print Front Window" application menu item.
- static const char * quit
  Localizable text for the "Quit xxx" application menu item.
- static const char * services
  Localizable text for the "Services" application menu item.
- static const char * show
  Localizable text for the "Show All" application menu item.

31.72.1 Member Function Documentation

31.72.1.1 custom_application_menu_items()

static void Fl_Mac_App_Menu::custom_application_menu_items (const Fl_Menu_Item * m) [static]

Adds custom menu item(s) to the application menu of the system menu bar.

They are positioned after the "Print Front Window" 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.

31.72.2 Member Data Documentation

31.72.2.1 print

const char* Fl_Mac_App_Menu::print [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 file:

- mac.H
31.73 Fl_Menu_ Class Reference

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

```c
#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 *, 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)
- 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.
- 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 *)
  Find the menu item index for a given menu pathname, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const
  Find the index into the menu array for a given item.
- int find_index (Fl_Callback *cb) const
  Find the index into the menu array for a given callback cb.
- const Fl_Menu_Item * find_item (const char *name) const
  Find the menu item for a given menu pathname, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

Generated by Doxygen
Find the menu item for the given callback \texttt{cb}.

- \texttt{Fl_Menu_\_} (int, int, int, int, const char \texttt{*}=0)
  
  Creates a new \texttt{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 \texttt{*}, int shortcut, \texttt{Fl_Callback \*}, void \texttt{*}=0, int=0)
  
  Inserts a new menu item at the specified index position.

- int insert (int index, const char a, const char b, \texttt{Fl_Callback \*c}, void d=0, int e=0)
  
  See int \texttt{Fl_Menu::insert(const char\* label, int shortcut, Fl_Callback\*, void \*user_data=0, int flags=0)}

- int item_pathname (char \texttt{*}name, int namelen, const Fl_Menu_Item \texttt{*}finditem=0) const
  
  Get the menu 'pathname' for the specified menuitem.

- const Fl_Menu_Item \texttt{*}menu () const
  
  Returns a pointer to the array of Fl_Menu_Items.

- void menu (const Fl_Menu_Item \texttt{*}m)
  
  Sets the menu array pointer directly.

- const Fl_Menu_Item \texttt{*}menu_end ()
  
  Finishes menu modifications and returns \texttt{menu()}.

- void mode (int i, int fl)
  
  Sets the flags of item \texttt{i}.

- int mode (int i) const
  
  Gets the flags of item \texttt{i}.

- const Fl_Menu_Item \texttt{*}mvalue () const
  
  Returns a pointer to the last menu item that was picked.

- const Fl_Menu_Item \texttt{*}picked (const Fl_Menu_Item \texttt{*})
  
  When user picks a menu item, call this.

- void remove (int)
  
  Deletes item \texttt{i} from the menu.

- void replace (int, const char \texttt{*})
  
  Changes the text of item \texttt{i}.

- void setonly (Fl_Menu_Item \texttt{*}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 \texttt{i} to \texttt{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 \texttt{*}testShortcut ()
  
  Returns the menu item with the entered shortcut (key value).

- const char \texttt{*}text () const
  
  Returns the title of the last item chosen.

- const char \texttt{*}text (int i) const
  
  Returns the title of item \texttt{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.

Protected Member Functions

- int **item_pathname**_ (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Attributes

- uchar **alloc**
- uchar **down_box**_
- Fl_Color **textcolor**_
- Fl_Font **textfont**_
- Fl_Fontsize **textsize**_

Additional Inherited Members

31.73.1 Detailed Description

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

Currently FLTK provides you with Fl_Menu_Button, Fl_Menu_Bar, and Fl_Choice.

The class contains a pointer to an array of structures of type Fl_Menu_Item. The array may either be supplied directly by the user program, or it may be "private": a dynamically allocated array managed by the Fl_Menu_.

When the user clicks a menu item, value() is set to that item and then:

- If the Fl_Menu_Item has a callback set, that callback is invoked with any userdata configured for it. (The Fl_Menu_ widget's callback is NOT invoked.)
- For any Fl_Menu_Items that don't have a callback set, the Fl_Menu_ widget's callback is invoked with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item←_pathname(), etc.

The line spacing between menu items can be controlled with the global setting Fl::menu_linespacing().

See also

Fl_Widget::shortcut_label(int)
### 31.73.2 Constructor & Destructor Documentation

#### 31.73.2.1 Fl_Menu_()

```cpp
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.

### 31.73.3 Member Function Documentation

#### 31.73.3.1 add() [1/2]

```cpp
int Fl_Menu_::add (  
  const char * label,  
  int shortcut,  
  Fl_Callback * callback,  
  void * userdata = 0,  
  int flags = 0 )
```

Adds a new menu item.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>label</code></td>
<td>The text label for the menu item.</td>
</tr>
<tr>
<td><code>shortcut</code></td>
<td>Optional keyboard shortcut that can be an int or string: (FL_CTRL+'a') or &quot;^a&quot;. Default 0 if none.</td>
</tr>
<tr>
<td><code>callback</code></td>
<td>Optional callback invoked when user clicks the item. Default 0 if none.</td>
</tr>
<tr>
<td><code>userdata</code></td>
<td>Optional user data passed as an argument to the callback. Default 0 if none.</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>Optional flags that control the type of menu item; see below. Default is 0 for none.</td>
</tr>
</tbody>
</table>

**Returns**

The index into the `menu()` array, where the entry was added.

**Description**

If the menu array was directly set with `menu(x)`, then `copy()` is done to make a private array.

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.
A menu item's callback must not add() items to its parent menu during the callback.

**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.

Generated by Doxygen
# - Alt
+ - Shift
^ - Control

Internally, the text shortcuts are converted to integer values using \texttt{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:

- \texttt{FL\_MENU\_INACTIVE} // Deactivate menu item (gray out)
- \texttt{FL\_MENU\_TOGGLE} // Item is a checkbox toggle (shows checkbox for on/off state)
- \texttt{FL\_MENU\_VALUE} // The on/off state for checkbox/radio buttons (if set, state is 'on')
- \texttt{FL\_MENU\_RADIO} // Item is a radio button (one checkbox of many can be on)
- \texttt{FL\_MENU\_INVISIBLE} // Item will not show up (shortcut will work)
- \texttt{FL\_SUBMENU\_POINTER} // Indicates user\_data() is a pointer to another menu array
- \texttt{FL\_SUBMENU} // This item is a submenu to other items
- \texttt{FL\_MENU\_DIVIDER} // Creates divider line below this item. Also ends a group of radio buttons.

If \texttt{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 \url{http://fltk.org/newsgroups.php?g=fltk.development+v:10086} and results of STR#2344

31.7.3.2 \texttt{add()} [2/2]

\begin{verbatim}
int Fl_Menu_::add (const char * str )
\end{verbatim}

This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and \texttt{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 \texttt{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

31.73.3 clear()

void Fl_Menu_::clear ( )

Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

Menus must not be cleared during a callback to the same menu.

31.73.3.4 clear_submenu()

int Fl_Menu_::clear_submenu ( int index )

Clears the specified submenu pointed to by index of all menu items.

This method is useful for clearing a submenu so that it can be re-populated with new items. Example: a “File/Recent Files/…” submenu that shows the last few files that have been opened.

The specified index must point to a submenu.

The submenu is cleared with remove(). If the menu array was directly set with menu(x), then copy() is done to make a private array.

Warning

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

Example:

```
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)

31.73.3.5 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).

31.73.3.6 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.

31.73.3.7 find_index() [1/3]

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.
31.73.3.8 find_index() [2/3]

```cpp
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()`

31.73.3.9 find_index() [3/3]

```cpp
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

\texttt{cb}  Find the first item with this callback

Returns

The index of the item with the specific callback, or -1 if not found

See also

\texttt{find\_index(const\ char*)}

31.73.3.10  \texttt{find\_item()} [1/2]

\texttt{const Fl\_Menu\_Item * Fl\_Menu\_::find\_item (}
\texttt{        const char * pathway )}

Find the menu item for a given menu \texttt{pathname}, such as "Edit/Copy".

This method finds a menu item in the menu array, also traversing submenus, but not submenu pointers (FL\_SUB~\texttt{MENU\_POINTER}).

To get the menu item's index, use \texttt{find\_index(const\ char*)}

Example:

```cpp
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

\texttt{pathname}  The path and name of the menu item

Returns

The item found, or NULL if not found

See also

\texttt{find\_index(const\ char*)}, \texttt{find\_item(Fl\_Callback*)}, \texttt{item\_pathname()}
31.73.3.11  find_item() [2/2]

```cpp
const Fl_Menu_Item * Fl_Menu_::find_item (  
    Fl_Callback * cb )
```

Find the menu item for the given callback `cb`.

This method finds a menu item in a menu array, also traversing submenus, but not submenu pointers. This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.

**Parameters**

| in  | cb  | find the first item with this callback |

**Returns**

The item found, or NULL if not found

**See also**

`find_item(const char*)`

31.73.3.12  global()

```cpp
void Fl_Menu_::global ( )
```

Make the shortcuts for this menu work no matter what window has the focus when you type it.

This is done by using `Fl::add_handler()`. This `Fl_Menu_` widget does not have to be visible (ie the window it is in can be hidden, or it does not have to be put in a window at all).

Currently there can be only one `global()` menu. Setting a new one will replace the old one. There is no way to remove the `global()` setting (so don't destroy the widget!)

31.73.3.13  insert()

```cpp
int Fl_Menu_::insert (  
    int index,  
    const char * label,  
    int shortcut,  
    Fl_Callback * callback,  
    void * userdata = 0,  
    int flags = 0 )
```

Inserts a new menu item at the specified `index` position.

If `index` is -1, the menu item is appended; same behavior as `add()`.

To properly insert a menu item, `label` must be the name of the item (eg. "Quit"), and not a 'menu pathname' (eg. "File/Quit"). If a menu pathname is specified, the value of `index` is ignored, the new item's position defined by the pathname.

For more details, see `add()`. Except for the `index` parameter, `add()` has more detailed information on parameters and behavior, and is functionally equivalent.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>The menu array's index position where the new item is inserted. If -1, behavior is the same as <code>add()</code>.</td>
</tr>
<tr>
<td><code>label</code></td>
<td>The text label for the menu item. If the label is a menu pathname, <code>index</code> is ignored, and the pathname indicates the position of the new item.</td>
</tr>
<tr>
<td><code>shortcut</code></td>
<td>Optional keyboard shortcut. Can be an int (FL_CTRL+'a') or a string (&quot;^a&quot;). Default is 0.</td>
</tr>
<tr>
<td><code>callback</code></td>
<td>Optional callback invoked when user clicks the item. Default 0 if none.</td>
</tr>
<tr>
<td><code>userdata</code></td>
<td>Optional user data passed as an argument to the callback. Default 0 if none.</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>Optional flags that control the type of menu item; see <code>add()</code> for more info. Default is 0 for none.</td>
</tr>
</tbody>
</table>

### Returns

The index into the `menu()` array, where the entry was added.

### See also

- `add()`

### 31.73.3.14 item_pathname()

```cpp
int Fl_Menu_::item_pathname (char ∗name, int namelen, const Fl_Menu_Item ∗finditem = 0) const
```

Get the menu 'pathname' for the specified menuitem.

If `finditem` is NULL, `mvalue()` is used (the most recently picked menuitem).

### Example:

```cpp
Fl_Menu_Bar ∗menubar = 0;
void my_menu_callback(Fl_Widget*, void*) {  
    char name[80];
    if (menubar->item_pathname(name, sizeof(name)-1) == 0) { // recently picked item
        if (strcmp(name, "File/&Open") == 0) { .. } // open invoked
        if (strcmp(name, "File/&Save") == 0) { .. } // save invoked
        if (strcmp(name, "Edit/&Copy") == 0) { .. } // copy invoked
    }
}
int main() {
    ..
    menubar = new Fl_Menu_Bar(..);
    menubar->add("File/&Open", 0, my_menu_callback);
    menubar->add("File/&Save", 0, my_menu_callback);
    menubar->add("Edit/&Copy", 0, my_menu_callback);
    ..
}
```

### Returns

- 0 : OK (name has menuitem's pathname)
- -1 : item not found (name="")
- -2 : 'name' not large enough (name="")

### See also

- `find_item()`
31.73 Fl_Menu_ Class Reference

31.73.3.15 menu() [1/2]

const Fl_Menu_Item* Fl_Menu_::menu () const [inline]

Returns a pointer to the array of Fl_Menu_Items.

This will either be the value passed to menu(value) or the private copy or an internal (temporary) location (see note below).

Note

Implementation details - may be changed in the future. All modifications of the menu array are done by copying the entire menu array to an internal storage for optimization of memory allocations, for instance when using add() or insert(). While this is done, menu() returns the pointer to this internal location. The entire menu will be copied back to private storage when needed, i.e. when another Fl_Menu_ is modified. You can force this reallocation after you’re done with all menu modifications by calling Fl_Menu_::menu_end() to make sure menu() returns a permanent pointer to private storage (until the menu is modified again). Note also that some menu methods (e.g. Fl_Menu_Button::popup()) call menu_end() internally to ensure a consistent menu array while the menu is open.

See also

size() – returns the size of the Fl_Menu_Item array.
menu_end() – finish menu modifications (optional)

Example: How to walk the array:

```c
for ( int t=0; t<menubar->size(); t++ ) { // walk array of items
    const Fl_Menu_Item item = menubar->menu()[t]; // get each item
    fprintf(stderr, "item #%d -- label=%s, value=%s type=%s\n",
        t,
        item.label() ? item.label() : "(Null)", // menu terminators have NULL labels
        (item.flags & FL_MENU_VALUE) ? "set" : "clear", // value of toggle or radio items
        (item.flags & FL_SUBMENU) ? "Submenu" : "Item"); // see if item is a submenu or actual item
}
```

31.73.3.16 menu() [2/2]

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.
31.73.3.17  menu_end()

const Fl_Menu_Item * Fl_Menu_::menu_end ()

Finishes menu modifications and returns menu().

Call menu_end() after using add(), insert(), remove(), or any other methods that may change the menu array if you
want to access the menu array anytime later with menu(). This should be called only once after the last menu
modification for performance reasons.

Does nothing if the menu array is already in a private location.

Some methods like Fl_Menu_Button::popup() call this method before their menu is opened.

Note

After menu changes like add(), insert(), etc. menu() would return a pointer to a temporary internal menu array
that may be relocated at unexpected times. This is due to performance considerations and may be changed
w/o further notice.

Since

1.4.0

Returns

New Fl_Menu_Item array pointer.

See also

Fl_Menu_::menu()

31.73.3.18  mode() [1/2]

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.

31.73.3.19  mode() [2/2]

int Fl_Menu_::mode (  
    int i ) const [inline]

Gets the flags of item i.

For a list of the flags, see Fl_Menu_Item.
31.73.3.20 mvalue()

```cpp
const Fl_Menu_Item * Fl_Menu_::mvalue ( ) const [inline]
```

Returns a pointer to the last menu item that was picked.

31.73.3.21 picked()

```cpp
const Fl_Menu_Item * Fl_Menu_::picked ( const Fl_Menu_Item * v )
```

When user picks a menu item, call this.

It will do the callback. Unfortunately this also casts away const for the checkboxes, but this was necessary so non-checkbox menus can really be declared const...

31.73.3.22 remove()

```cpp
void Fl_Menu_::remove ( int i )
```

Deletes item \texttt{i} from the menu.

If the menu array was directly set with menu(x) then \texttt{copy()} is done to make a private array.

No items must be removed from a menu during a callback to the same menu.

Parameters

| \texttt{i} | index into menu array |

31.73.3.23 replace()

```cpp
void Fl_Menu_::replace ( int i, const char * str )
```

Changes the text of item \texttt{i}.

This is the only way to get slash into an \texttt{add()}ed menu item. If the menu array was directly set with menu(x) then \texttt{copy()} is done to make a private array.

Parameters

| \texttt{i} | index into menu array |
| \texttt{str} | new label for menu item at index i |
31.73.3.24 setonly()

```cpp
void Fl_Menu_::setonly (Fl_Menu_Item * item)
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

31.73.3.25 shortcut()

```cpp
void Fl_Menu_::shortcut (int i, int s) [inline]
```

Changes the shortcut of item \(i\) to \(s\).

31.73.3.26 size()

```cpp
int Fl_Menu_::size () const
```

This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.

This includes the "terminator" item at the end. To copy a menu array you need to copy `size()`\(\times\)sizeof(Fl_Menu_Item) bytes. If the menu is NULL this returns zero (an empty menu will return 1).

31.73.3.27 test_shortcut()

```cpp
const Fl_Menu_Item* Fl_Menu_::test_shortcut () [inline]
```

Returns the menu item with the entered shortcut (key value).

This searches the complete `menu()` for a shortcut that matches the entered key value. It must be called for a FL_KEYBOARD or FL_SHORTCUT event.

If a match is found, the menu's callback will be called.

Returns

matched Fl_Menu_Item or NULL.
31.73.3.28 text() [1/2]

const char* Fl_Menu_::text ( ) const  [inline]

Returns the title of the last item chosen.

31.73.3.29 text() [2/2]

const char* Fl_Menu_::text ( int i ) const  [inline]

Returns the title of item i.

31.73.3.30 textcolor() [1/2]

Fl_Color Fl_Menu_::textcolor ( ) const  [inline]

Get the current color of menu item labels.

31.73.3.31 textcolor() [2/2]

void Fl_Menu_::textcolor ( Fl_Color c )  [inline]

Sets the current color of menu item labels.

31.73.3.32 textfont() [1/2]

Fl_Font Fl_Menu_::textfont ( ) const  [inline]

Gets the current font of menu item labels.

31.73.3.33 textfont() [2/2]

void Fl_Menu_::textfont ( Fl_Font c )  [inline]

Sets the current font of menu item labels.
31.73.3.34 textsize() [1/2]

Fl_Fontsize Fl_Menu_::textsize ( ) const [inline]

Gets the font size of menu item labels.

31.73.3.35 textsize() [2/2]

void Fl_Menu_::textsize ( Fl_Fontsize c ) [inline]

Sets the font size of menu item labels.

31.73.3.36 value() [1/3]

int Fl_Menu_::value ( ) const [inline]

Returns the index into menu() of the last item chosen by the user.

It is zero initially.

31.73.3.37 value() [2/3]

int Fl_Menu_::value ( const Fl_Menu_Item * m )

The value is the index into menu() of the last item chosen by the user.

It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero if the new value is different than the old one.

31.73.3.38 value() [3/3]

int Fl_Menu_::value ( int i ) [inline]

The value is the index into menu() of the last item chosen by the user.

It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero if the new value is different than the old one.

The documentation for this class was generated from the following files:

- Fl_Menu_.H
- Fl_Menu_.cxx
- Fl_Menu_add.cxx
- Fl_Menu_global.cxx
31.74 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

#include <Fl_Menu_Bar.H>

Inheritance diagram for Fl_Menu_Bar:

```
Fl_Widget
   Fl_Menu_
      Fl_Menu_Bar
         Fl_Sys_Menu_Bar
```

Public Member Functions

- Fl_Menu_Bar (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.

- int handle (int)
  
  Handles the specified event.

- virtual void update ()
  
  Updates the menu bar after any change to its items.

Protected Member Functions

- void draw ()
  
  Draws the widget.

Friends

- class Fl_Sys_Menu_Bar_Driver

Additional Inherited Members

31.74.1 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.
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.

### 31.74.2 Constructor & Destructor Documentation

#### 31.74.2.1 Fl_Menu_Bar()

```cpp
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.

`labelsizes()`, `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.
31.74.3 Member Function Documentation

31.74.3.1 draw()

```cpp
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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

Reimplemented in `Fl_Sys_Menu_Bar`.

31.74.3.2 handle()

```cpp
int Fl_Menu_Bar::handle(int event) [virtual]
```

Handles the specified event.

You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.

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>

Generated by Doxygen
See also

Fl_Event

Reimplemented from Fl_Widget.

31.74.3.3 update()

virtual void Fl_Menu_Bar::update ( ) [inline], [virtual]

Updates the menu bar after any change to its items.

This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.

Reimplemented in Fl_Sys_Menu_Bar.

The documentation for this class was generated from the following files:

- Fl_Menu_Bar.H
- Fl_Menu_Bar.cxx

31.75 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:

```
void 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.

Protected Member Functions

- **void draw ()**
  Draws the widget.

Additional Inherited Members

31.75.1 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.

![menu_button](image)

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.

Generated by Doxygen
31.75.2 Member Enumeration Documentation

31.75.2.1 popup_buttons

```
enum 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.

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPUP1</td>
<td>pops up with the mouse 1st button.</td>
</tr>
<tr>
<td>POPUP2</td>
<td>pops up with the mouse 2nd button.</td>
</tr>
<tr>
<td>POPUP12</td>
<td>pops up with the mouse 1st or 2nd buttons.</td>
</tr>
<tr>
<td>POPUP3</td>
<td>pops up with the mouse 3rd button.</td>
</tr>
<tr>
<td>POPUP13</td>
<td>pops up with the mouse 1st or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP23</td>
<td>pops up with the mouse 2nd or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP123</td>
<td>pops up with any mouse button.</td>
</tr>
</tbody>
</table>

31.75.3 Constructor & Destructor Documentation

31.75.3.1 Fl_Menu_Button()

```
Fl_Menu_Button::Fl_Menu_Button ( 
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0 )
```

Creates a new `Fl_Menu_Button` widget using the given position, size, and label string.

The default boxtype is FL_UP_BOX.

The constructor sets `menu()` to NULL. See `Fl_Menu` for the methods to set or change the menu.

31.75.4 Member Function Documentation
31.75.4.1  draw()

void Fl_Menu_Button::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```c
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

31.75.4.2  handle()

int Fl_Menu_Button::handle ( int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

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.
### 31.75.4.3 `popup()`

```cpp
const Fl_Menu_Item * Fl_Menu_Button::popup ()
```

Act exactly as though the user clicked the button or typed the shortcut key.

The menu appears, it waits for the user to pick an item, and if they pick one it sets `value()` and does the callback or sets `changed()` as described above. The menu item is returned or NULL if the user dismisses the menu.

**Note**

Since FLTK 1.4.0 `Fl_Menu_::menu_end()` is called before the menu pops up to make sure the menu array is located in private storage.

**See also**

`Fl_Menu_::menu_end()`

The documentation for this class was generated from the following files:

- `Fl_Menu_Button.H`
- `Fl_Menu_Button.cxx`

### 31.76 `Fl_Menu_Item` Struct Reference

The `Fl_Menu_Item` structure defines a single menu item that is used by the `Fl_Menu_` class.

```cpp
#include <Fl_Menu_Item.H>
```

**Public Member Functions**

- `void activate ()`
  
  Allows a menu item to be picked.
- `int active () const`
  
  Gets whether or not the item can be picked.
- `int activevisible () const`
  
  Returns non 0 if `FL_INACTIVE` and `FL_INVISIBLE` are cleared, 0 otherwise.
- `int add (const char *, int shortcut, Fl_Callback *, void *, int=0)`
  
  Adds a menu item.
- `int add (const char *, const char *, Fl_Callback *, void *, int)`
  
  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, void *p)`
  
  Sets the menu item's callback function and `userdata()` argument.
• void callback (Fl_Callback *c)
  Sets the menu item's callback function.

• 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, void *arg) 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 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.

• const Fl_Menu_Item *first () const
  Returns the first menu item, same as next(0).

• Fl_Menu_Item *first ()
  Returns the first menu item, same as next(0).

• void hide ()
  Hides an item in the menu.

• void image (Fl_Image *image)
  Compatibility API for FLUID, same as image->label(this).

• void image (Fl_Image &image)
  Compatibility API for FLUID, same as image.label(this).

• 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.

- **const Fl_Menu_Item * next (int i=1) const**
  
  Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

- **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 * popup (int X, int Y, const char *title=0, const Fl_Menu_Item *picked=0, const Fl_Menu_*=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_*=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 widget's 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 labeltype_
  how the menu item text looks like

- int shortcut_
  menu item shortcut

- const char * text
  menu item text, returned by `label()`

- void * user_data_
  menu item user_data for the menu's callback

31.76.1 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_;  // Deactivate menu item (gray out)
    Fl_Callback* callback_;  // Item is a checkbox toggle (shows checkbox for on/off state)
    void* user_data_;  // The on/off state for checkbox/radio buttons (if set, state is 'on')
    int flags;  // Item is a radio button (one checkbox of many can be on)
    uchar labeltype_;  // Indicates user_data() is a pointer to another menu array
    uchar labelfont_;  // This item is a submenu to other items
    uchar labelsize_;  // Creates divider line below this item. Also ends a group of radio buttons.
    uchar labelcolor_;  // ??? -- reserved
};
enum { // values for flags:
    FL_MENU_INACTIVE = 1,  // Deactivate menu item (gray out)
    FL_MENU_TOGGLE = 2,    // Item is a checkbox toggle (shows checkbox for on/off state)
    FL_MENU_VALUE = 4,     // Item will not show up (shortcut will work)
    FL_MENU_RADIO = 8,     // Item is a radio button (one checkbox of many can be on)
    FL_SUBMENU_POINTER = 0x20,  // Indicates user_data() is a pointer to another menu array
    FL_SUBMENU = 0x40,     // This item is a submenu to other items
    FL_MENU_DIVIDER = 0x80, // Creates divider line below this item. Also ends a group of radio buttons.
    FL_MENU_HORIZONTAL = 0x100  // ??? -- reserved
};
```

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_VALUE},
    {"strange", 0, strange_cb},
    {"charm", 0, charm_cb},
    {"truth", 0, truth_cb},
    {"beauty", 0, beauty_cb},
    {"submenu", 0, 0, 0, FL_SUBMENU},
    {"one"},
    {"two"},

    Generated by Doxygen
```
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.

31.76.2 Member Function Documentation

31.76.2.1 activate()

void Fl_Menu_Item::activate () [inline]

Allows a menu item to be picked.
active()

```cpp
int Fl_Menu_Item::active ( ) const [inline]
```

Gets whether or not the item can be picked.

activevisible()

```cpp
int Fl_Menu_Item::activevisible ( ) const [inline]
```

Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.

add()

```cpp
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)

argument() [1/2]

```cpp
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.
31.76.2.6 argument() [2/2]

```cpp
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.

31.76.2.7 callback() [1/5]

```cpp
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_Item*)w)->mvalue()` where `w` is the widget argument.

31.76.2.8 callback() [2/5]

```cpp
void Fl_Menu_Item::callback (Fl_Callback * c, void * p) [inline]
```

Sets the menu item's callback function and `userdata()` argument.

See also

```cpp
Fl_Callback_p Fl_MenuItem::callback() const
```

31.76.2.9 callback() [3/5]

```cpp
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

```cpp
Fl_Callback_p Fl_MenuItem::callback() const
```
### 31.76.2.10 callback [4/5]

```cpp
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`

### 31.76.2.11 callback [5/5]

```cpp
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`

### 31.76.2.12 check()

```cpp
void Fl_Menu_Item::check () [inline]
```

back compatibility only

**Deprecated**

### 31.76.2.13 checkbox()

```cpp
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.
31.76.2.14 checked()

```cpp
int Fl_Menu_Item::checked ( ) const [inline]
```

back compatibility only

**Deprecated**

31.76.2.15 clear()

```cpp
void Fl_Menu_Item::clear ( ) [inline]
```

Turns the check or radio item "off" for the menu item.

31.76.2.16 deactivate()

```cpp
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.

31.76.2.17 do_callback() [1/3]

```cpp
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.

31.76.2.18 do_callback() [2/3]

```cpp
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.
31.76.2.19  do_callback() [3/3]

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.

31.76.2.20  draw()

void Fl_Menu_Item::draw ( 
    int x, 
    int y, 
    int w, 
    int h, 
    const Fl_Menu_ * m, 
    int selected = 0 ) const

Draws the menu item in bounding box x,y,w,h, optionally selects the item.

31.76.2.21  find_shortcut()

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>returns the index of the item, if ip is not NULL.</td>
</tr>
<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

31.76.2.22  first() [1/2]

const Fl_Menu_Item* Fl_Menu_Item::first ( ) const [inline]
Returns the first menu item, same as next(0).

31.76.2.23  first() [2/2]

Fl_Menu_Item* Fl_Menu_Item::first ( ) [inline]
Returns the first menu item, same as next(0).

31.76.2.24  hide()

void Fl_Menu_Item::hide ( ) [inline]
Hides an item in the menu.

31.76.2.25  image() [1/2]

void Fl_Menu_Item::image (  
   Fl_Image * image ) [inline]
Compatibility API for FLUID, same as image->label(this).

Note
This method is intended for internal use by fluid and may not do what you expect.

31.76.2.26  image() [2/2]

void Fl_Menu_Item::image (  
   Fl_Image & image ) [inline]
Compatibility API for FLUID, same as image.label(this).

Note
This method is intended for internal use by fluid and may not do what you expect.

31.76.2.27  insert()

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

| in | index     | insert new items here |
| in | mytext    | new label string, details see above |
| in | sc        | keyboard shortcut for new item |
| in | cb        | callback function for new item |
| in | data      | user data for new item |
| in | myflags   | menu flags as described in FL_Menu_Item |

Returns

the index into the menu() array, where the entry was added

31.76.2.28 label()

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.

31.76.2.29 labelcolor()[1/2]

Fl_Color Fl_Menu_Item::labelcolor() const [inline]

Gets the menu item’s label color.

This color is passed to the labelfont 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.

31.76.2.30 labelcolor()[2/2]

void Fl_Menu_Item::labelcolor(
    Fl_Color a) [inline]

Sets the menu item’s label color.

See also

    Fl_Color Fl_Menu_Item::labelcolor() const

Generated by Doxygen
31.76.2.31 labelfont() [1/2]

**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.

31.76.2.32 labelfont() [2/2]

void Fl_Menu_Item::labelfont ( 
    **Fl_Font** a ) [inline]

Sets the menu item's label font.

Fonts are identified by small 8-bit indexes into a table. See the enumeration list for predefined fonts. The default value is a Helvetica font. The function **Fl::set_font()** can define new fonts.

31.76.2.33 labelsize() [1/2]

**Fl_Fontsize** Fl_Menu_Item::labelsize ( ) const [inline]

Gets the label font pixel size/height.

31.76.2.34 labelsize() [2/2]

void Fl_Menu_Item::labelsize ( 
    **Fl_Fontsize** a ) [inline]

Sets the label font pixel size/height.

31.76.2.35 labeltype() [1/2]

**Fl_Labeltype** Fl_Menu_Item::labeltype ( ) const [inline]

Returns the menu item's labeltype.

A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the **label()** pointer as another form of data such as a bitmap. The value **FL_NORMAL_LABEL** prints the label as text.
31.76.2.36 labeltype() [2/2]

void Fl_Menu_Item::labeltype ( Fl_Labeltype a ) [inline]

Sets the menu item’s labeltype.

A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the label() pointer as another form of data such as a bitmap. The value FL_NORMAL_LABEL prints the label as text.

31.76.2.37 measure()

int Fl_Menu_Item::measure ( int * hp, const Fl_Menu_ * m ) const

Measures width of label, including effect of & characters.

Optionally, can get height if hp is not NULL.

31.76.2.38 next() [1/2]

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.

31.76.2.39 next() [2/2]

Fl_Menu_Item* Fl_Menu_Item::next ( int i = 1 ) [inline]

Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

There are two calls so that you can advance through const and non-const data.

31.76.2.40 popup()

const Fl_Menu_Item * Fl_Menu_Item::popup ( int X, int Y, const char * title = 0, const Fl_Menu_Item * picked = 0, const Fl_Menu_ * menu_button = 0 ) const

This method is called by widgets that want to display menus.

The menu stays up until the user picks an item or dismisses it. The selected item (or NULL if none) is returned. This does not do the callbacks or change the state of check or radio items.

The menu is positioned so the cursor is centered over the item picked. This will work even if picked is in a submenu. If picked is zero or not in the menu item table the menu is positioned with the cursor in the top-left corner.
Parameters

| in   | X,Y                                      | the position of the mouse cursor, relative to the window that got the most recent event (usually you can pass Fl::event_x() and Fl::event_y() unchanged here). |
| in   | title                                    | a character string title for the menu. If non-zero a small box appears above the menu with the title in it. |
| in   | picked                                   | if this pointer is not NULL, the popup menu will appear so that the picked menu is under the mouse pointer. |
| in   | menu_button                              | is a pointer to an Fl_Menu_ from which the color and boxtypes for the menu are pulled. If NULL then defaults are used. |

Returns

a pointer to the menu item selected by the user, or NULL

31.76.2.41 pulldown()

const Fl_Menu_Item * Fl_Menu_Item::pulldown (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const Fl_Menu_Item * initial_item = 0,  
    const Fl_Menu_ * pbutton = 0,  
    const Fl_Menu_Item * title = 0,  
    int menubar = 0 ) const

Pulldown() is similar to popup(), but a rectangle is provided to position the menu. The menu is made at least W wide, and the picked item initial_item is centered over the rectangle (like Fl_Choice uses). If initial_item is NULL or not found, the menu is aligned just below the rectangle (like a pulldown menu). The title and menubar arguments are used internally by the Fl_Menu_Bar widget.

31.76.2.42 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.

31.76.2.43 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.
31.76.2.44 setonly()

```cpp
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_Item::setonly(Fl_Menu_Item*)` instead.

31.76.2.45 shortcut() [1/2]

```cpp
int Fl_Menu_Item::shortcut ( ) const [inline]
```

Gets what key combination shortcut will trigger the menu item.

31.76.2.46 shortcut() [2/2]

```cpp
void Fl_Menu_Item::shortcut ( int s ) [inline]
```

Sets exactly what key combination will trigger the menu item.

The value is a logical 'or' of a key and a set of shift flags, for instance FL_ALT+'a' or FL_ALT+FL_F+10 or just 'a'. A value of zero disables the shortcut.

The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.

The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

31.76.2.47 show()

```cpp
void Fl_Menu_Item::show ( ) [inline]
```

Makes an item visible in the menu.
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:

```c
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()\n\nsize() = %d\n\n", schemechoice->menu()->size());
```

Output:

```
schemechoice->menu()->size() = 5
```

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.

const Fl_Menu_Item * Fl_Menu_Item::testShortcut() 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 ’

void Fl_Menu_Item::uncheck() [inline]

back compatibility only

Deprecated.
31.76.2.52 value()

```cpp
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.

31.76.2.53 visible()

```cpp
int Fl_Menu_Item::visible ( ) const [inline]
```

Gets the visibility of an item.

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`

31.77 Fl_Menu_Window Class Reference

The `Fl_Menu_Window` widget is a window type used for menus.

```cpp
#include <Fl_Menu_Window.H>
```

Inheritance diagram for `Fl_Menu_Window`:

```
Fl_Widget
  └── Fl_Group
      └── Fl_Window
          └── Fl_Single_Window
              └── Fl_Menu_Window
```

Generated by Doxygen
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 in 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.

Additional Inherited Members

31.77.1 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.

31.77.2 Constructor & Destructor Documentation

31.77.2.1 ~Fl_Menu_Window()

Fl_Menu_Window::~Fl_Menu_Window ()

Destroys the window and all of its children.
31.77.2.2 Fl_Menu_Window() [1/2]

Fl_Menu_Window::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.

31.77.2.3 Fl_Menu_Window() [2/2]

Fl_Menu_Window::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.

31.77.3 Member Function Documentation

31.77.3.1 clear_overlay()

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.

31.77.3.2 flush()

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.
31.77.3.3  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.

31.77.3.4  set_overlay()

```cpp
void Fl_Menu_Window::set_overlay ( ) [inline]
```

Tells FLTK to use hardware overlay planes if they are available.

31.77.3.5  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
The **Fl Multi_Browser** class is a subclass of **Fl Browser** which lets the user select any set of the lines.

```cpp
#include <Fl Multi_Browser.H>
```

Inheritance diagram for Fl Multi_Browser:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Group</td>
</tr>
<tr>
<td>Fl_Browser_</td>
</tr>
<tr>
<td>Fl_Browser</td>
</tr>
<tr>
<td>Fl Multi_Browser</td>
</tr>
</tbody>
</table>
```

### 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.*

### Additional Inherited Members

#### 31.78.1 Detailed Description

The **Fl Multi_Browser** class is a subclass of **Fl Browser** which lets the user select any set of the lines.

```
<table>
<thead>
<tr>
<th>Aaa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bbb</td>
</tr>
<tr>
<td>Ccc</td>
</tr>
<tr>
<td>Ddd</td>
</tr>
</tbody>
</table>
```

*Figure 31.25 Fl Multi_Browser*

The user interface is Macintosh style: clicking an item turns off all the others and selects that one, dragging selects all the items the mouse moves over, and ctrl + click (Cmd+click on the Mac OS platform) toggles the items. Shift + click extends the selection until the clicked item. This is different from how forms did it. Normally the callback is done when the user releases the mouse, but you can change this with when().

See **Fl Browser** for methods to add and remove lines from the browser.

#### 31.78.2 Constructor & Destructor Documentation

Generated by Doxygen
31.78.2.1 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

31.79 Fl_Multi_Label Struct Reference

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.

#include <Fl_Multi_Label.H>

Public Member Functions

• void label (Fl_Widget ∗)
   This method is used to associate a Fl_Multi_Label with a Fl_Widget.
• void label (Fl_Menu_Item ∗)
   This method is used to associate a Fl_Multi_Label with a Fl_Menu_Item.

Public Attributes

• const char ∗ labela  
   Holds the "leftmost" of the two elements in the composite label.
• const char ∗ labelb  
   Holds the "rightmost" of the two elements in the composite label.
• uchar typea  
   Holds the "type" of labela.
• uchar typeb  
   Holds the "type" of labelb.
31.79.1 Detailed Description

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.

Most regular FLTK widgets now support the ability to associate both images and text with a label but some special cases, notably the non-widget Fl_Menu_Item objects, do not. Fl_Multi_Label may be used to create menu items that have an icon and text, which would not normally be possible for an Fl_Menu_Item. For example, Fl_Multi_Label is used in the New->Code submenu in fluid, and others.

![Figure 31.26 Menu items with icons using Fl_Multi_Label](image)

Each Fl_Multi_Label holds two elements, labela and labelb; each may hold either a text label (const char*) or an image (Fl_Image*). When displayed, labela is drawn first and labelb is drawn immediately to its right.

More complex labels 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.

Example Use: Fl_Menu_Bar

```cpp
Fl_Pixmap *image = new Fl_Pixmap(...); // image for menu item; any Fl_Image based widget
Fl_Menu_Bar *menu = new Fl_Menu_Bar(...); // can be any Fl_Menu_ oriented widget (Fl_Choice, Fl_Menu_Button...) // Create a menu item int i = menu->add("File/New", ...);
Fl_Menu_Item *item = (Fl_Menu_Item*)&(menu->menu()[i]);
// Create a multi label, assign it an image + text Fl_Multi_Label *ml = new Fl_Multi_Label;
// Left side of label is an image ml->typea = FL_IMAGE_LABEL;
ml->labela = (const char*)image; // any Fl_Image widget: Fl_Pixmap, Fl_PNG_Image, etc..
// Right side of label is label text ml->typeb = FL_NORMAL_LABEL;
ml->labelb = item->label();
// Assign the multilabel to the menu item ml->label(item);
```

See also

Fl_Label and Fl_Labeltype and examples/howto-menu-with-images.cxx

31.79.2 Member Function Documentation
31.79.2.1 label() [1/2]

```c
void Fl_Multi_Label::label ( Fl_Widget * o )
```

This method is used to associate a `Fl_Multi_Label` with a `Fl_Widget`.

31.79.2.2 label() [2/2]

```c
void Fl_Multi_Label::label ( Fl_Menu_Item * o )
```

This method is used to associate a `Fl_Multi_Label` with a `Fl_Menu_Item`.

31.79.3 Member Data Documentation

31.79.3.1 labela

```c
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*).

31.79.3.2 labelb

```c
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*).

31.79.3.3 typea

```c
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.
31.80 **Fl_Multiline_Input Class Reference**

This input field displays `\n` characters as new lines rather than `\r`, and accepts the Return, Tab, and up and down arrow keys.

```c
#include <Fl_Multiline_Input.H>
```

Inheritance diagram for Fl_Multiline_Input:

```
Inheritance diagram for Fl_Multiline_Input:
   Fl_Multiline_Input
       |           |
       v           v
   Fl_Input
       |           |
       v           v
   Fl_Input_
       |           |
       v           v
   Fl_Widget
```

### Public Member Functions

- `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.

### Additional Inherited Members

31.80.1 **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`.
31.80.2 Constructor & Destructor Documentation

31.80.2.1 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

31.81 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:

```
Fl_Widget
   \--------
   |        |
   |        |
Fl_Input
   \--------
   |        |
   |        |
Fl_Input_
   \--------
   |        |
   |        |
Fl_Output
   \--------
   |        |
   |        |
Fl_Multiline_Output
```

Public Member Functions

- **Fl_Multiline_Output** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Multiline_Output widget using the given position, size, and label string.
Additional Inherited Members

31.81.1 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.

31.81.2 Constructor & Destructor Documentation

31.81.2.1 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

31.82 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.

- **void directory (const char ∗val)**
  Preset the directory the browser will show when opened.

- **const char ∗ directory () const**
  Returns the current preset directory() value.

- **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.

- **void filter_value (int i)**
  Sets which filter will be initially selected.

- **int filter_value () const**
  Returns which filter value was last selected by the user.

- **int filters () const**
  Gets how many filters were available, not including "All Files".

- **Fl_Native_File_Chooser (int val=BROWSE_FILE)**
  The constructor.

- **void options (int o)**
  Sets the platform specific chooser options to val.

- **int options () const**
  Gets the platform specific Fl_Native_File_Chooser::Option flags.

- **void preset_file (const char ∗f)**
  Sets the default filename for the chooser.

- **const char ∗ preset_file () const**
  Get the preset filename.

- **int show ()**
  Post the chooser's dialog.

- **void title (const char ∗t)**
  Set the title of the file chooser's dialog window.

- **const char ∗ title () const**
  Get the title of the file chooser's dialog window.

- **void type (int t)**
  Sets the current Fl_Native_File_Chooser::Type of browser.

- **int type () const**
  Gets the current Fl_Native_File_Chooser::Type of browser.

- **∼Fl_Native_File_Chooser ()**
  Destructor.

Static Public Attributes

- **static const char ∗ file_exists_message = “File exists. Are you sure you want to overwrite?”**
  Localizable message.
31.82 Fl_Native_File_Chooser Class Reference

31.82.1 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:

```c++
#include <FL/Fl_Native_File_Chooser.H>
```

The following example shows how to pick a single file:

```c++
// Create and post the local native file chooser
#include <FL/Fl_Native_File_Chooser.H>
[..]
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE_FILE);
fnfc.filter("Text\text{.txt}\n"C Files\text{.cxx,\text{h,\text{c)}}");
fnfc.directory("/var/tmp"); // default directory to use
// Show native chooser
switch ( fnfc.show() ) {
  case -1: printf("ERROR: %s\n", fnfc.errmsg()); break; // ERROR
  case 1: printf("CANCEL\n"); break; // CANCEL
  default: printf("PICKED: %s\n", fnfc.filename()); break; // FILE CHOSEN
}
```

The `Fl_Native_File_Chooser` widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the `fl_fopen()` or `fl_open()` utility functions that handle these names in a cross-platform way (whereas the standard `fopen()`/`open()` functions fail on the Windows platform to open files with a non-ASCII name).

### Platform Specific Caveats

- Under X windows, and if `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 `Fl_File_Chooser` if they are not. In the first case, calling `fl_register_images()` adds a "Preview" button to the GTK chooser dialog. In the latter case, it's best if you call `Fl_File_Icon::load_system_icons()` at the start of `main()`, to enable the nicer looking file browser widgets. Use the static public attributes of class `Fl_File_Chooser` to localize the browser.

- Some operating systems support certain OS specific options; see `Fl_Native_File_Chooser::options()` for a list.
31.82.2 Member Enumeration Documentation

31.82.2.1 Option

```cpp
enum Fl_Native_File_Chooser::Option
```

**Enumerator**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OPTIONS</td>
<td>no options enabled</td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Show native 'Save As' overwrite confirm dialog.</td>
</tr>
<tr>
<td>NEW_FOLDER</td>
<td>Show 'New Folder' icon (if supported)</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>enable preview mode (if supported)</td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter pilots the output file extension (if supported)</td>
</tr>
</tbody>
</table>

31.82.2.2 Type

```cpp
enum Fl_Native_File_Chooser::Type
```
## 31.82.3 Constructor & Destructor Documentation

### 31.82.3.1 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()`.

### 31.82.3.2 ~Fl_Native_File_Chooser()

```cpp
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ()
```

Destructor.

Deallocation any resources allocated to this widget.

## 31.82.4 Member Function Documentation

### 31.82.4.1 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));
    }
}
```
31.82.4.2  directory()

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.

31.82.4.3  errmsg()

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.

31.82.4.4  filename() [1/2]

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).

31.82.4.5  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:

```c
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for ( int n=0; n<fnfc->count(); n++ ) {
        printf("%s") 's'\"\n", n, fnfc->filename(n));
    }
}
```

31.82.4.6  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.
31.82.4.7 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.

31.82.4.8 filter_value() [1/2]

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.

31.82.4.9 filter_value() [2/2]

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.

31.82.4.10 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>
<tbody>
<tr>
<td>NEW_FOLDER</td>
<td>Shows the 'New Folder' button.</td>
<td>Ignored</td>
<td>Used</td>
<td>Used</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>Enables the 'Preview' mode by default.</td>
<td>Ignored</td>
<td>Ignored</td>
<td>Used</td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Confirm dialog if BROWSE_SAVE_FILE file exists.</td>
<td>Used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter pilots the output file extension.</td>
<td>Ignored</td>
<td>Used</td>
<td></td>
</tr>
</tbody>
</table>

Generated by Doxygen
31.82.4.11 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.

31.82.4.12 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

31.82.4.13 title () [1/2]

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.

31.82.4.14 title () [2/2]

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.

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser.H
- Fl_Native_File_Chooser.cxx
31.83  Fl_Nice_Slider Class Reference

Inheritance diagram for Fl_Nice_Slider:

```
   Fl_Widget
     |
     v
   Fl_Valuator
     |
     v
   Fl_Slider
     |
     v
Fl_Nice_Slider
```

Public Member Functions

- `Fl_Nice_Slider` (int X, int Y, int W, int H, const char *L=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Nice_Slider.H
- Fl_Slider.cxx

31.84  Fl_Output Class Reference

This widget displays a piece of text.

```
#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.

Additional Inherited Members

31.84.1 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.

![The quick brown fox jumped over the lazy dog.](image)

There is a single subclass, Fl_Multiline_Output, which allows you to display multiple lines of text. Fl_Multiline_Output does not provide scroll bars. If a more complete text editing widget is needed, use Fl_Text_Display instead.

The text may contain any characters except \0, and will correctly display anything, using ^X notation for unprintable control characters and \nnn notation for unprintable characters with the high bit set. It assumes the font can draw any characters in the ISO-Latin1 character set.

31.84.2 Constructor & Destructor Documentation
31.84.2.1 Fl_Output()

\begin{verbatim}
Fl_Output::Fl_Output ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char * l = 0 )
\end{verbatim}

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

31.85 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:

\begin{center}
\begin{tikzpicture}
  \node[anchor=east] (1) at (0,0) {Fl_Widget};
  \node[anchor=east] (2) at (0,-1) {Fl_Group};
  \node[anchor=east] (3) at (0,-2) {Fl_Window};
  \node[anchor=east] (4) at (0,-3) {Fl_Double_Window};
  \node[anchor=east] (5) at (0,-4) {Fl_Overlay_Window};
  \draw (1) -- (2);
  \draw (2) -- (3);
  \draw (3) -- (4);
  \draw (4) -- (5);
\end{tikzpicture}
\end{center}

Public Member Functions

- virtual Fl_Overlay_Window * as_overlay_window ()
  \textit{Return non-null if this is an Fl_Overlay_Window object.}
- int can_do_overlay ()
  \textit{Returns non-zero if there's hardware overlay support.}
- virtual void draw_overlay ()=0
  \textit{You must subclass Fl_Overlay_Window and provide this method.}
- void flush ()

Generated by Doxygen
Forces the window to be drawn, this window is also made current and calls `draw()`.

- **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.

**Protected Member Functions**

- **Fl_Overlay_Window(int W, int H, const char *l=0)**
  See `Fl_Overlay_Window::Fl_Overlay_Window(int X, int Y, int W, int H, const char *l=0)`

- **Fl_Overlay_Window(int X, int Y, int W, int H, const char *l=0)**
  Creates a new `Fl_Overlay_Window` widget using the given position, size, and label (title) string.

**Additional Inherited Members**

### 31.85.1 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.

### 31.85.2 Constructor & Destructor Documentation

#### 31.85.2.1 Fl_Overlay_Window()

```cpp
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.
31.85.3 Member Function Documentation

31.85.3.1 draw_overlay()

```
virtual void Fl_Overlay_Window::draw_overlay ( ) [pure virtual]
```

You must subclass Fl_Overlay_Window and provide this method.

It is just like a draw() method, except it draws the overlay. The overlay will have already been "cleared" when this is called. You can use any of the routines described in `<FL/fl_draw.H>`.

31.85.3.2 flush()

```
void Fl_Overlay_Window::flush ( ) [virtual]
```

Forces the window to be drawn, this window is also made current and calls draw().

Reimplemented from Fl_Double_Window.

31.85.3.3 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.

31.85.3.4 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().
31.85.3.5 resize()

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.

31.85.3.6 show()

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

```c
#include <Fl_Pack.H>
```

Inheritance diagram for Fl_Pack:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Pack
```

Public Types

- ```enum {
    VERTICAL = 0, HORIZONTAL = 1
  }```

Public Member Functions

- ```Fl_Pack (int X, int Y, int W, int H, const char *L=0)```
  
  Creates a new Fl_Pack widget using the given position, size, and label string.

- ```uchar horizontal () const```
  
  Returns non-zero if Fl_Pack alignment is horizontal.

- ```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.

Protected Member Functions

- ```void draw ()```
  
  Draws the widget.

Additional Inherited Members

### 31.86.1 Detailed Description

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

If `type()` is Fl_Pack::HORIZONTAL all the children are resized to the height of the Fl_Pack, and are moved next to each other horizontally. If `type()` is not Fl_Pack::HORIZONTAL then the children are resized to the width and are stacked below each other. Then the Fl_Pack resizes itself to surround the child widgets.

You may want to put the Fl_Pack inside an Fl_Scroll.

The ‘resizable()’ for Fl_Pack is set to NULL by default. Its behavior is slightly different than in a normal Fl_Group widget: only if the resizable() widget is the last widget in the group it is extended to take the full available width or height, respectively, of the Fl_Pack group.
Note

You can nest Fl_Pack widgets or put them inside Fl_Scroll widgets or inside other group widgets but their behavior can sometimes be "surprising". This is partly due to the fact that Fl_Pack widgets resize themselves during their draw() operation, trying to react on their child widgets resizing themselves during their draw() operations which can be confusing. If you want to achieve special resize behavior of nested group widgets it can sometimes be easier to derive your own specialized group widget than to try to make nested Fl_Pack widgets behave as expected.

See also

Fl_Group::resizable()

31.86.2 Constructor & Destructor Documentation

31.86.2.1 Fl_Pack()

Fl_Pack::Fl_Pack (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * L = 0  
)

Creates a new Fl_Pack widget using the given position, size, and label string.

The default boxtype is FL_NO_BOX.

The default type() is Fl_Pack::VERTICAL.

The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Pack and all of its children can be automatic (local) variables, but you must declare the Fl_Pack first, so that it is destroyed last.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y</th>
<th>X and Y coordinates (position)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W,H</td>
<td>width and height, respectively</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>label (optional)</td>
</tr>
</tbody>
</table>

31.86.3 Member Function Documentation

31.86.3.1 draw()

void Fl_Pack::draw ( ) [protected], [virtual]  

Generated by Doxygen
Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

### 31.86.3.2 horizontal()

```cpp
uchar Fl_Pack::horizontal ( ) const [inline]
```

Returns non-zero if `Fl_Pack` alignment is horizontal.

Returns

- non-zero if `Fl_Pack` alignment is horizontal (`Fl_Pack::HORIZONTAL`)

**Note**

Currently the return value is the same as `Fl_Group::type()`, but this may change in the future. Do not set any other values than the following with `Fl_Pack::type()`:

- `Fl_Pack::VERTICAL` (Default)
- `Fl_Pack::HORIZONTAL`

See class `Fl_Pack` documentation for details.

The documentation for this class was generated from the following files:

- `Fl_Pack.H`
- `Fl_Pack.cxx`

### 31.87 Fl_Paged_Device Class Reference

Represents page-structured drawing surfaces.

```cpp
#include <Fl_Paged_Device.H>
```

Inheritance diagram for Fl_Paged_Device:

```
Fl_Surface_Device
    Fl_Widget_Surface
        Fl_Paged_Device
            Fl_PostScript_File_Device
            Fl_Printer
```

Generated by Doxygen
Classes

- struct page_format
  width, height and name of a page format

Public Types

- enum Page_Format {
  A0 = 0, A1, A2, A3,
  A4, A5, A6, A7,
  A8, A9, B0, B1,
  B2, B3, B4, B5,
  B6, B7, B8, B9,
  B10, CSE, DLE, EXECUTIVE,
  FOLIO, LEDGER, LEGAL, LETTER,
  TABLOID, ENVELOPE, MEDIA = 0x1000 }
  Possible page formats.

- enum Page_Layout { PORTRAIT = 0, LANDSCAPE = 0x100, REVERSED = 0x200, ORIENTATION = 0x300 }
  Possible page layouts.

Public Member Functions

- virtual int begin_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  Begins a print job.
- virtual int begin_page (void)
  Begins a new printed page.
- virtual void end_job (void)
  To be called at the end of a print job.
- virtual int end_page (void)
  To be called at the end of each page.
- virtual void margins (int *left, int *top, int *right, int *bottom)
  Computes the dimensions of margins that lie between the printable page area and the full page.
- void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  Synonym of draw(Fl_Widget*, int, int)
- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Synonym of draw_decorated_window(Fl_Window*, int, int)
- virtual void rotate (float angle)
  Rotates the graphics operations relatively to paper.
- virtual void scale (float scale_x=0., float scale_y=0.)
  Changes the scaling of page coordinates.
- int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).
- int start_page ()
  Synonym of begin_page().
- virtual ~Fl_Paged_Device ()
  The destructor.
Static Public Attributes

- static const page_format page_formats [NO_PAGE_FORMATS]
  width, height and name of all elements of the enum Page_Format.

Protected Member Functions

- Fl_Paged_Device ()
  The constructor.

Additional Inherited Members

31.87.1 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.

31.87.2 Member Enumeration Documentation

31.87.2.1 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>

31.87.2.2 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>
<tr>
<td>REVERSED</td>
<td>Reversed orientation.</td>
</tr>
<tr>
<td>ORIENTATION</td>
<td>orientation</td>
</tr>
</tbody>
</table>

---

### 31.87.3 Member Function Documentation

#### 31.87.3.1 begin_job()

```cpp
int Fl_Paged_Device::begin_job (
   int pagecount = 0,
   int * frompage = NULL,
   int * topage = NULL,
   char ** perr_message = NULL ) [virtual]
```

 Begins a print job.

**Parameters**

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecount</td>
<td>frompage</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></td>
<td>topage</td>
<td>if non-null, *frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if non-null, *topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td></td>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]'d after use.</td>
</tr>
</tbody>
</table>

**Returns**

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

#### 31.87.3.2 begin_page()

```cpp
int Fl_Paged_Device::begin_page (
   void ) [virtual]
```

 Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.

**Returns**

0 if OK, non-zero if any error

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.
31.87.3.3 end_page()

```cpp
int Fl_Paged_Device::end_page ()
  [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`, and `Fl_Printer`.

31.87.3.4 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>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 in `Fl_PostScript_File_Device`, and `Fl_Printer`.

31.87.3.5 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

| angle | Rotation angle in counter-clockwise degrees. |

Generated by Doxygen
31.87.3.6 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>scale←_x</th>
<th>Horizontal dimensions of plot are multiplied by this quantity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale←_y</td>
<td>Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor);</td>
</tr>
</tbody>
</table>

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

31.87.3.7 start_job()

```cpp
int Fl_Paged_Device::start_job (  
    int pagecount = 0,  
    int * frompage = NULL,  
    int * topage = NULL,  
    char ** perr_message = NULL ) [inline]
```

Synonym of `begin_job(int pagecount, int *frompage, int *topage, char **perr_message)`.

For API compatibility with FLTK 1.3.x

31.87.3.8 start_page()

```cpp
int Fl_Paged_Device::start_page ( ) [inline]
```

Synonym of `begin_page()`.

For API compatibility with FLTK 1.3.x

The documentation for this class was generated from the following files:

- Fl_Paged_Device.H
- Fl_Paged_Device.cxx
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_XPM_Image Fl/gif/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`.

- `virtual Fl_Image * copy (int W, int H)`
  
  Creates a resized copy of the specified image.

- `Fl_Image * copy ()`

- `virtual void desaturate ()`
  
  The `desaturate()` method converts an image to grayscale.

- `virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)`
  
  Draws the image to the current drawing surface with a bounding box.

- `void draw (int X, int Y)`

- `Fl_Pixmap (char *const *D)`
  
  The constructors create a new pixmap from the specified XPM data.

- `Fl_Pixmap (uchar *const *D)`

- `Fl_Pixmap (const char *const *D)`

- `Fl_Pixmap (const uchar *const *D)`

- `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 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 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 Attributes

- `int alloc_data`
Protected Member Functions

- void measure ()

Friends

- class Fl_Graphics_Driver

Additional Inherited Members

31.88.1 Detailed Description

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap ) images, including transparency.

31.88.2 Constructor & Destructor Documentation

31.88.2.1 Fl_Pixmap() [1/4]

Fl_Pixmap::Fl_Pixmap ( char const * D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

31.88.2.2 Fl_Pixmap() [2/4]

Fl_Pixmap::Fl_Pixmap ( uchar const * D ) [inline], [explicit]

The constructors create a new pixmap from the specified XPM data.

31.88.2.3 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.
31.88.2.4 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.

31.88.3 Member Function Documentation

31.88.3.1 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.

31.88.3.2 copy()

Fl_Image * Fl_Pixmap::copy (int W, int H) [virtual]

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.

Parameters

| W/H | width and height of the returned copied image |

Reimplemented from Fl_Image.

31.88.3.3 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`.

### 31.88.3.4 draw()

```cpp
void Fl_Pixmap::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box.

Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.

In other words: `fl_push_clip(X,Y,W,H)` is applied, the image is drawn with its upper-left corner at X-cx, Y-cy and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from `Fl_Image`.

### 31.88.3.5 label()[1/2]

```cpp
void Fl_Pixmap::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`.

### 31.88.3.6 label()[2/2]

```cpp
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`. 
31.88.3.7  uncache()

void Fl_Pixmap::uncache ( ) [virtual]
If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:
  • Fl_Pixmap.H
  • Fl_Pixmap.cxx

31.89  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.

31.89.1  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 option-
ally 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:
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.

Fl_Plugin_Manager mgr("effects");
int i, n = mgr.plugins();
for (i=0; i<n; i++) {
  My_Plugin *pin = (My_Plugin*)mgr.plugin(i);
  pin->do_something();
}
### 31.89.2 Constructor & Destructor Documentation

#### 31.89.2.1 Fl_Plugin()

Fl_Plugin::Fl_Plugin (  
    const char ∗ klass,  
    const char ∗ name )  

Create a plugin.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>klass</th>
<th>plugins are grouped in classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>every plugin should have a unique name</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Plugin.H
- Fl_Preferences.cxx

### 31.90 Fl_Plugin_Manager Class Reference

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

#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 (int index)**
  
  *Return the address of a plugin by index.*

- **Fl_Plugin ∗ plugin (const char ∗name)**
  
  *Return the address of a plugin by name.*

- **int plugins ()**
  
  *Return the number of plugins in the klass.*

- **~Fl_Plugin_Manager ()**
  
  *Remove the plugin manager.*
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.

Additional Inherited Members

31.90.1 Detailed Description

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

See also
   Fl_Plugin

31.90.2 Constructor & Destructor Documentation

31.90.2.1 ~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.

31.90.3 Member Function Documentation

31.90.3.1 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.
31.90.3.2 load()

```c
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.

31.90.3.3 removePlugin()

```c
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`

31.91 `Fl_PNG_Image` Class Reference

The `Fl_PNG_Image` class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

```c
#include <Fl_PNG_Image.H>
```

Inheritance diagram for `Fl_PNG_Image`:

```
Fl_PNG_Image
  |  
  v  
Fl_RGB_Image
  |  
  v  
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.*
31.91 Fl_PNG_Image Class Reference

Additional Inherited Members

31.91.1 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.

31.91.2 Constructor & Destructor Documentation

31.91.2.1 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 |

31.91.2.2 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 |

Generated by Doxygen
The documentation for this class was generated from the following files:

- Fl_PNG_Image.H
- Fl_PNG_Image.cxx

## 31.92 Fl_PNM_Image Class Reference

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

```c
#include <Fl_PNM_Image.H>
```

Inheritance diagram for Fl_PNM_Image:

```
Fl_PNM_Image
|     
|     Fl_RGB_Image
|     Fl_PNM_Image
```

### Public Member Functions

- **Fl_PNM_Image (const char ∗filename)**

  The constructor loads the named PNM image.

### Additional Inherited Members

#### 31.92.1 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.

#### 31.92.2 Constructor & Destructor Documentation

##### 31.92.2.1 Fl_PNM_Image()

```c
Fl_PNM_Image::Fl_PNM_Image (const char * filename)
```

The constructor loads the named PNM image.

The destructor frees all memory and server resources that are used by the image.

Use Fl_Image: :fail() to check if Fl_PNM_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNM format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.
Parameters

| in  | filename | a full path and name pointing to a valid jpeg file. |

The documentation for this class was generated from the following files:

- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

### 31.93 Fl_Positioner Class Reference

This class is provided for Forms compatibility.

```c
#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 xminimum** () const
  
  Gets the X axis minimum.

- **void xminimum** (double a)
  
  Same as xbounds(a, xmaximum())

- **void xstep** (double a)
  
  Sets the stepping value for the X axis.

- **double xvalue** () const
  
  Gets the X axis coordinate.

- **int 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.

Protected Member Functions

- void draw (int, int, int, int)
- void draw ()
  
  Draws the widget.

- int handle (int, int, int, int)

Additional Inherited Members

31.93.1 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().

![Fl_Positioner](Image)

Figure 31.29 Fl_Positioner

31.93.2 Constructor & Destructor Documentation
31.93 Fl_Positioner Class Reference

31.93.2.1 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.

31.93.3 Member Function Documentation

31.93.3.1 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:

Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

31.93.3.2 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.

31.93.3.3 value()

```cpp
int Fl_Positioner::value ( double X, double Y )
```

Returns the current position in x and y.

31.93.3.4 xbounds()

```cpp
void Fl_Positioner::xbounds ( double a, double b )
```

Sets the X axis bounds.

31.93.3.5 xstep()

```cpp
void Fl_Positioner::xstep ( double s ) [inline]
```

Sets the stepping value for the X axis.
31.93.3.6 xvalue() [1/2]

double Fl_Positioner::xvalue ( ) const [inline]

Gets the X axis coordinate.

31.93.3.7 xvalue() [2/2]

int Fl_Positioner::xvalue ( double X )

Sets the X axis coordinate.

31.93.3.8 ybounds()

void Fl_Positioner::ybounds ( double a, double b )

Sets the Y axis bounds.

31.93.3.9 ystep()

void Fl_Positioner::ystep ( double a ) [inline]

Sets the stepping value for the Y axis.

31.93.3.10 yvalue() [1/2]

double Fl_Positioner::yvalue ( ) const [inline]

Gets the Y axis coordinate.
### 31.93.3.11  \texttt{yvalue()} [2/2]

```c
int Fl_Positioner::yvalue ( double Y )
```

Sets the Y axis coordinate.

The documentation for this class was generated from the following files:

- \texttt{Fl_Positioner.H}
- \texttt{Fl_Positioner.cxx}

### 31.94  \texttt{Fl_PostScript_File_Device} Class Reference

To send graphical output to a PostScript file.

```c
#include <Fl_PostScript.H>
```

Inheritance diagram for \texttt{Fl_PostScript_File_Device}:

```
Fl_Surface_Device
    ↓
Fl_Widget_Surface
    ↓
Fl_Paged_Device
    ↓
Fl_PostScript_File_Device
```

**Public Member Functions**

- int \texttt{begin\_job} (int pagecount, int *from, int *to, char **perr\_message)
  
  *Don't use with this class.*

- int \texttt{begin\_job} (int pagecount=0, enum \texttt{Fl\_Paged\_Device::Page\_Format} format=\texttt{Fl\_Paged\_Device::A4}, enum \texttt{Fl\_Paged\_Device::Page\_Layout} layout=\texttt{Fl\_Paged\_Device::PORTRAIT})

  *Begins the session where all graphics requests will go to a local PostScript file.*

- int \texttt{begin\_job} (FILE *ps\_output, int pagecount=0, enum \texttt{Fl\_Paged\_Device::Page\_Format} format=\texttt{Fl\_Paged\_Device::A4}, enum \texttt{Fl\_Paged\_Device::Page\_Layout} layout=\texttt{Fl\_Paged\_Device::PORTRAIT})

  *Begins the session where all graphics requests will go to FILE pointer.*

- int \texttt{begin\_page} (void)

  *Begins a new printed page.*

- void \texttt{end\_job} (void)

  *To be called at the end of a print job.*

- int \texttt{end\_page} (void)

  *To be called at the end of each page.*

- FILE * \texttt{file} ()

  *Returns the underlying FILE* receiving all PostScript data.*

- \texttt{Fl\_PostScript\_File\_Device} ()
The constructor.

• void 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 coordinates of the current origin of graphics functions.

• void origin (int x, int y)

  Sets the position of the origin of graphics in the drawable part of the drawing surface.

• int printable_rect (int *w, int *h)

  Computes the width and height of the drawable area of the drawing surface.

• 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=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Synonym of begin_job().

• int start_job (FILE *ps_output, int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Synonym of begin_job().

• 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.

Static Public Attributes

• static const char * file_chooser_title

  Label of the PostScript file chooser window.

Protected Member Functions

• Fl_PostScript_Graphics_Driver * driver ()

  Returns the PostScript driver of this drawing surface.

Additional Inherited Members

31.94.1 Detailed Description

To send graphical output to a PostScript file.

This class is used exactly as the Fl_Printer class except for the begin_job() call, two variants of which are usable and allow to specify what page format and layout are desired.

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.
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.

### 31.94.2 Member Function Documentation

#### 31.94.2.1 begin_job() [1/3]

```cpp
int Fl_PostScript_File_Device::begin_job (  
    int pagecount,
    int * from,
    int * to,
    char ** perr_message ) [virtual]
```

Don't use with this class.

Reimplemented from `Fl_Paged_Device`.

#### 31.94.2.2 begin_job() [2/3]

```cpp
int Fl_PostScript_File_Device::begin_job (  
    int pagecount = 0,
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to a local PostScript file.

Opens a file dialog entitled with `Fl_PostScript_File_Device::file_chooser_title` to select an output PostScript file.
31.94 Fl_PostScript_File_Device Class Reference

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 to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>Desired page format.</td>
</tr>
<tr>
<td><code>layout</code></td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the file dialog, 2 if fopen failed on user-selected output file.

31.94.2.3 `begin_job()` [3/3]

```cpp
int Fl_PostScript_File_Device::begin_job (  
    FILE * ps_output,  
    int pagecount = 0,  
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,  
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to FILE pointer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ps_output</code></td>
<td>A writable FILE pointer that will receive PostScript output and that should not be closed until after <code>end_job()</code> has been called.</td>
</tr>
<tr>
<td><code>pagecount</code></td>
<td>The total number of pages to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>Desired page format.</td>
</tr>
<tr>
<td><code>layout</code></td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

always 0.

31.94.2.4 `begin_page()`

```cpp
int Fl_PostScript_File_Device::begin_page (  
    void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.

Returns

0 if OK, non-zero if any error

Reimplemented from `Fl_Paged_Device`. 

Generated by Doxygen
31.94.2.5 end_page()

```cpp
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.

31.94.2.6 margins()

```cpp
void Fl_PostScript_File_Device::margins (
    int ∗ left,
    int ∗ top,
    int ∗ right,
    int ∗ bottom ) [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.

Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

**Parameters**

<table>
<thead>
<tr>
<th>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.

31.94.2.7 origin() [1/2]

```cpp
void Fl_PostScript_File_Device::origin (
    int ∗ x,
    int ∗ y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>x,y</th>
<th>If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.</th>
</tr>
</thead>
</table>

Generated by Doxygen
Reimplemented from Fl_Widget_Surface.

### 31.94.2.8 origin()

```c
void Fl_PostScript_File_Device::origin ( int x, int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from Fl_Paged_Device).

**Parameters**

| in | x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics |

Reimplemented from Fl_Widget_Surface.

### 31.94.2.9 printable_rect()

```c
int Fl_PostScript_File_Device::printable_rect ( int * w, int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**

- 0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

### 31.94.2.10 rotate()

```c
void Fl_PostScript_File_Device::rotate ( float angle ) [virtual]
```

Rotates the graphics operations relatively to paper.

The rotation is centered on the current graphics origin. Successive `rotate()` calls don't combine their effects.
Parameters

| angle | Rotation angle in counter-clockwise degrees. |

Reimplemented from **Fl_Paged_Device**.

### 31.94.2.11 scale()

```cpp
template<typename Fl_Paged_Device::scale ( 
  float scale_x,
  float scale_y = 0. ) [virtual]
```

Changes the scaling of page coordinates.

This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don’t combine their effects.

Parameters

| scale←-_x | Horizontal dimensions of plot are multiplied by this quantity. |
| scale←-_y | Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor); |

Reimplemented from **Fl_Paged_Device**.

### 31.94.2.12 start_job() [1/2]

```cpp
int Fl_PostScript_File_Device::start_job ( 
  int pagecount = 0,
  enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
  enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT ) [inline]
```

Synonym of `begin_job()`.

For API compatibility with FLTK 1.3.x

### 31.94.2.13 start_job() [2/2]

```cpp
int Fl_PostScript_File_Device::start_job ( 
  FILE * ps_output,
  int pagecount = 0,
  enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
  enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT ) [inline]
```

Synonym of `begin_job()`.

For API compatibility with FLTK 1.3.x
31.94.2.14 translate()

```cpp
void Fl_PostScript_File_Device::translate (  
    int x,  
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.

Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following file:

- `Fl_PostScript.H`

## 31.95 Fl_Preferences Class Reference

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

```cpp
#include <Fl_Preferences.H>
```

Inheritance diagram for `Fl_Preferences`:

```
Fl_Preferences
\|-- Fl_Plugin_Manager
```

### Classes

- struct `Entry`
- class `Name`
  - *'Name'* provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.
- class `Node`
- class `RootNode`

### Public Types

- typedef `void *ID`
  - Every `Fl_Preferences`-Group has a unique ID.
- enum `Root`
  - `SYSTEM = 0`, `USER`, `ROOT_MASK = 0xFF`, `CORE = 0x100`, `CORE_SYSTEM = CORE | SYSTEM`, `CORE_USER = CORE | USER`
  - Define the scope of the preferences.

Generated by Doxygen
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 (Root root, const char ∗ vendor, const char ∗ application)
  *The constructor creates a group that manages name/value pairs and child groups.*
- 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 (Fl_Preferences &parent, const char ∗ group)
  *Generate or read a new group of entries within another group.*
- 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)
  *Open a child group using a given index.*
- Fl_Preferences (Fl_Preferences ∗ parent, int groupIndex)
  *Create another reference to a Preferences group.*
- Fl_Preferences (ID id)
  *Create a new dataset access point using a dataset ID.*
- void flush ()
  *Writes all preferences to disk.*
- char get (const char ∗ entry, int &value, int 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, double &value, double defaultValue)
  *Reads an entry from the group.*
- 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, 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, int value)
  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, 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, 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).

• 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 void file_access (unsigned int flags)
  Tell the FLTK preferences system which files in the file system it may read, create, or write.

• static unsigned int file_access ()
  Return the current file access permissions for the FLTK preferences system.

• 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.
Static Public Attributes

- static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK
  
  set this to give FLTK and applications permission to read, write, and create preference files
- static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK
  
  set this to allow FLTK and applications to read preference files
- static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK
  
  set this to allow FLTK and applications to create and write preference files
- static const unsigned int APP_OK = SYSTEM_OK | USER_OK
  
  set this if it is ok for applications to read, create, and write any kind of preference files
- static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK
  
  set this if it is ok for FLTK to read, create, or write preference files
- static const unsigned int CORE_READ_OK = 0x0010
  
  Set this if it is ok for FLTK to read preference files.
- static const unsigned int CORE_WRITE_OK = 0x0020
  
  Set this if it is ok for FLTK to create or write preference files.
- static const unsigned int NONE = 0x0000
  
  Set this, if no call to Fl_Preferences shall access the file system.
- static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK
  
  set this if it is ok for applications to read, create, and write system wide preference files
- static const unsigned int SYSTEM_READ_OK = 0x0004
  
  set this if it is ok for applications to read system wide preference files
- static const unsigned int SYSTEM_WRITE_OK = 0x0008
  
  set this if it is ok for applications to create and write system wide preference files
- static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK
  
  set this if it is ok for applications to read, create, and write user preference files
- static const unsigned int USER_READ_OK = 0x0001
  
  set this if it is ok for applications to read user preference files
- static const unsigned int USER_WRITE_OK = 0x0002
  
  set this if it is ok for applications to create and write user preference files

Protected Attributes

- Node * node
- RootNode * rootNode

Friends

- class Node
- class RootNode
31.95.1 Detailed Description

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

It is similar to the Registry on Windows 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 in these 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 **Fl_Preferences::getuserinfoPath**().

**Note**

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preferences files has changed slightly. Please see **Fl_Preferences::Fl_Preferences(Root, const char**, **const char**) for details.

31.95.2 Member Typedef Documentation

31.95.2.1 ID

typedef void* Fl_Preferences::ID

Every Fl_Preferences-Group has a unique ID.

ID’s can be retrieved from an Fl_Preferences-Group and can then be used to create more Fl_Preference references to the same data set, as long as the database remains open.

31.95.3 Member Enumeration Documentation

31.95.3.1 Root

enum Fl_Preferences::Root

Define the scope of the preferences.
### Class Documentation

#### Enumerator

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>Preferences are used system-wide.</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>Preferences apply only to the current user.</td>
</tr>
<tr>
<td>ROOT_MASK</td>
<td>masks for the values above</td>
</tr>
<tr>
<td>CORE</td>
<td>OR'd by FLTK to read and write core library preferences and options.</td>
</tr>
</tbody>
</table>

#### 31.95.4 Constructor & Destructor Documentation

##### 31.95.4.1 Fl_Preferences() [1/7]

```cpp
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. "research.matthiasm.com" or "fluid.fltk.org". The application argument can be the working title or final name of your application. Both vendor and application must be valid UNIX path segments and may contain forward slashes to create deeper file structures.

A set of Preferences marked "run-time" exists exactly once per application and only as long as the application runs. It can be used as a database for volatile information. FLTK uses it to register plugins at run-time.

**Note**

On **Windows**, the directory is constructed by querying the `Common AppData` or `AppData` key of the `Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders` registry entry. The filename and path is then constructed as `${query}/${vendor}/${application}.prefs` . If the query call fails, data will be stored in RAM only and be lost when the app exits.

In FLTK versions before 1.4.0, if querying the registry failed, preferences would be written to

```
C:\FLTK\${vendor}\${application}.prefs.
```

**Note**

On **Linux**, the `USER` directory is constructed by reading `$HOME` . If `$HOME` is not set or not pointing to an existing directory, we are checking the path member of the passwd struct returned by `getpwuid(getuid())` . If all attempts fail, data will be stored in RAM only and be lost when the app exits. The filename and path is then constructed as `${directory}/.fltk/${vendor}/${application}.prefs`. The `SYSTEM` directory is hardcoded as `/etc/fltk/${vendor}/${application}.prefs`.

In FLTK versions before 1.4.0, if `$HOME` was not set, the `USER` path would be empty, generating `${vendor}/${application}.prefs`, which was used relative to the current working directory.
Note

On macOS, the USER directory is constructed by reading $HOME. If $HOME is not set or not pointing to an existing directory, we check the path returned by NSHomeDirectory(), and finally checking the path member of the passwd struct returned by getpwuid(getuid()). If all attempts fail, data will be stored in RAM only and be lost when the app exits. The filename and path is then constructed as $(directory)/~Library/Preferences/${vendor}/${application}.prefs. The SYSTEM directory is hard-coded as /Library/Preferences/${vendor}/${application}.prefs.

In FLTK versions before 1.4.0, if $HOME was not set, the USER path would be NULL, generating <null>/Library/Preferences/${vendor}/${application}.prefs, which would silently fail to create a preferences file.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>path to the directory that contains the preferences file</td>
</tr>
<tr>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>

### Todo (Matt)
Before the release of 1.4.0, I want to make a further attempt to write a preferences file smarter. I plan to use a subgroup of the "runtime" preferences to store data and stay accessible until the application exits. Data would be stored under .://${vendor}/${application}.prefs in RAM, but not on disk.

### Todo (Matt)
I want a way to access the type of the root preferences (SYSTEM, USER, MEMORY), and the state of the file access (OK, FILE_SYSTEM_FAIL, PERMISSION_FAIL, etc.), and probably the dirty() flag as well.

### Todo (Matt)
Also, I need to explain runtime preferences.

### Todo (Matt)
Lastly, I think I have to put short sample code in the Doxygen docs. The test app ist just not enough.

---

31.95.4.2 Fl_Preferences()

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 ./.prefs. If application is NULL, path is taken literally as the file path and name.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>path to the directory that contains the preferences file</td>
</tr>
<tr>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>

Generated by Doxygen
31.95.4.3 Fl_Preferences() [3/7]

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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>parent</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>group</code></td>
</tr>
</tbody>
</table>

31.95.4.4 Fl_Preferences() [4/7]

Fl_Preferences::Fl_Preferences (  
  Fl_Preferences * parent,  
  const char * group )

Create or access a group of preferences using a name.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><code>parent</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>group</code></td>
</tr>
</tbody>
</table>

See also

Fl_Preferences( Fl_Preferences&, const char *group )

31.95.4.5 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 |
|     | groupIndex | zero based index into child groups |

31.95.4.6 Fl_Preferences()

Fl_Preferences::Fl_Preferences (  
    Fl_Preferences ∗ parent,  
    int groupIndex )

See also

    Fl_Preferences( Fl_Preferences&, int groupIndex )

31.95.4.7 Fl_Preferences()

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.

31.95.4.8 ~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.

31.95.5 Member Function Documentation

31.95.5.1 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 |

Returns

0 if deleting the entry failed

31.95.5.2 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

| in | group | name of the group to delete |

Returns

0 if call failed

31.95.5.3 entries()

```cpp
int Fl_Preferences::entries ( )
```

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

31.95.5.4 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

| in  | index | number indexing the requested entry |

Returns

pointer to value cstring

31.95.5.5 entryExists()

char Fl_Preferences::entryExists (const char *key)

Returns non-zero if an entry with this name exists.

Parameters

| in  | key | name of entry that is searched for |

Returns

0 if entry was not found

31.95.5.6 file_access() [1/2]

void Fl_PREFERENCES::file_access (unsigned int flags) [static]

Tell the FLTK preferences system which files in the file system it may read, create, or write.

The FLTK core library will try to read or even create or write preference files when calling Fl::option(), Fl_FileChooser, the printing panel, and possibly some other internal functions. If your application wants to keep FLTK from touching the file system, call this function before making any other FLTK calls:

```c
// neither FLTK nor the app may read, create, or write preference files
Fl_PREFERENCES::file_access( Fl_PREFERENCES::NONE );
```

or

```c
// FLTK may not read, create, or write preference files, but the application may
Fl_PREFERENCES::file_access( Fl_PREFERENCES::APP_OK );
```

All flags can be combined using an OR operator. If flags are not set, that specific access to the file system will not be allowed. By default, all access is granted. To clear one or more flags from the default setting, use:

```c
Fl_PREFERENCES::file_access(
    Fl_PREFERENCES::file_access()
    &~ Fl_PREFERENCES::SYSTEM_WRITE );
```

If preferences are created using a filename (instead of Fl_PREFERENCES::USER or Fl_PREFERENCES::SYSTEM), file access is handled as if the Fl_PREFERENCES::USER flag was set.

Generated by Doxygen
See also

Fl_Preferences::NONE and others for a list of flags.
Fl_Preferences::file_access()

31.95.5.7  file_access() [2/2]

unsigned int Fl_Preferences::file_access ( ) [static]

Return the current file access permissions for the FLTK preferences system.

See also

Fl_Preferences::file_access(unsigned int)

31.95.5.8  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.

31.95.5.9  get() [1/7]

char Fl_Preferences::get (  
    const char * key,  
    int & value,  
    int defaultValue )

Reads an entry from the group.

A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used
31.95.5.10 get() [2/7]

```cpp
char Fl_Preferences::get (  
    const char * key,  
    float & value,  
    float defaultValue )
```

Reads an entry from the group.

A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

**Returns**

0 if the default value was used

31.95.5.11 get() [3/7]

```cpp
char Fl_Preferences::get (  
    const char * key,  
    double & value,  
    double defaultValue )
```

Reads an entry from the group.

A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

**Returns**

0 if the default value was used

Generated by Doxygen
31.95.5.12 get() [4/7]

```cpp
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 freed 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

31.95.5.13 get() [5/7]

```cpp
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
31.95 Fl_Preferences Class Reference

31.95.5.14 get() [6/7]

```cpp
char Fl_Preferences::get ( 
    const char * key, 
    void * & data, 
    const void * defaultValue, 
    int defaultSize )
```

Reads an entry from the group.

A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). `get()` allocates memory of sufficient size to hold the value. The buffer must be `free()`d by the developer using 'free(value)'.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

31.95.5.15 get() [7/7]

```cpp
char 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.

### 31.95.5.16 getUserdataPath()

```cpp
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.

There is no way to verify that the path name fit into the buffer. If the name is too long, it will be clipped.

This function can be used with direct paths that don't end in .prefs. `getUserDataPath()` will remove any extension and end the path with a `/`. If the file name has no extension, `getUserDataPath()` will append .data/ to the path name.

**Example:**

```cpp
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( path, FL_PATH_MAX );
```

creates the preferences database in the directory (User 'matt' on Linux):

`/Users/matt/.fltk/matthiasm.com/test.prefs`

...and returns the userdata path:

`/Users/matt/.fltk/matthiasm.com/test/`

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>path</th>
<th>buffer for user data path</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pathlen</td>
<td>size of path buffer (should be at least FL_PATH_MAX)</td>
</tr>
</tbody>
</table>

**Returns**

1 if there is no filename (path will be unmodified)
1 if pathlen is 0 (path will be unmodified)
1 if a path was created successfully, path will contain the path name ending in a '/'
0 if path was not created for some reason; path will contain the pathname that could not be created
See also

Fl_Preferences::Fl_Preferences(Root, const char*, const char*)

31.95.5.17  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

31.95.5.18  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 |

Returns

0 if no group by that name was found

31.95.5.19  groups()

int Fl_Preferences::groups ( )

Returns the number of groups that are contained within a group.
Returns

0 for no groups at all

31.95.5.20  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!

31.95.5.21  set() [1/7]

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

31.95.5.22  set() [2/7]

char Fl_Preferences::set (  
    const char * key,  
    float value )

Generated by Doxygen
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
**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

---

**31.95.5.25  set() [5/7]**

def 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>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

---

**31.95.5.26  set() [6/7]**

def 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>
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

- **key**: name of entry
- **data**: set this entry to value
- **dsize**: size of data array

Returns

0 if setting the value failed

Returns the size of the value part of an entry.

Parameters

- **key**: name of entry

Returns

size of value
31.95.6.1 CORE_READ_OK

const unsigned int Fl_Preferences::CORE_READ_OK = 0x0010 [static]

Set this if it is ok for FLTK to read preference files.

USER_READ_OK and/or SYSTEM_READ_OK must also be set.

31.95.6.2 CORE_WRITE_OK

const unsigned int Fl_Preferences::CORE_WRITE_OK = 0x0020 [static]

Set this if it is ok for FLTK to create or write preference files.

USER_WRITE_OK and/or SYSTEM_WRITE_OK must also be set.

31.95.6.3 NONE

const unsigned int Fl_Preferences::NONE = 0x0000 [static]

Set this, if no call to Fl_Preferences shall access the file system.

See also

Fl_Preferences::file_access(unsigned int)
Fl_Preferences::file_access()

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

31.96 Fl_Printer Class Reference

OS-independent print support.

#include <Fl_Printer.H>

Inheritance diagram for Fl_Printer:

```
Fl_Surface_Device
    Fl_Widget_Surface
        Fl_Paged_Device
            Fl_Printer
```
Public Member Functions

- int **begin_job** (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  
  Begins a print job.

- int **begin_page** (void)
  
  Begins a new printed page.

- void **draw_decorated_window** (Fl_Window *win, int x_offset=0, int y_offset=0)
  
  Draws a window with its title bar and frame if any.

- void **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.

- virtual bool **is_current** ()
  
  Is this surface the current drawing surface?

- 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 coordinates of the current origin of graphics functions.

- void **origin** (int x, int y)
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- int **printable_rect** (int *w, int *h)
  
  Computes the width and height of the drawable area of the drawing surface.

- 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.

- 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.

Static Public Attributes

These attributes are useful for the Linux/Unix platform only.

- static const char * **dialog_title** = "Print"
  
  [this text may be customized at run-time]

- static const char * **dialog_printer** = "Printer:"
  
  [this text may be customized at run-time]

- static const char * **dialog_range** = "Print Range"
  
  [this text may be customized at run-time]

- static const char * **dialog_copies** = "Copies"
  
  [this text may be customized at run-time]

- static const char * **dialog_all** = "All"
  
  [this text may be customized at run-time]

- static const char * **dialog_pages** = "Pages"
31.96.1 Detailed Description

OS-independent print support.

**FL_Printer** allows to use all drawing, color, text, image, and clip FLTK functions, and to have them operate on printed page(s). There are two main, non exclusive, ways to use it.

- Print any widget (standard, custom, **Fl_Window**, **Fl_Gl_Window**) as it appears on screen, with optional translation, scaling and rotation. This is done by calling `print_widget()`, `print_window()` or `print_window_part()`.

- Use a series of FLTK graphics commands (e.g., font, text, lines, colors, clip, image) to compose a page appropriately shaped for printing.

In both cases, begin by `begin_job()`, `begin_page()`, `printable_rect()` and `origin()` calls and finish by `end_page()` and `end_job()` calls.

Example of use: print a widget centered in a page

```cpp
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>

int width, height;
Fl_Widget *widget = ... // a widget we want printed
Fl_Printer *printer = new Fl_Printer();
if (printer->begin_job(1) == 0) {
    printer->begin_page();
    printer->printable_rect(&width, &height);
    fl_color(FL_BLACK);
    fl_line_style(FL_SOLID, 2);
    fl_rect(0, 0, width, height);
    fl_font(FL_COURIER, 12);
    time_t now; time(&now); fl_draw(ctime(&now), 0, fl_height());
    printer->origin(width/2, height/2);
    printer->print_widget(widget, -widget->w()/2, -widget->h()/2);
    printer->end_page();
    printer->end_job();
}
delete printer;
```
Platform specifics

- Unix/Linux platforms:
  - FLTK expresses all graphics data using (Level 2) PostScript and sends that to the selected printer. See class `Fl_PostScript_File_Device` for a description of how text appears in print.
  - Unless it has been previously changed, the default paper size is A4. To change that, press the "Properties" button of the "Print" dialog window opened by an `Fl_Printer::begin_job()` call. This opens a "Printer Properties" window where it's possible to select the adequate paper size. Finally press the "Save" button therein to assign the chosen paper size to the chosen printer for this and all further print operations.
  - Class `Fl_RGB_Image` prints but loses its transparency if it has one.
  - Use the static public attributes of this class to set the print dialog to other languages than English. For example, the "Printer:" dialog item `Fl_Printer::dialog_printer` can be set to French with:
    ```
    Fl_Printer::dialog_printer = "Imprimante:";
    ```
    before creation of the `Fl_Printer` object.
  - Use `Fl_PostScript_File_Device::file_chooser_title` to customize the title of the file chooser dialog that opens when using the "Print To File" option of the print dialog.

- Windows platform: Transparent `Fl_RGB_Image`'s don't print with exact transparency on most printers (a workaround is to use `print_window_part()`). `Fl_RGB_Image`'s don't rotate() well.

- Mac OS X platform: all graphics requests print as on display and accept rotation and scaling.

31.96.2 Member Function Documentation

31.96.2.1 begin_job()

```cpp
int Fl_Printer::begin_job ( 
  int pagecount = 0, 
  int * frompage = NULL, 
  int * topage = NULL, 
  char ** perr_message = NULL ) [virtual]
```

Begins a print job.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pagecount</td>
<td>the total number of pages of the job (or 0 if you don't know the number of pages)</td>
</tr>
<tr>
<td>out</td>
<td>frompage</td>
<td>if non-null, *frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>topage</td>
<td>if non-null, *topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]'d after use.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented from `Fl_Paged_Device`. 

---

Generated by Doxygen
31.96.2.2 begin_page()

```cpp
int Fl_Printer::begin_page ( void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

31.96.2.3 draw_decorated_window()

```cpp
void Fl_Printer::draw_decorated_window ( Fl_Window * win,
                                        int win_offset_x = 0,
                                        int win_offset_y = 0 ) [virtual]
```

Draws a window with its title bar and frame if any.

`win_offset_x` and `win_offset_y` are optional coordinates of where to position the window top left. Equivalent to `draw()` if `win` is a subwindow or has no border. Use `Fl_Window::decorated_w()` and `Fl_Window::decorated_h()` to get the size of the framed window.

Reimplemented from Fl_Widget_Surface.

31.96.2.4 end_page()

```cpp
int Fl_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.

31.96.2.5 margins()

```cpp
void Fl_Printer::margins ( int * left,
                          int * top,
                          int * right,
                          int * bottom ) [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.

Values are in the same unit as that used by FLTK drawing functions. They are changed by `scale()` calls.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out left</code></td>
<td>If non-null, <code>*left</code> is set to the left margin size.</td>
</tr>
<tr>
<td><code>out top</code></td>
<td>If non-null, <code>*top</code> is set to the top margin size.</td>
</tr>
<tr>
<td><code>out right</code></td>
<td>If non-null, <code>*right</code> is set to the right margin size.</td>
</tr>
<tr>
<td><code>out bottom</code></td>
<td>If non-null, <code>*bottom</code> is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Paged_Device`.

---

**31.96.2.6 origin() [1/2]**

```cpp
global void Fl_Printer::origin (int *x, int *y) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out x, y</code></td>
<td>If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Widget_Surface`.

---

**31.96.2.7 origin() [2/2]**

```cpp
global void Fl_Printer::origin (int x, int y) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in x, y</code></td>
<td>Horizontal and vertical positions in the drawing surface of the desired origin of graphics.</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Widget_Surface`. 

Generated by Doxygen
31.96.2.8 printable_rect()

```cpp
text int Fl_Printer::printable_rect (  
    int * w,  
    int * h ); [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

Returns

0 if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`.

31.96.2.9 rotate()

```cpp
text void Fl_Printer::rotate (  
    float angle ); [virtual]
```

Rotates the graphics operations relatively to paper.

The rotation is centered on the current graphics origin. Successive `rotate()` calls don’t combine their effects.

Parameters

- **angle**: Rotation angle in counter-clockwise degrees.

Reimplemented from `Fl_Paged_Device`.

31.96.2.10 scale()

```cpp
text 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{scale}_x)</td>
<td>Horizontal dimensions of plot are multiplied by this quantity.</td>
</tr>
<tr>
<td>(\text{scale}_y)</td>
<td>Same as above, vertically. The value 0. is equivalent to setting (\text{scale}_y = \text{scale}_x). Thus, (\text{scale(factor)}) is equivalent to (\text{scale(factor, factor)}).</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

31.96.2.11 set_current()

```cpp
void Fl_Printer::set_current ()
{
    void ) [virtual]
```

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Reimplemented from Fl_Surface_Device.

31.96.2.12 translate()

```cpp
void Fl_Printer::translate ( int x, int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.

Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following files:

- Fl_Printer.H
- Fl_Printer.cxx

31.97 Fl_Progress Class Reference

Displays a progress bar for the user.

```cpp
#include <Fl_Progress.H>
```

Inheritance diagram for Fl_Progress:

```
Fl_Widget
     |
     v
Fl_Progress
```

Generated by Doxygen
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.

- `void maximum (float v)`
  Sets the maximum value in the progress widget.

- `float maximum () const`
  Gets the maximum value in the progress widget.

- `void minimum (float v)`
  Sets the minimum value in the progress widget.

- `float minimum () const`
  Gets the minimum value in the progress widget.

- `void value (float v)`
  Sets the current value in the progress widget.

- `float value () const`
  Gets the current value in the progress widget.

Protected Member Functions

- `virtual void draw ()`
  Draws the progress bar.

Additional Inherited Members

31.97.1 Detailed Description

Displays a progress bar for the user.

31.97.2 Constructor & Destructor Documentation

31.97.2.1 Fl_Progress()

```
Fl_Progress::Fl_Progress ( 
  int X, 
  int Y, 
  int W, 
  int H, 
  const char * L = 0 )
```

The constructor creates the progress bar using the position, size, and label.

You can set the background color with `color()` and the progress bar color with `selection_color()`, or you can set both colors together with `color(unsigned bg, unsigned sel)`.

The default colors are `FL_BACKGROUND2_COLOR` and `FL_YELLOW`, resp.
31.97.3  Member Function Documentation

31.97.3.1  draw()

```cpp
void Fl_Progress::draw {
    void ) [protected], [virtual]
```

Draws the progress bar.

Implements Fl_Widget.

31.97.3.2  maximum() [1/2]

```cpp
void Fl_Progress::maximum (  
    float v ) [inline]
```

Sets the maximum value in the progress widget.

31.97.3.3  maximum() [2/2]

```cpp
float Fl_Progress::maximum ( ) const [inline]
```

Gets the maximum value in the progress widget.

31.97.3.4  minimum() [1/2]

```cpp
void Fl_Progress::minimum (  
    float v ) [inline]
```

Sets the minimum value in the progress widget.

31.97.3.5  minimum() [2/2]

```cpp
float Fl_Progress::minimum ( ) const [inline]
```

Gets the minimum value in the progress widget.
31.97.3.6 value() [1/2]

```cpp
void Fl_Progress::value (float v) [inline]
```

Sets the current value in the progress widget.

31.97.3.7 value() [2/2]

```cpp
float Fl_Progress::value ( ) const [inline]
```

Gets the current value in the progress widget.

The documentation for this class was generated from the following files:

- Fl_Progress.H
- Fl_Progress.cxx

### 31.98 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:

```
Fl_Widget
    |
    V
Fl_Button
    |
    V
Fl_Radio_Button
```

**Public Member Functions**

- `Fl_Radio_Button (int X, int Y, int W, int H, const char *L=0)`
  
  The constructor creates the button using the given position, size, and label.

**Additional Inherited Members**

### 31.98.1 Constructor & Destructor Documentation

#### 31.98.1.1 Fl_Radio_Button()

```cpp
Fl_Radio_Button::Fl_Radio_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char *L = 0 )
```

The constructor creates the button using the given position, size, and label.

The Button type() is set to FL_RADIO_BUTTON.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Radio_Button.H
- Fl_Button.cxx

### 31.99 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)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Radio_Light_Button.H
- Fl_Light_Button.cxx

### 31.100 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
```

Generated by Doxygen
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.

Additional Inherited Members

### 31.100.1 Constructor & Destructor Documentation

#### 31.100.1.1 Fl_Radio_Round_Button()

```c
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

| in  | X, Y, W, H | position and size of the widget |
| in  | L         | widget label, default is no label |

The documentation for this class was generated from the following files:

- Fl_Radio_Round_Button.H
- Fl_Round_Button.cxx

### 31.101 Fl_Rect Class Reference

Rectangle with standard FLTK coordinates (X, Y, W, H).

```
#include <Fl_Rect.H>
```

Public Member Functions

- **int b()** const
  
  gets the bottom edge (y + h).
- **Fl_Rect()**
  
  The default constructor creates an empty rectangle (x = y = w = h = 0).
31.101 Fl_Rect Class Reference

- Fl_Rect (int W, int H)
  This constructor creates a rectangle with \( x = y = 0 \) and the given width and height.
- Fl_Rect (int X, int Y, int W, int H)
  This constructor creates a rectangle with the given \( x,y \) coordinates and the given width and height.
- Fl_Rect (const Fl_Widget &widget)
  This constructor creates a rectangle based on a widget's position and size.
- Fl_Rect (const Fl_Widget &const widget)
  This constructor creates a rectangle based on a widget's position and size.
- int h () const
  gets the height
- void h (int H)
  sets the height
- int r () const
  gets the right edge \( (x + w) \).
- int w () const
  gets the width
- void w (int W)
  sets the width
- int x () const
  gets the x coordinate (left edge)
- void x (int X)
  sets the x coordinate (left edge)
- int y () const
  gets the y coordinate (top edge)
- void y (int Y)
  sets the y coordinate (top edge)

31.101.1 Detailed Description

Rectangle with standard FLTK coordinates \((X, Y, W, H)\).

This may be used internally, for overloaded widget constructors and other overloaded methods like \texttt{fl_measure()}, \texttt{fl_text_extents()}, \texttt{fl_rect()}, \texttt{fl_rectf()}, and maybe more.

31.101.2 Constructor & Destructor Documentation

31.101.2.1 Fl_Rect()

\texttt{Fl_Rect::Fl_Rect ( ) [inline]}

The default constructor creates an empty rectangle \((x = y = w = h = 0)\).
31.101.2.2  Fl_Rect()  [2/5]

Fl_Rect::Fl_Rect (  
    int W, 
    int H ) [inline]

This constructor creates a rectangle with \(x = y = 0\) and the given width and height.

31.101.2.3  Fl_Rect()  [3/5]

Fl_Rect::Fl_Rect (  
    int X, 
    int Y, 
    int W, 
    int H ) [inline]

This constructor creates a rectangle with the given \(x,y\) coordinates and the given width and height.

31.101.2.4  Fl_Rect()  [4/5]

Fl_Rect::Fl_Rect (  
    const Fl_Widget & widget ) [inline]

This constructor creates a rectangle based on a widget's position and size.

31.101.2.5  Fl_Rect()  [5/5]

Fl_Rect::Fl_Rect (  
    const Fl_Widget *const widget ) [inline]

This constructor creates a rectangle based on a widget's position and size.

31.101.3  Member Function Documentation

31.101.3.1  b()  

int Fl_Rect::b ( ) const [inline]

gets the bottom edge \((y + h)\).

Note

\(r()\) and \(b()\) are coordinates outside the area of the rectangle.
int Fl_Rect::r() const [inline]

gets the right edge (x + w).

Note

r() and b() are coordinates outside the area of the rectangle.

The documentation for this class was generated from the following file:

• Fl_Rect.H

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

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

A local struct to manage a region defined by xywh.

#include <Fl_Scroll.H>
Public Attributes

- int h
- int w
- int x
- int y

31.103.1 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

31.104 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
   
Fl_Button
   
Fl_Repeat_Button
```

Public Member Functions

- void deactivate ()
- Fl_Repeat_Button (int X, int Y, int W, int H, const char *=0)
  
  Creates a new Fl_Repeat_Button widget using the given position, size, and label string.
- int handle (int)
  
  Handles the specified event.

Additional Inherited Members

31.104.1 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.
31.104.2 Constructor & Destructor Documentation

31.104.2.1 Fl_Repeat_Button()

Fl_Repeat_Button::Fl_Repeat_Button ( 
  int X, 
  int Y, 
  int W, 
  int H, 
  const char ∗ l = 0 )

Creates a new Fl_Repeat_Button widget using the given position, size, and label string.

The default boxtype is FL_UP_BOX. Deletes the button.

31.104.3 Member Function Documentation

31.104.3.1 handle()

int Fl_Repeat_Button::handle ( 
  int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

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

### 31.105 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.

```c
#include <Fl_Return_Button.H>
```

Inheritance diagram for Fl_Return_Button:

![Inheritance Diagram](diagram.png)

#### 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.

#### Protected Member Functions

- **void draw** ()
  
  Draws the widget.

#### Additional Inherited Members

31.105.1 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.

![Figure 31.30 Fl_Return_Button](flreturnbutton.png)
31.105 Fl_Return_Button Class Reference

31.105.2 Constructor & Destructor Documentation

31.105.2.1 Fl_Return_Button()

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.

31.105.3 Member Function Documentation

31.105.3.1 draw()

void Fl_Return_Button::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget’s draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Button.

31.105.3.2 handle()

int Fl_Return_Button::handle (  
  int event ) [virtual]

Handles the specified event.

You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.
## Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></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_Button`.

The documentation for this class was generated from the following files:

- `Fl_Return_Button.H`
- `Fl_Return_Button.cxx`

### 31.106 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_RGB_Image
       `-- Fl_Image
           `-- Fl_RGB_Image
               `-- Fl_BMP_Image
               `-- Fl_JPEG_Image
               `-- Fl_PNG_Image
               `-- Fl_PNM_Image
               `-- Fl_SVG_Image
```

### Public Member Functions

- **virtual `Fl_SVG_Image` * as_svg_image ()**
  
  Returns whether an image is an `Fl_SVG_Image` or not.
- **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 (int W, int H)**
  
  Creates a resized copy of the specified image.
- **`Fl_Image` * copy ()**
  
  The `copy()` method converts an image to grayscale.
virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
   Draws the image to the current drawing surface with a bounding box.
virtual void draw (int X, int Y)

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.
Fl_RGB_Image (const Fl_Pixmap ∗pxm, Fl_Color bg=FL_GRAY)
The constructor creates a new RGBA image from the specified Fl_Pixmap.

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 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 normalize ()
   Makes sure the object is fully initialized.
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.

Static Public Member Functions

static void max_size (size_t size)
   Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
static size_t max_size ()
   Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Public Attributes

int alloc_array
   If non-zero, the object’s data array is delete[]d when deleting the object.
const uchar ∗array
   Points to the start of the object’s data array.

Friends

class Fl_Graphics_Driver

Additional Inherited Members

31.106.1 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.
31.106.2 Constructor & Destructor Documentation

31.106.2.1 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 \( W \times H \times D \) 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 \( W \times D \) is assumed, otherwise LD must be greater than or equal to \( W \times D \) 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)
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.

Member Function Documentation

as_svg_image()

virtual Fl_SVG_Image* Fl_RGB_Image::as_svg_image ( ) [inline], [virtual]

Returns whether an image is an Fl_SVG_Image or not.

This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of Fl_SVG_Image or NULL if not.

Reimplemented in Fl_SVG_Image.

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.

Reimplemented in Fl_SVG_Image.

copy()

Fl_Image* Fl_RGB_Image::copy (int W, int H) [virtual]

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.
Parameters

\( W, H \)  width and height of the returned copied image

Reimplemented from Fl_Image.

Reimplemented in Fl_SVG_Image.

31.106.3.4 desaturate()

void Fl_RGB_Image::desaturate ( ) [virtual]

The \texttt{desaturate()} method converts an image to grayscale.

If the image contains an alpha channel (depth = 4), the alpha channel is preserved.

An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

Reimplemented from Fl_Image.

Reimplemented in Fl_SVG_Image.

31.106.3.5 draw()

void Fl_RGB_Image::draw ( int \( X, Y, W, H \), int \( cx = 0, cy = 0 \) ) [virtual]

Draws the image to the current drawing surface with a bounding box.

Arguments \( X, Y, W, H \) specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the \( cx \) and \( cy \) arguments.

In other words: \texttt{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, \texttt{fl_pop_clip()} is applied.

Reimplemented from Fl_Image.

Reimplemented in Fl_SVG_Image.
31.106.3.6 label() [1/2]

```cpp
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`.

31.106.3.7 label() [2/2]

```cpp
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`.

31.106.3.8 max_size() [1/2]

```cpp
static void Fl_RGB_Image::max_size (  
    size_t size ) [inline], [static]
```

Sets the maximum allowed image size in bytes when creating an `Fl_RGB_Image` object.

The image size in bytes of an `Fl_RGB_Image` object is the value of the product `w() * h() * d()`. If this product exceeds `size`, the created object of a derived class of `Fl_RGB_Image` won’t be loaded with the image data. This does not apply to direct RGB image creation with `Fl_RGB_Image::Fl_RGB_Image(const uchar * bits, int W, int H, int D, int LD)`. The default `max_size()` value is essentially infinite.

31.106.3.9 max_size() [2/2]

```cpp
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

```cpp
    void Fl_RGB_Image::max_size(size_t)
```
31.106.3.10 normalize()

virtual void Fl_RGB_Image::normalize ( ) [inline], [virtual]

Makes sure the object is fully initialized.

In particular, makes sure member variable array is non-null.

Reimplemented in Fl_SVG_Image.

31.106.3.11 uncache()

void Fl_RGB_Image::uncache ( ) [virtual]

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented from Fl_Image.

31.106.4 Member Data Documentation

31.106.4.1 array

const uchar* Fl_RGB_Image::array

Points to the start of the object's data array.

See also

   class Fl_SVG_Image which delays initialization of this member variable.

The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

Generated by Doxygen
The `Fl_Roller` widget is a "dolly" control commonly used to move 3D objects.

```c
#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.

### Protected Member Functions

- **void draw ()**

  Draws the widget.

### Additional Inherited Members

#### 31.107.1 Detailed Description

The `Fl_Roller` widget is a "dolly" control commonly used to move 3D objects.

The roller can be controlled by clicking and dragging the mouse, by the corresponding arrow keys when the roller has the keyboard focus, or by the mousewheels when the mouse pointer is positioned over the roller widget.
31.107.2 Constructor & Destructor Documentation

31.107.2.1 Fl_Roller()

Fl_Roller::Fl_Roller (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * L = 0 )

Creates a new Fl_Roller widget using the given position, size, and label string.

The default boxtype is FL_NO_BOX.

Inherited destructor destroys the valuator.

31.107.3 Member Function Documentation

31.107.3.1 draw()

void Fl_Roller::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

31.107.3.2 handle()

int Fl_Roller::handle (  
  int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

The documentation for this class was generated from the following files:

- Fl_Roller.H
- Fl_Roller.cxx

31.108 Fl_Round_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

#include <Fl_Round_Button.H>

Inheritance diagram for Fl_Round_Button:

```
Fl_Widget
  ↓
Fl_Button
  ↓
Fl_Light_Button
  ↓
Fl_Round_Button
  ↓
Fl_Radio_Round_Button
```

Public Member Functions

- **Fl_Round_Button** (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Round_Button widget using the given position, size, and label string.
Additional Inherited Members

31.108.1 Detailed Description

Buttons generate callbacks when they are clicked by the user.

You control exactly when and how by changing the values for `type()` and `when()`.

The `Fl_Round_Button` subclass 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`.

31.108.2 Constructor & Destructor Documentation

31.108.2.1 Fl_Round_Button()

```
Fl_Round_Button::Fl_Round_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ L = 0 )
```

Creates a new `Fl_Round_Button` widget using the given position, size, and label string.

The `Fl_Round_Button` subclass displays the "ON" state by turning on a light, rather than drawing pushed in.

The default box type is `FL_NO_BOX`, which draws the label w/o a box right of the checkmark.

The shape of the "light" is set with `down_box()` and its default value is `FL_ROUND_DOWN_BOX`.

The color of the light when on is controlled with `selection_color()`, which defaults to `FL_FOREGROUND_COLOR` (usually black).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>
A clock widget of type FL_ROUND_CLOCK.

#include <Fl_Round_Clock.H>

Inheritance diagram for Fl_Round_Clock:

```
Fl_Widget
  ▼
Fl_Clock_Output
    ▼
Fl_Clock
      ▼
Fl_Round_Clock
```

Public Member Functions

- **Fl_Round_Clock** (int X, int Y, int W, int H, const char *L=0)

  *Creates the clock widget, setting his type and box.*

Additional Inherited Members

31.109.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.

Has no box.

31.109.2 Constructor & Destructor Documentation
31.109.2.1  Fl_Round_Clock()

Fl_Round_Clock::Fl_Round_Clock (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ L = 0 )

Creates the clock widget, setting his type and box.

Create an Fl_Round_Clock widget using the given position, size, and label string.

The clock type is FL_ROUND_CLOCK and the boxtype is FL_NO_BOX.

This constructor is the same as Fl_Clock(FL_ROUND_CLOCK, X, Y, W, H, L).

See also

    Fl_Clock(uchar, int, int, int, int, const char ∗)

Parameters

| in X, Y, W, H | position and size of the widget |
| in L | widget label, default is no label |

The documentation for this class was generated from the following files:

- Fl_Round_Clock.H
- Fl_Clock.cxx

31.110  Fl_Scroll Class Reference

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

#include <Fl_Scroll.H>

Inheritance diagram for Fl_Scroll:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Scroll
```

Classes

- struct ScrollInfo

Structure to manage scrollbar and widget interior sizes.
Public Types

• enum {
  HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4,
  HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }

Public Member Functions

• void clear ()
  Clear all but the scrollbars...

• Fl_Scroll (int X, int Y, int W, int H, const char ∗=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 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.

Generated by Doxygen
Additional Inherited Members

31.110.1 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 31.34 Fl_Scroll](image)

If all of the child widgets are packed together into a solid rectangle then you want to set box() to FL_NO_BOX or one of the _FRAME types. This will result in the best output. However, if the child widgets are a sparse arrangement you must set box() to a real _BOX type. This can result in some blinking during redrawing, but that can be solved by using a Fl_Double_Window.

By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll. Use Fl_Scroll::type() to change this as follows:

- 0 - No scrollbars
- Fl_Scroll::HORIZONTAL - Only a horizontal scrollbar.
- Fl_Scroll::VERTICAL - Only a vertical scrollbar.
- Fl_Scroll::BOTH - The default is both scrollbars.
- Fl_Scroll::HORIZONTAL_ALWAYS - Horizontal scrollbar always on, vertical always off.
- Fl_Scroll::VERTICAL_ALWAYS - Vertical scrollbar always on, horizontal always off.
- Fl_Scroll::BOTH_ALWAYS - Both always on.

Use scrollbar.align(int) (see void Fl_Widget::align(Fl_Align)) to change what side the scrollbars are drawn on.

If the FL_ALIGN_LEFT bit is on, the vertical scrollbar is on the left. If the FL_ALIGN_TOP bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.

This widget can also be used to pan around a single child widget “canvas”. This child widget should be of your own class, with a draw() method that draws the contents. The scrolling is done by changing the x() and y() of the widget, so this child must use the x() and y() to position its drawing. To speed up drawing it should test fl_not_clipped(int x, int y, int w, int h) to find out if a particular area of the widget must be drawn.

Another very useful child is a single Fl_Pack, which is itself a group that packs its children together and changes size to surround them. Filling the Fl_Pack with Fl_Tabs groups (and then putting normal widgets inside those) gives you a very powerful scrolling list of individually-opening 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.
31.110.2 Constructor & Destructor Documentation

31.110.2.1 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.

31.110.3 Member Function Documentation

31.110.3.1 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).

31.110.3.2 clear()

void Fl_Scroll::clear ( )

Clear all but the scrollbars...
31.110.3.3 draw()

```c
void Fl_Scroll::draw () [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget’s draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```c
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

31.110.3.4 handle()

```c
int Fl_Scroll::handle (int event) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>if the event was not used or understood</th>
</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_Group`. 

---

Generated by Doxygen
31.110.3.5  recalc_scrollbars()

    void Fl_Scroll::recalc_scrollbars ( ScrollInfo & si ) [protected]

Calculate visibility/size/position of scrollbars, find children's bounding box.

The \texttt{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 \texttt{resize()} handling.

Parameters

\begin{verbatim}
  in  si  - ScrollInfo structure
\end{verbatim}

Returns

Structure containing the calculated info.

31.110.3.6  resize()

    void Fl_Scroll::resize ( int X, int Y, int W, int H ) [virtual]

Resizes the \texttt{Fl_Scroll} widget and moves its children if necessary.

The \texttt{Fl_Scroll} widget first resizes itself, and then it moves all its children if (and only if) the \texttt{Fl_Scroll} widget has been moved. The children are moved by the same amount as the \texttt{Fl_Scroll} widget has been moved, hence all children keep their relative positions.

Note

\texttt{Fl_Scroll::resize()} does not call \texttt{Fl_Group::resize()}, and child widgets are not resized.

Since children of an \texttt{Fl_Scroll} are not resized, the \texttt{resizable()} widget is ignored (if it is set).

The scrollbars are moved to their proper positions, as given by \texttt{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 \texttt{Fl_Scroll} is drawn the next time. This may change in a future release.

See also

\texttt{Fl_Group::resizable()}
\texttt{Fl_Widget::resize(int,int,int,int)}

Reimplemented from \texttt{Fl_Group}.  

---

Generated by Doxygen
31.110.3.7 scroll_to()

```cpp
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:

```cpp
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,"b2"); // relative (x,y) = (50,50)
Fl_Box b3 ( 60,110,50,"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.

31.110.3.8 scrollbar_size() [1/2]

```cpp
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)

31.110.3.9 scrollbar_size() [2/2]

```cpp
void Fl_Scroll::scrollbar_size (  
    int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.

Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.
Parameters

| in | newSize | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size() |

See also

Fl::scrollbar_size()

31.110.3.10 xposition()

int Fl_Scroll::xposition ( ) const [inline]

Gets the current horizontal scrolling position.

31.110.3.11 yposition()

int Fl_Scroll::yposition ( ) const [inline]

Gets the current vertical scrolling position.

The documentation for this class was generated from the following files:

- Fl_Scroll.H
- Fl_Scroll.cxx

31.111 Fl_Scrollbar Class Reference

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar.

#include <Fl_Scrollbar.H>

Inheritance diagram for Fl_Scrollbar:

```
Fl_Widget
    ↓
Fl_Valuator
    ↓
Fl_Slider
    ↓
Fl_Scrollbar
```
Public Member Functions

- **Fl_Scrollbar** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Scrollbar widget with given position, size, and label.

- **int handle** (int)
  
  Handles the specified event.

- **int linesize** () const
  
  Get the size of step, in lines, that the arrow keys move.

- **void linesize** (int i)
  
  This number controls how big the steps are that the arrow keys do.

- **int value** () const
  
  Gets the integer value (position) of the slider in the scrollbar.

- **int value** (int p)
  
  Sets the value (position) of the slider in the scrollbar.

- **int value** (int pos, int windowSize, int first_line, int total_lines)
  
  Sets the position, size and range of the slider in the scrollbar.

- **~Fl_Scrollbar** ()
  
  Destroys the Scrollbar.

Protected Member Functions

- **void draw** ()
  
  Draws the widget.

Additional Inherited Members

31.111.1 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.
### Constructor & Destructor Documentation

#### Fl_Scrollbar()

```cpp
Fl_Scrollbar::Fl_Scrollbar ( int X, int Y, int W, int H, const char * L = 0 )
```

Creates a new Fl_Scrollbar widget with given position, size, and label.
You need to do type(FL_HORIZONTAL) if you want a horizontal scrollbar.

#### ~Fl_Scrollbar()

```cpp
Fl_ScrollView::~Fl_ScrollView ( )
```

Destroys the Scrollbar.

### Member Function Documentation

#### draw()

```cpp
void Fl_ScrollView::draw ( ) [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_ScrollView
s->draw(); // calls Fl_ScrollView::draw()
```

Implements Fl_Widget.

#### handle()

```cpp
int Fl_ScrollView::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.

31.111.3.3 linesize()

void Fl_Scrollbar::linesize (  
    int i ) [inline]

This number controls how big the steps are that the arrow keys do.
In addition page up/down move by the size last sent to value() minus one linesize(). The default is 16.

31.111.3.4 value() [1/3]

int Fl_Scrollbar::value ( ) const [inline]

Gets the integer value (position) of the slider in the scrollbar.
You can get the floating point value with Fl_Slider::value().

See also

Fl_Scrollbar::value(int p)
Fl_Scrollbar::value(int pos, int size, int first, int total)

31.111.3.5 value() [2/3]

int Fl_Scrollbar::value (  
    int p ) [inline]

Sets the value (position) of the slider in the scrollbar.

See also

Fl_Scrollbar::value()
Fl_Scrollbar::value(int pos, int size, int first, int total)
31.112 Fl_Scroll::ScrollInfo::Fl_Scrollbar_Data Struct Reference

A local struct to manage a scrollbar's xywh region and tab values.

```c
#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`

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_Scroll.H`
- `Fl_Scrollbar.cxx`

---

31.111.3.6  `value()` [3/3]

```c
int Fl_Scrollbar::value (  
    int pos, 
    int windowSize, 
    int first_line, 
    int total_lines ) [inline]
```

Sets the position, size and range of the slider in the scrollbar.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>pos</th>
<th>position, first line displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>windowSize</td>
<td>number of lines displayed</td>
</tr>
<tr>
<td>in</td>
<td>first_line</td>
<td>number of first line</td>
</tr>
<tr>
<td>in</td>
<td>total_lines</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

---

Generated by Doxygen
31.112.1 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

31.113 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.

#include <Fl_Secret_Input.H>

Inheritance diagram for Fl_Secret_Input:

```
Fl_Widget
 |   
|   Fl_Input_
|   |   Fl_Input
|   |   |   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.

Additional Inherited Members

31.113.1 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.

31.113.2 Constructor & Destructor Documentation
31.113.2.1  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.

31.113.3  Member Function Documentation

31.113.3.1  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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in  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_Input.

The documentation for this class was generated from the following files:

- Fl_Secret_Input.H
- Fl_Input.cxx

Generated by Doxygen
31.114  Fl_Select_Browser Class Reference

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

#include <Fl_Select_Browser.H>

Inheritance diagram for Fl_Select_Browser:

```
Fl_Widget
  |
  v
Fl_Group
  |
  v
Fl_Browser_
  |
  v
Fl_Browser
  |
  v
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.

Additional Inherited Members

31.114.1  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.

31.114.2  Constructor & Destructor Documentation
31.114.2.1 Fl_Select_Browser()

Fl_Select_Browser::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.

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

31.115 Fl_Shared_Image Class Reference

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

#include <Fl_Shared_Image.H>

Inheritance diagram for Fl_Shared_Image:

```
+-----------------+            +-----------------+
| Fl_Shared_Image |  ↓           | Fl_Image         |
|                  |            | Fl_Shared_Image |
```

Public Member Functions

- virtual Fl_Shared_Image ∗ as_shared_image ()
  
  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)
  
  The color_average() method averages the colors in the image with the FLTK color value c.
- virtual Fl_Image ∗ copy (int W, int H)
  
  Creates a resized copy of the specified image.
- Fl_Image ∗ copy ()
- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws the image to the current drawing surface with a bounding box.
- void draw (int X, int Y)
- 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.
- virtual void uncache ()
  If the image has been cached for display, delete the cache data.

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.

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.

Static Protected Member Functions

- static int compare (Fl_Shared_Image **i0, Fl_Shared_Image **i1)
  Compares two shared images.

Protected Attributes

- int alloc_image_
- Fl_Image * image_
- const char * name_
- int original_
- int refcount_
Static Protected Attributes

- static int alloc_handlers_ = 0
- static int alloc_images_ = 0
- static Fl_Shared_Handler *handlers_ = 0
- static Fl_Shared_Image **images_ = 0
- static int num_handlers_ = 0
- static int num_images_ = 0

Friends

- class Fl_Graphics_Driver
- class Fl_JPEG_Image
- class Fl_PNG_Image

Additional Inherited Members

31.115.1 Detailed Description

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

Most applications will also want to link against the fltk_images library and call the fl_register_images() function to support standard image formats such as BMP, GIF, JPEG, PNG, and SVG (unless the library was built with the option removing SVG support).

Images can be requested (loaded) with Fl_Shared_IMAGE::get(), find(), and some other methods. All images are cached in an internal list of shared images and should be released when they are no longer needed. A refcount is used to determine if a released image is to be destroyed with delete.

See also

- Fl_Shared_IMAGE::get()
- Fl_Shared_IMAGE::find()
- Fl_Shared_IMAGE::release()

31.115.2 Constructor & Destructor Documentation

31.115.2.1 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.
**31.115.2.2 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.

**31.115.2.3 ~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.

**31.115.3 Member Function Documentation**

**31.115.3.1 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().

**31.115.3.2 as_shared_image()**

virtual Fl_Shared_Image* Fl_Shared_Image::as_shared_image ( ) [inline], [virtual]

Returns whether an image is an Fl_Shared_Image or not.

This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_\_Image.

Since

1.4.0

Reimplemented from Fl_Image.
31.115.3.3 color_average()

```cpp
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`.

31.115.3.4 compare()

```cpp
int Fl_Shared_Image::compare (
    Fl_Shared_Image ** i0,
    Fl_Shared_Image ** i1 ) [static], [protected]
```

Compares two shared images.

The order of comparison is:

1. Image name, usually the filename used to load it
2. Image width
3. Image height

A special case is considered if the width of one of the images is zero and the other image is marked `original`. In this case the images match, i.e. the comparison returns success (0).

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 usually used in two steps:

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.

Returns

Whether the images match or their relative sort order (see text).
Return values

<table>
<thead>
<tr>
<th></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>

### 31.115.3.5 copy()

```cpp
Fl_Image * Fl_Shared_Image::copy (  
    int W,  
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.

**Parameters**

- **W,H** width and height of the returned copied image

Reimplemented from Fl_Image.

### 31.115.3.6 desaturate()

```cpp
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.

### 31.115.3.7 draw()

```cpp
void Fl_Shared_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```
Draws the image to the current drawing surface with a bounding box.

Arguments \( X, Y, W, H \) specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the \( cx \) and \( cy \) arguments.

In other words: \( \text{fl\_push\_clip}(X,Y,W,H) \) is applied, the image is drawn with its upper-left corner at \( X-cx, Y-cy \) and its own width and height, \( \text{fl\_pop\_clip()} \) is applied.

Reimplemented from \texttt{Fl\_Image}.

### 31.115.3.8 find()

\texttt{Fl\_Shared\_Image * Fl\_Shared\_Image::find (}
\texttt{
  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 \texttt{Fl\_Shared\_Image::release()} when no longer needed.

### 31.115.3.9 get()[1/2]

\texttt{Fl\_Shared\_Image * Fl\_Shared\_Image::get (}
\texttt{
  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 \texttt{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)`

31.115.3.10  get()[2/2]

```cpp
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 <code>Fl_RGB_Image</code> used to build a new shared image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>own_it</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

31.115.3.11  num_images()

```cpp
int Fl_Shared_Image::num_images ( ) [static]
```

Returns the total number of shared images in the array.

31.115.3.12  original()

```cpp
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

31.115.3.13 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.

31.115.3.14 release()

void Fl_Shared_Image::release ( ) [virtual]

Releases and possibly destroys (if refcount <= 0) a shared image.

In the latter case, it will reorganize the shared image array so that no hole will occur.

Reimplemented from Fl_Image.

31.115.3.15 reload()

void Fl_Shared_Image::reload ( )

Reloads the shared image from disk.

31.115.3.16 remove_handler()

void Fl_Shared_Image::remove_handler ( Fl_Shared_Handler f ) [static]

Removes a shared image handler.
31.115.3.17 uncache()

```cpp
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

### 31.116 Fl_Simple.Counter Class Reference

This widget creates a counter with only 2 arrow buttons.

```cpp
#include <Fl_Simple.Counter.H>
```

Inheritance diagram for Fl_Simple.Counter:

```
Fl_Widget
   |
   v
Fl_Valuator
   |
   v
Fl.Counter
   |
   v
Fl_Simple.Counter
```

**Public Member Functions**

- **Fl_Simple.Counter** (int X, int Y, int W, int H, const char *L=0)

**Additional Inherited Members**

### 31.116.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

![Figure 31.36 Fl_Simple.Counter](image)

**Figure 31.36 Fl_Simple.Counter**

The documentation for this class was generated from the following files:

- Fl_Simple.Counter.H
- Fl.Counter.cxx
31.117 Fl_Simple_Terminal Class Reference

This is a continuous text scroll widget for logging and debugging output, much like a terminal.

```cpp
#include <Fl_Simple_Terminal.H>
```

Inheritance diagram for Fl_Simple_Terminal:

```
+--- Fl_Widget
|   +--- Fl_Group
|       +--- Fl_Text_Display
|               +--- Fl_Simple_Terminal
```

Public Member Functions

- `void ansi (bool val)`
  
  Enable/disable support of ANSI sequences like "\033[31m", which sets the color/font/weight/size of any text that follows.

- `bool ansi () const`
  
  Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.

- `void append (const char ∗s, int len=-1)`
  
  Appends new string 's' to terminal.

- `void clear ()`
  
  Clears the terminal's screen and history.

- `void current_style_index (int)`
  
  Set the style table index used as the current drawing color/font/weight/size for new text.

- `int current_style_index () const`
  
  Get the style table index used as the current drawing color/font/weight/size for new text.

- `Fl_Simple_Terminal (int X, int Y , int W, int H, const char ∗l=0)`
  
  Creates a new Fl_Simple_Terminal widget that can be a child of other FLTK widgets.

- `void history_lines (int)`
  
  Sets the maximum number of lines for the terminal history.

- `int history_lines () const`
  
  Get the maximum number of terminal history lines last set by history_lines(int).

- `void normal_style_index (int)`
  
  Sets the style table index used by the ANSI terminal reset sequence "\033[0m", which resets the current drawing color/font/weight/size to "normal".

- `int normal_style_index () const`
  
  Gets the style table index used by the ANSI terminal reset sequence "\033[0m".

- `void printf (const char ∗fmt,...)`
  
  Appends printf formatted messages to the terminal.

- `void remove_lines (int start, int count)`
  
  Remove the specified range of lines from the terminal, starting with line 'start' and removing 'count' lines.

- `void stay_at_bottom (bool)`
  
  Configure the terminal to remain scrolled to the bottom when possible, chasing the end of the buffer whenever new text is added.
- `bool stay_at_bottom () const`
  
  Gets the current value of the `stay_at_bottom(bool)` flag.

- `void style_table (Fl_Text_Display::Style_Table_Entry *stable, int stable_size, int normal_style_index=0)`
  
  Set a user defined style table, which controls the font colors, faces, weights and sizes available for the terminal's text content.

- `const Fl_Text_Display::Style_Table_Entry * style_table () const`
  
  Return the current style table being used.

- `int style_table_size () const`
  
  Return the current style table's size (in bytes).

- `void text (const char *s, int len=-1)`
  
  Replaces the terminal with new text content in string 's'.

- `const char * text () const`
  
  Returns entire text content of the terminal as a single string.

- `void vprintf (const char *fmt, va_list ap)`
  
  Appends printf formatted messages to the terminal.

- `~Fl_Simple_Terminal ()`
  
  Destructor for this widget; removes any internal allocations for the terminal, including text buffer, style buffer, etc.

Protected Member Functions

- `virtual void draw ()`
  
  Draws the widget, including a cursor at the end of the buffer.

- `void enforce_history_lines ()`
  
  Enforce 'history_lines' limit on the history buffer by trimming off lines from the top of the buffer.

- `void enforce_stay_at_bottom ()`
  
  Scroll to last line unless someone has manually scrolled the vertical scrollbar away from the bottom.

- `void vscroll_cb2 (Fl_Widget *, void *)`

Static Protected Member Functions

- `static void vscroll_cb (Fl_Widget *, void *)`

Protected Attributes

- `Fl_Text_Buffer * buf`
- `Fl_Text_Buffer * sbuf`

Additional Inherited Members

31.117.1 Detailed Description

This is a continuous text scroll widget for logging and debugging output, much like a terminal.

Includes `printf()` for appending messages, a line limit for the screen history size, ANSI sequences to control text color, font face, font weight and font size.

This is useful in place of using stdout/stderr for logging messages when no terminal is available, such as when an application is invoked from a desktop shortcut, dock, or file browser.

Like a regular console terminal, the vertical scrollbar 'tracks' the bottom of the buffer as new output is added. If the user scrolls away from the bottom, this 'tracking' feature is temporarily suspended, so the user can browse the terminal history without fighting the scrollbar when new text is added asynchronously. When the user returns the scroller to the bottom of the display, the scrollbar's tracking resumes.

Features include:
• `history_lines(int)` can define a maximum size for the terminal screen history

• `stay_at_bottom(bool)` can be used to cause the terminal to keep scrolled to the bottom

• `ansi(bool)` enables ANSI sequences within the text to control text colors

• `style_table()` can be used to define custom color/font/weight/size combinations

What this widget is NOT is a full terminal emulator; it does NOT handle stdio redirection, pipes, pseudo ttys, termio character cooking, keyboard input processing, screen addressing, random cursor positioning, curses(3) compatibility, or VT100/xterm emulation.

It is a simple text display widget that leverages the features of the `Fl_Text_Display` base class to handle terminal-like behavior, such as logging events or debug information.

Example use:

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

tty = new Fl_Simple_Terminal(...);
tty->ansi(true); // enable use of "\033[3m"
tty->printf("The time is now: \033[32m%s\033[0m", date_time_str);
```

Example application:

```c
#include <time.h> //START
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Simple_Terminal.H>
#define TERMINAL_HEIGHT 120

// Globals
Fl_Double_Window *G_win = 0;
Fl_Box *G_box = 0;
Fl_Simple_Terminal *G_tty = 0;

// Append a date/time message to the terminal every 2 seconds
void tick_cb(void *data) {
    time_t lt = time(NULL);
    G_tty->printf("Timer tick: \033[32m%s\033[0m\n", ctime(&lt));
    Fli::repeat_timeout(2.0, tick_cb, data);
}

int main(int argc, char **argv) {
    G_win = new Fl_Double_Window(500, 200+TERMINAL_HEIGHT, "Your App");
    G_win->begin();
    G_box = new Fl_Box(0, 0, G_win->w(), 200, "Your app GUI in this area.\n\nYour app's debugging output in tty below");
    // Add simple terminal to bottom of app window for scrolling history of status messages.
    G_tty = new Fl_Simple_Terminal(0,200,G_win->w(),TERMINAL_HEIGHT);
    G_tty->ansi(true); // enable use of "\033[3m"
    G_win->end();
    G_win->resizable(G_win);
    G_win->show();
    Fli::add_timeout(0.5, tick_cb);
    return Fl::run();
} //END
```

Generated by Doxygen
Style Tables For Color/Font/Fontsize Control

Internally this widget derives from Fl_Text_Display, and therefore inherits some of its idiosyncracies. In particular, when colors are used, the base class's concept of a 'style table' is used.

The 'style table' is similar to a color mapped image; where each pixel is a single value that is an index into a table of colors to minimize per-pixel memory use.

The style table has a similar goal; since every character in the terminal can potentially be a different color, instead of managing several integer attribute values per-character, a single character for each character is used as an index into the style table, choosing one of the available color/font/weight/size values available. This saves on as much as 3 to 4 times the memory use, useful when there's a large amount of text.

When ansi() is set to 'true', ANSI sequences of the form "\033[#m" can be used to select different colors, font faces, font weights (bold,italic..), and font sizes, where '#' is the index number into the style table. Example:

```
\033[0mThis text uses the 1st entry in the style table
\033[1mThis text uses the 2nd entry in the style table
\033[2mThis text uses the 3rd entry in the style table
```

etc..

There is a built-in style table that provides some commonly used ANSI colors for "\033[30m" through "\033[37m" (blk,red,grn,yel,blu,mag,cyn,wht), and a brighter version of those colors for "\033[40" through "\033[47m". See ansi(bool) for more info.

You can also supply a custom style table using style_table(Style_Table_Entry*,int,int), allowing you to define your own color/font/weight/size combinations. See that method's docs for more info.

All style index numbers are rounded to the size of the style table (via modulus) to protect the style array from overruns.

31.117.2 Member Function Documentation

31.117.2.1 ansi() [1/2]

```cpp
void Fl_Simple_Terminal::ansi (  
   bool val )
```

Enable/disable support of ANSI sequences like "\033[31m", which sets the color/font/weight/size of any text that follows.

If enabled, ANSI sequences of the form "\033[#m" can be used to change font color, face, and size, where '#' is an index number into the current style table. These "escape sequences" are hidden from view.

If disabled, the textcolor() / textfont() / texsize() methods define the color and font for all text in the terminal. ANSI sequences are not handled specially, and rendered as raw text.

A built-in style table is provided, but you can configure a custom style table using style_table(Style_Table_Entry*,int,int) for your own colors and fonts.

The built-in style table supports these ANSI sequences:
ANSI Sequence | Color Name | Font Face + Size | Remarks
------------- | ---------- | --------------- | ------------
"\033[0m" | Normal | FL_COURIER, 14 | Resets to default color/font/weight/size
"\033[30m" | Medium Black | FL_COURIER, 14 |
"\033[31m" | Medium Red | FL_COURIER, 14 |
"\033[32m" | Medium Green | FL_COURIER, 14 |
"\033[33m" | Medium Yellow | FL_COURIER, 14 |
"\033[34m" | Medium Blue | FL_COURIER, 14 |
"\033[35m" | Medium Magenta | FL_COURIER, 14 |
"\033[36m" | Medium Cyan | FL_COURIER, 14 |
"\033[37m" | Medium White | FL_COURIER, 14 | The color when "\033[0m" reset is used
"\033[40m" | Bright Black | FL_COURIER, 14 |
"\033[41m" | Bright Red | FL_COURIER, 14 |
"\033[42m" | Bright Green | FL_COURIER, 14 |
"\033[43m" | Bright Yellow | FL_COURIER, 14 |
"\033[44m" | Bright Blue | FL_COURIER, 14 |
"\033[45m" | Bright Magenta | FL_COURIER, 14 |
"\033[46m" | Bright Cyan | FL_COURIER, 14 |
"\033[47m" | Bright White | FL_COURIER, 14 |

Here’s example code demonstrating the use of ANSI codes to select the built-in colors, and how it looks in the terminal:

```
Figure 31.37 Fl_Simple_Terminal built-in ANSI sequences
```

Note

Changing the `ansi(bool)` value clears the buffer and forces a `redraw()`. Enabling ANSI mode overrides `textfont()`, `textsize()`, `textcolor()` completely, which are controlled instead by `current_style_index()` and the current `style_table()`.

See also

- `style_table(Style_Table_Entry*, int, int)`, `current_style_index()`, `normal_style_index()`

31.117.2.2 ansi()[2/2]

`bool Fl_Simple_Terminal::ansi() const`

Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.

When true, ANSI sequences in the text stream control color, font and font sizes of text (e.g. "\033[41mThis is Red\033[0m"). For more info, see `ansi(bool)`.

See also

- `ansi(bool)`

Generated by Doxygen
### append()

```cpp
void Fl_Simple_Terminal::append (const char * s, int len = -1 )
```

Appends new string 's' to terminal.

The string can contain UTF-8, crlf's, and ANSI sequences are also supported when ansi(bool) is set to 'true'.

**Parameters**

<table>
<thead>
<tr>
<th>s</th>
<th>string to append.</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>optional length of string can be specified if known to save the internals from having to call strlen()</td>
</tr>
</tbody>
</table>

**See also**

printf(), vprintf(), text(), clear()

### clear()

```cpp
void Fl_Simple_Terminal::clear ( )
```

Clears the terminal’s screen and history.

Cursor moves to top of window.

### current_style_index()

```cpp
void Fl_Simple_Terminal::current_style_index ( int val )
```

Set the style table index used as the current drawing color/font/weight/size for new text.

**For example:**

```cpp
; tty->ansi(true);
tty->append("Some normal text.\n");
tty->current_style_index(2); // same as "\033[2m"
tty->append("This text will be green.\n");
tty->current_style_index(tty->normal_style_index()); // same as "\033[0m"
tty->append("Back to normal text.\n");
```

This value can also be changed by an ANSI sequence like "\033[#m", where # would be a new style index value. So if the application executes: `term->append("\033[4mTesting")`, then `current_style_index()` will be left set to 4.

The index number specified should be within the number of items in the current style table. Values larger than the table will be clamped to the size of the table with a modulus operation.

**Effective only when ansi(bool) is 'true'.**
31.117.2.6 current_style_index() [2/2]

```
int Fl_Simple_Terminal::current_style_index ( ) const
```

Get the style table index used as the current drawing color/font/weight/size for new text.
This value is also controlled by the ANSI sequence "\033[#m", where # would be a new style index value. So if the
application executes: term->append("\033[4mTesting"), then current_style_index() returns 4.

See also

```
current_style_index(int)
```

31.117.2.7 draw()

```
void Fl_Simple_Terminal::draw (    void ) [protected], [virtual]
```

Draws the widget, including a cursor at the end of the buffer.
This is needed since currently Fl_Text_Display doesn't provide a reliable way to always do this.
Reimplemented from Fl_Text_Display.

31.117.2.8 enforce_history_lines()

```
void Fl_Simple_Terminal::enforce_history_lines ( ) [protected]
```

Enforce 'history_lines' limit on the history buffer by trimming off lines from the top of the buffer.
This is a protected member called automatically by the public API functions. Only internal methods or subclasses
adjusting the internal buffer directly should need to call this.

31.117.2.9 enforce_stay_at_bottom()

```
void Fl_Simple_Terminal::enforce_stay_at_bottom ( ) [protected]
```

Scroll to last line unless someone has manually scrolled the vertical scrollbar away from the bottom.
This is a protected member called automatically by the public API functions. Only internal methods or subclasses
adjusting the internal buffer directly should need to call this.

31.117.2.10 history_lines() [1/2]

```
void Fl_Simple_Terminal::history_lines (    int maxlines )
```

Sets the maximum number of lines for the terminal history.
The new limit value is automatically enforced on the current screen history, truncating off any lines that exceed the
new limit.
When a limit is set, the buffer is trimmed as new text is appended, ensuring the buffer never displays more than the
specified number of lines.
The default maximum is 500 lines.
Parameters

| **maxlines** | Maximum number of lines kept on the terminal buffer history. Use -1 for an unlimited scroll history. A value of 0 is not recommended. |

31.117.2.11  **history_lines() [2/2]**

```cpp
int Fl_Simple_Terminal::history_lines () const
```

Get the maximum number of terminal history lines last set by `history_lines(int)`.
-1 indicates an unlimited scroll history.

See also

- `history_lines(int)`

31.117.2.12  **normal_style_index() [1/2]**

```cpp
void Fl_Simple_Terminal::normal_style_index ( int val )
```

Sets the style table index used by the ANSI terminal reset sequence "\033[0m", which resets the current drawing color/font/weight/size to "normal".

Effective only when `ansi(bool)` is 'true'.

See also

- `ansi(bool)`, `style_table(Style_Table_Entry∗,int,int)`

Note

Changing this value does not change the current drawing color. To change that, use `current_style_index(int)`.

31.117.2.13  **normal_style_index() [2/2]**

```cpp
int Fl_Simple_Terminal::normal_style_index ( ) const
```

Acts as a simple accessor returning the current normal_style_index().

This is the value last set by `normal_style_index(int)`, or as set by the 3rd argument to `style_table(Style_Table_← Entry∗,int,int)`.

See also

- `normal_style_index(int)`, `ansi(bool)`, `style_table(Style_Table_Entry∗,int,int)`
### 31.117.2.14 printf()

```c
void Fl_Simple_Terminal::printf (const char * fmt, ... )
```

Appends printf formatted messages to the terminal.

The string can contain UTF-8, crlf's, and ANSI sequences are also supported when `ansi(bool)` is set to 'true'.

**Example:**

```c
#include <FL/Fl_Simple_Terminal.H>
int main(...) {
    // Create a simple terminal, and append some messages to it
    Fl_Simple_Terminal *tty = new Fl_Simple_Terminal(...);
    // Append three lines of formatted text to the buffer
    tty->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
    tty->printf("The current PID is %ld.\n", (long)getpid());
}
```

**Note**

See `Fl_Text_Buffer::vprintf()` for limitations.

**Parameters**

| in | fmt | is a printf format string for the message text. |

### 31.117.2.15 remove_lines()

```c
void Fl_Simple_Terminal::remove_lines (int start, int count )
```

Remove the specified range of lines from the terminal, starting with line 'start' and removing 'count' lines.

This method is used to enforce the history limit.

**Parameters**

| start | – starting line to remove |
| count | – number of lines to remove |

### 31.117.2.16 stay_at_bottom()

```c
void Fl_Simple_Terminal::stay_at_bottom ()
```
bool stay_at_bottom()

Configure the terminal to remain scrolled to the bottom when possible, chasing the end of the buffer whenever new
text is added.

If disabled, the terminal behaves more like a text display widget; the scrollbar does not chase the bottom of the
buffer.

If the user scrolls away from the bottom, this 'chasing' feature is temporarily disabled. This prevents the user
from having to fight the scrollbar chasing the end of the buffer while browsing when new text is also being added
asynchronously. When the user returns the scroller to the bottom of the display, the chasing behavior resumes.

The default is 'true'.

31.117.2.17 stay_at_bottom() [2/2]

bool Fl_Simple_Terminal::stay_at_bottom() const

Gets the current value of the stay_at_bottom(bool) flag.

When true, the terminal tries to keep the scrollbar scrolled to the bottom when new text is added.

See also

stay_at_bottom(bool)

31.117.2.18 style_table() [1/2]

void Fl_Simple_Terminal::style_table()

Fl_Text_Display::Style_Table_Entry * stable,
int stable_size,
int normal_style_index = 0 )

Set a user defined style table, which controls the font colors, faces, weights and sizes available for the terminal's
text content.

ansi(bool) must be set to 'true' for the defined style table to be used at all.

If 'stable' is NULL, then the "built in" style table is used. For info about the built-in colors, see ansi(bool).

Which style table entry used for drawing depends on the value last set by current_style_index(), or by the ANSI
sequence "\033[#{\#m", where '#' is the index into the style table array, the index limited to the size of the array via
modulus.

If the index# passed via "\033[#{\#m is larger than the number of elements in the table, the value is clamped via
modulus. So for a 10 element table, the following ANSI codes would all be equivalent, selecting the 5th element in
the table: "\033[5m", "\033[15m", "\033[25m", etc. This is because 5=(15%10)=(25%10), etc.

A special exception is made for "\033[0m", which is supposed to "reset" the current style table to default
color/font/weight/size, as last set by normal_style_index, or by the API method normal_style_index(int).

In cases like the built-in style table, where the 17th item is the "normal" color, the 'normal_style_index' is set to 17
so that "\033[0m" resets to that color, instead of the first element in the table.

If you want "\033[0m" to simply pick the first element in the table, then set 'normal_style_index' to 0.

An example of defining a custom style table (white courier 14, red courier 14, and white helvetica 14):

Generated by Doxygen
```c
int main() {
    // Our custom style table
    Fl_Text_Display::Style_Table_Entry mystyle[] = {
        // Font Color Font Face Font Size Index ANSI Sequence
        // ------------ ------------------------- --------- ----- ------------
        { FL_WHITE, FL_COURIER_BOLD, 14 }, // 0    "\\033[0m" ("default")
        { FL_RED,   FL_COURIER_BOLD, 14 }, // 1    "\\033[1m"
        { FL_WHITE, FL_HELVETICA,   14 }, // 2    "\\033[2m"
    };
    // Create terminal, enable ANSI and our style table
    tty = new Fl_Simple_Terminal(..);
    tty->ansi(true); // enable ANSI codes
    tty->style_table(&mystyle[0], sizeof(mystyle), 0); // use our custom style table
    // Now write to terminal, with ANSI that uses our style table
    tty->printf("\\033[0mNormal Text\\n\n\\033[1mRed Courier Text\\n\n\\033[2mWhite Helvetica\\n\n\n\\033[0mBack to normal.\n\n");

    Note
    Changing the style table clear()s the terminal.
    You currently can't control /background/ color of text, a limitation of Fl_Text_Display's current implementation.
    The caller is responsible for managing the memory of the style table.
    Until STR#3412 is repaired, Fl_Text_Display has scrolling bug if the style table's font size != textsize()
}
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stable</code></td>
<td>- the style table, an array of structs of the type Fl_Text_Display::Style_Table_Entry. Can be NULL to use the default style table (see ansi(bool)).</td>
</tr>
<tr>
<td><code>stable_size</code></td>
<td>- the sizeof() the style table (in bytes). Set this to 0 if 'stable' is NULL.</td>
</tr>
<tr>
<td><code>normal_style_index</code></td>
<td>- the style table index used when the special ANSI sequence &quot;033[0m&quot; is encountered. Normally use 0 so that sequence selects the first item in the table. Only use different values if a different entry in the table should be the default. This value should not be larger than the number of items in the table, or it will be clamped with a modulus operation. This value is ignored if stable is NULL.</td>
</tr>
</tbody>
</table>

### Example

```c
31.117.2.19    style_table() [2/2]

const Fl_Text_Display::Style_Table_Entry * Fl_Simple_Terminal::style_table () const

Return the current style table being used.

This is the value last passed as the 1st argument to style_table(Style_Table_Entry*,int,int). If no style table was defined, the built-in style table is returned.

ansi(bool) must be set to 'true' for the style table to be used at all.

See also

style_table(Style_Table_Entry*,int,int)```
31.117.2.20  style_table_size()

```c
int Fl_Simple_Terminal::style_table_size ( ) const
```

Return the current style table's size (in bytes).
This is the value last passed as the 2nd argument to `style_table(Style_Table_Entry∗,int,int)`.

31.117.2.21  text() [1/2]

```c
void Fl_Simple_Terminal::text ( const char ∗ s, int len = -1 )
```

Replaces the terminal with new text content in string 's'.
The string can contain UTF-8, crlf's, and ANSI sequences are also supported when `ansi(bool)` is set to 'true'.
Old terminal content is completely cleared.

Parameters

<table>
<thead>
<tr>
<th>s</th>
<th>string to append.</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>optional length of string can be specified if known to save the internals from having to call strlen()</td>
</tr>
</tbody>
</table>

See also

- `append()`, `printf()`, `vprintf()`, `clear()`

31.117.2.22  text() [2/2]

```c
const char ∗ Fl_Simple_Terminal::text ( ) const
```

Returns entire text content of the terminal as a single string.
This includes the screen history, as well as the visible onscreen content.

31.117.2.23  vprintf()

```c
void Fl_Simple_Terminal::vprintf ( const char ∗ fmt, va_list ap )
```

Appends printf formatted messages to the terminal.
Subclasses can use this to implement their own `printf()` functionality.
The string can contain UTF-8, crlf's, and ANSI sequences are also supported when `ansi(bool)` is set to 'true'.

Note

The expanded string is currently limited to 1024 characters.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt</td>
<td>is a printf format string for the message text.</td>
</tr>
<tr>
<td>ap</td>
<td>is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Simple_Terminal.H
- Fl_Simple_Terminal.cxx

### 31.118 Fl_Single_Window Class Reference

This is the same as Fl_Window.

```cpp
#include <Fl_Single_Window.H>
```

Inheritance diagram for Fl_Single_Window:

```
Fl_Widget
     |
     v
Fl_Group
     |
     v
Fl_Window
     |
     v
Fl_Single_Window
     |
     v
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.

- **int** **make_current** ()

- **void** **show** ()
  
  Puts the window on the screen.

- **void** **show** (int a, char **b)

#### Additional Inherited Members

### 31.118.1 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.
31.118.2 Member Function Documentation

31.118.2.1 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

`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`

31.119 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`:

```
Fl_Slider
    |      |
    |      |
    |      |
    |      |
```

Generated by Doxygen
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
  Get the dimensions of the moving piece of slider.
- void slider_size (double v)
  Set the dimensions of the moving piece of slider.

Protected Member Functions

- void draw (int, int, int, int)
- void draw ()
  Draws the widget.
- int handle (int, int, int, int, int)

Additional Inherited Members

31.19.1 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.
31.119.2 Constructor & Destructor Documentation

31.119.2.1 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.

31.119.3 Member Function Documentation

31.119.3.1 bounds()

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.
31.119.3.2 draw()

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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

Reimplemented in Fl_Value_Slider.

31.119.3.3 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

- **event** the kind of event received

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.
Reimplemented in Fl_Value_Slider.

### 31.119.3.4 scrollvalue()

```cpp
int Fl_Slider::scrollvalue (  
    int pos,  
    int size,  
    int first,  
    int total )
```

Sets the size and position of the sliding knob in the box.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>pos</th>
<th>position of first line displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of window in lines</td>
</tr>
<tr>
<td>in</td>
<td>first</td>
<td>number of first line</td>
</tr>
<tr>
<td>in</td>
<td>total</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

Returns Fl_Valuator::value(p)

### 31.119.3.5 slider() [1/2]

```cpp
Fl_Boxtype Fl_Slider::slider ( ) const [inline]
```

Gets the slider box type.

### 31.119.3.6 slider() [2/2]

```cpp
void Fl_Slider::slider (  
    Fl_Boxtype c ) [inline]
```

Sets the slider box type.

### 31.119.3.7 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
This widget is a combination of a numerical input widget and repeat buttons.

```
#include <Fl_Spinner.H>
```

Inheritance diagram for Fl_Spinner:

```
Fl_Widget
    ▼
     ▼
Fl_Group
    ▼
Fl_Spinner
```

Classes

- class Fl_Spinner_Input

Public Member Functions

- `void color (Fl_Color v)`
  Sets the background color of the spinner widget's input field.
- `Fl_Color color () const`
  Returns the background color of the spinner widget's input field.
- `Fl_Spinner (int X, int Y, int W, int H, const char ∗L=0)`
  Creates a new Fl_Spinner widget using the given position, size, and label string.
- `const char ∗ format () const`
  Returns the format string for the value.
- `void format (const char ∗f)`
  Sets the format string for the value.
- `int handle (int event)`
  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.
- `void maximum_size (int m)`
  Sets the maximum width of the input field.
- `int maximum_size () const`
  Returns the maximum width of the input field.
- `double minimum () const`
  Gets the minimum value of the widget.
- `void minimum (double m)`
  Sets the minimum value of the widget.
- `void range (double a, double b)`
  Sets the minimum and maximum values for the widget.
- `void resize (int X, int Y, int W, int H)`
Resizes the Fl_Group widget and all of its children.

- void selection_color (Fl_Color val)
  
  Sets the selection color of the spinner widget's input field.

- Fl_Color selection_color () const
  
  Returns the selection color of the spinner widget's input field.

- void step (double s)
  
  Sets or returns the amount to change the value when the user clicks a button.

- double step () const
  
  Gets the amount to change the value when the user clicks a button.

- Fl_Color textcolor () const
  
  Gets the color of the text in the input field.

- void textcolor (Fl_Color c)
  
  Sets the color of the text in the input field.

- Fl_Font textfont () const
  
  Gets the font of the text in the input field.

- void textfont (Fl_Font f)
  
  Sets the font of the text in the input field.

- Fl_Fontsize textsize () const
  
  Gets the size of the text in the input field.

- void textsize (Fl_Fontsize s)
  
  Sets the size of the text in the input field.

- void type (uchar v)
  
  Sets the numeric representation in the input field.

- uchar type () const
  
  Gets the numeric representation in the input field.

- double value () const
  
  Gets the current value of the widget.

- void value (double v)
  
  Sets the current value of the input widget.

- void wrap (int set)
  
  Sets whether the spinner wraps around at upper and lower bounds.

- int wrap () const
  
  Gets the wrap mode of the Fl_Spinner widget.

Protected Attributes

- Fl_Repeat_Button down_button_
- Fl_Spinner_Input input_
- Fl_Repeat_Button up_button_

Additional Inherited Members

31.120.1 Detailed Description

This widget is a combination of a numerical input widget and repeat buttons.

The user can either type into the input area or use the buttons to change the value.
31.120 Fl_Spinner Class Reference

31.120.2 Constructor & Destructor Documentation

31.120.2.1 Fl_Spinner()

Fl_Spinner::Fl_Spinner {
    int X,
    int Y,
    int W,
    int H,
    const char * L = 0
}

Creates a new Fl_Spinner widget using the given position, size, and label string.

The inherited destructor destroys the widget and any value associated with it.

31.120.3 Member Function Documentation

31.120.3.1 format()[1/2]

const char* Fl_Spinner::format () const [inline]

Returns the format string for the value.

31.120.3.2 format()[2/2]

void Fl_Spinner::format {
    const char * f } [inline]

Sets the format string for the value.
31.120.3.3 handle()

```cpp
int Fl_Spinner::handle ( int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**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.

31.120.3.4 maximum() [1/2]

```cpp
double Fl_Spinner::maximum ( ) const [inline]
```

Gets the maximum value of the widget.

31.120.3.5 maximum() [2/2]

```cpp
void Fl_Spinner::maximum ( double m ) [inline]
```

Sets the maximum value of the widget.
31.120.3.6  minimum() [1/2]

double Fl_Spinner::minimum ( ) const [inline]

GETS THE MINIMUM VALUE OF THE WIDGET.

31.120.3.7  minimum() [2/2]

void Fl_Spinner::minimum ( double m ) [inline]

SETS THE MINIMUM VALUE OF THE WIDGET.

31.120.3.8  range()

void Fl_Spinner::range ( double a, double b ) [inline]

SETS THE MINIMUM AND MAXIMUM VALUES FOR THE WIDGET.

31.120.3.9  resize()

void Fl_Spinner::resize ( int X, int Y, int W, int H ) [virtual]

RESIZES THE Fl_Group WIDGET AND ALL OF ITS CHILDREN.

The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

See also

Fl_Group::resizable(Fl_Widget*)
Fl_Group::resizable()
Fl_Widget::resize(int,int,int)

Reimplemented from Fl_Group.
Sets or returns the amount to change the value when the user clicks a button.

Before setting step to a non-integer value, the spinner `type()` should be changed to floating point.

See also

```cpp
    double Fl_Spinner::step() const
```

Gets the amount to change the value when the user clicks a button.

See also

```cpp
    Fl_Spinner::step(double)
```

Gets the color of the text in the input field.

```cpp
    void Fl_Spinner::textcolor ( 
        Fl_Color c ) [inline]
```

Sets the color of the text in the input field.

```cpp
    Fl_Font Fl_Spinner::textfont ( ) const [inline]
```

Gets the font of the text in the input field.
31.120.3.15  textfont() [2/2]

    void Fl_Spinner::textfont (  
            Fl_Font f       ) [inline]

Sets the font of the text in the input field.

31.120.3.16  textSize() [1/2]

    Fl_Fontsize Fl_Spinner::textsize ( ) const      [inline]

Gets the size of the text in the input field.

31.120.3.17  textSize() [2/2]

    void Fl_Spinner::textsize (    
            Fl_Fontsize s       ) [inline]

Sets the size of the text in the input field.

31.120.3.18  type() [1/2]

    void Fl_Spinner::type (  
            uchar v         )

Sets the numeric representation in the input field.

Valid values are FL_INT_INPUT and FL_FLOAT_INPUT. Also changes the format() template. Setting a new spinner  
type via a superclass pointer will not work.

Note

    type() is not a virtual function.

31.120.3.19  type() [2/2]

    uchar Fl_Spinner::type ( ) const       [inline]

Gets the numeric representation in the input field.

See also

    Fl_Spinner::type(uchar)
31.120.3.20  value()  [1/2]

double Fl_Spinner::value() const  [inline]

Gets the current value of the widget.

31.120.3.21  value()  [2/2]

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.

31.120.3.22  wrap()  [1/2]

void Fl_Spinner::wrap (  
  int set )  [inline]

Sets whether the spinner wraps around at upper and lower bounds.
If wrap mode is on the spinner value is set to the minimum() or maximum() if the value exceeds the upper or lower bounds, resp., if it was changed by one of the buttons or the FL_Up or FL_Down keys.
The spinner stops at the upper and lower bounds if wrap mode is off.
The default wrap mode is on for backwards compatibility with FLTK 1.3.x and older versions.

Note

Wrap mode does not apply to the input field if the input value is edited directly as a number. The input value is always clipped to the allowed range as if wrap mode was off when the input field is left (i.e. loses focus).

See also

minimum(), maximum()

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>non-zero sets wrap mode, zero resets wrap mode</th>
</tr>
</thead>
</table>

Since

1.4.0
int Fl_Spinner::wrap ( ) const [inline]

Gets the wrap mode of the Fl_Spinner widget.

See also

  void wrap(int)

Since

  1.4.0

The documentation for this class was generated from the following files:

  • Fl_Spinner.H
  • Fl_Spinner.cxx

Public Member Functions

  • Fl_Spinner_Input (int X, int Y, int W, int H)
  • int handle (int event)

    Handle events of Fl_Spinner's embedded input widget.

Additional Inherited Members
31.121.1 handle()

```cpp
int Fl_Spinner::Fl_Spinner_Input::handle (int event) [virtual]
```

Handles events of `Fl_Spinner`'s embedded input widget.

Works like `Fl_Input::handle()` but ignores `FL_Up` and `FL_Down` keys so they can be handled by the parent widget (`Fl_Spinner`).

Reimplemented from `Fl_Input`.

The documentation for this class was generated from the following files:

- `Fl_Spinner.H`
- `Fl_Spinner.cxx`

31.122 Fl_Surface_Device Class Reference

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

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for `Fl_Surface_Device`:

```
Fl_Surface_Device
  ↓
Fl_Display_Device Fl_Widget_Surface
  ↓
Fl_Copy_Surface Fl_EPS_File_Surface Fl_Image_Surface Fl_Paged_Device Fl_SVG_File_Surface
  ↓
Fl_PostScript_File_Device Fl_Printer
```

Public Member Functions

- `Fl_Graphics_Driver * driver ()`
  
  Returns the graphics driver of this drawing surface.
- `virtual bool is_current ()`
  
  Is this surface the current drawing surface?
- `virtual void set_current (void)`
  
  Make this surface the current drawing surface.
- `virtual ~Fl_Surface_Device ()`
  
  The destructor.

Static Public Member Functions

- `static Fl_Surface_Device * pop_current ()`
  
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- `static void push_current (Fl_Surface_Device *new_current)`
  
  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- `static Fl_Surface_Device * surface ()`
  
  The current drawing surface.
Protected Member Functions

- void **driver** (Fl_Graphics_Driver ∗graphics_driver)
  
  Sets the graphics driver of this drawing surface.

- virtual void **end_current** ()

- Fl_Surface_Device (Fl_Graphics_Driver ∗graphics_driver)
  
  Constructor that sets the graphics driver to use for the created surface.

31.122.1 Detailed Description

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

Any FLTK application has at any time a current drawing surface to which all drawing requests are directed. The current surface is given by Fl_Surface_Device::surface(). When main() begins running, the current drawing surface has been set to the computer's display, an instance of the Fl_Display_Device class.

A drawing surface other than the computer's display, is typically used as follows:

1. Create surface, an object from a particular Fl_Surface_Device derived class (e.g., Fl_Copy_Surface, Fl_Printer).

2. Call Fl_Surface_Device::push_current(surface); to redirect all graphics requests to surface which becomes the new current drawing surface (not necessary with class Fl_Printer because it is done by Fl_Printer::begin_job()).

3. At this point all 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()) operate on the new current drawing surface. Drawing surfaces from Fl_Widget_Surface derived classes allow additional ways to draw to them (e.g., Fl_Printer::print_widget(), Fl_Image_Surface::draw()).

4. After all drawing requests have been performed, redirect graphics requests back to their previous destination with Fl_Surface_Device::pop_current().

5. Delete surface.

For back-compatibility, it is also possible to use the Fl_Surface_Device::set_current() member function to change the current drawing surface, once to the new surface, once to the previous one.

Class Fl_Surface_Device can also be derived to define new kinds of graphical output usable with FLTK drawing functions. An example would be to draw to a PDF file. This would require to create a new class, say PDF_File_Surface, derived from class Fl_Surface_Device, and another new class, say PDF_Graphics_Driver, derived from class Fl_Graphics_Driver. Class PDF_Graphics_Driver should implement all virtual methods of the Fl_Graphics_Driver class to support all FLTK drawing functions and have them draw into PDF files. Alternatively, class PDF_Graphics_Driver could implement only some virtual methods, and only part of the FLTK drawing API would be usable when drawing to PDF files.
31.122.1 Fl_Surface.Device()

Fl_Surface_Device::Fl_Surface_Device (  
    Fl_Graphics_Driver ∗ graphics_driver ) [inline], [protected]
Constructor that sets the graphics driver to use for the created surface.

31.122.2 ~Fl_Surface_Device()

Fl_Surface_Device::~Fl_Surface_Device ( ) [virtual]
The destructor.

31.122.3 Member Function Documentation

31.122.3.1 driver() [1/2]

void Fl_Surface_Device::driver (  
    Fl_Graphics_Driver ∗ graphics_driver ) [inline], [protected]
Sets the graphics driver of this drawing surface.

31.122.3.2 driver() [2/2]

Fl_Graphics_Driver ∗ Fl_Surface_Device::driver ( ) [inline]
Returns the graphics driver of this drawing surface.

31.122.3.3 pop_current()

Fl_Surface_Device ∗ Fl_Surface_Device::pop_current ( ) [static]
Removes the top element from the current drawing surface stack, and makes the new top element current.
Returns
A pointer to the new current drawing surface.

Version
1.4.0
31.122.3.4 push_current()  

void Fl_Surface_Device::push_current (  
    Fl_Surface_Device ∗ new_current ) [static]  

Pushes new_current on top of the stack of current drawing surfaces, and makes it current.  
new_current will receive all future graphics requests.  

Version  
1.4.0

31.122.3.5 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. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().  
Reimplemented in Fl_Printer, Fl_Image_Surface, and Fl_Copy_Surface.

31.122.3.6 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 requests  

The documentation for this class was generated from the following files:  

- Fl_Device.H  
- Fl_Device.cxx

31.123 Fl_SVG_File_Surface Class Reference

A drawing surface producing a Scalable Vector Graphics (SVG) file.  
#include <Fl_SVG_File_Surface.H>  

Incheritance diagram for Fl_SVG_File_Surface:

```
Fl_Surface_Device

  Fl_Widget_Surface

    Fl_SVG_File_Surface
```

Generated by Doxygen
Public Member Functions

- **int close ()**
  
  Closes with function fclose() the FILE pointer where SVG data is output.

- **FILE * file ()**
  
  Returns the underlying FILE pointer.

- **Fl_SVG_File_Surface (int width, int height, FILE *svg)**
  
  Constructor of the SVG drawing surface.

- **virtual void origin (int x, int y)**
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- **virtual int printable_rect (int *w, int *h)**
  
  Computes the width and height of the drawable area of the drawing surface.

- **virtual void translate (int x, int y)**
  
  Translates the current graphics origin accounting for the current rotation.

- **virtual void untranslate ()**
  
  Undoes the effect of a previous translate() call.

- **\~Fl_SVG_File_Surface ()**
  
 Destructor.

Additional Inherited Members

31.123.1 Detailed Description

A drawing surface producing a Scalable Vector Graphics (SVG) file.

This drawing surface allows to store any FLTK graphics in vectorial form in a "Scalable Vector Graphics" file.

Usage example:

```cpp
Fl_Window *win = ... // Window to draw to a .svg file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *svg = fl_fopen("/path/to/mywindow.svg", "w");
if (svg) {
    Fl_SVG_File_Surface *surface = new Fl_SVG_File_Surface(ww, wh, svg);
    Fl_Surface_Device::push_current(surface);
    fl_color(FL_WHITE);
    fl_rectf(0, 0, ww, wh);
    surface->draw_decorated_window(win);
    Fl_Surface_Device::pop_current();
    delete surface; // the .svg file is not complete until the destructor was run
    fclose(svg);
}
```

Note

FLTK uses the PNG and JPEG libraries to encode images to the SVG format. For this reason, class **Fl_SVG_File_Surface** is placed in the fltk_images library. If JPEG is not available at application build time, PNG is enough (but produces a quite larger output). If PNG isn't available either, images don't appear in the SVG output.

31.123.2 Constructor & Destructor Documentation

Generated by Doxygen
31.123.2.1 Fl_SVG_File_Surface()

Fl_SVG_File_Surface::Fl_SVG_File_Surface (  
   int width,  
   int height,  
   FILE * svg )

Constructor of the SVG drawing surface.
### Parameters

<table>
<thead>
<tr>
<th>width, height</th>
<th>Width and height of the graphics area in FLTK drawing units</th>
</tr>
</thead>
<tbody>
<tr>
<td>svg</td>
<td>A writable FILE pointer where the SVG data are to be sent. The resulting SVG data are not complete until after destruction of the Fl_SVG_File_Surface object or after calling close().</td>
</tr>
</tbody>
</table>

#### 31.123.2.2 \(~\text{Fl}_\text{SVG}_\text{File}_\text{Surface}()\)

**Fl_SVG_File_Surface::~Fl_SVG_File_Surface()**

Destructor.

The underlying FILE pointer remains open after destruction of the Fl_SVG_File_Surface object unless close() was called.

#### 31.123.3 Member Function Documentation

##### 31.123.3.1 close()

**int Fl_SVG_File_Surface::close()**

Closes with function fclose() the FILE pointer where SVG data is output.

The only operation possible after this on the Fl_SVG_File_Surface object is its destruction.

**Returns**

The value returned by fclose().

##### 31.123.3.2 origin()

**virtual void Fl_SVG_File_Surface::origin ( int x, int y ) [virtual]**

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, printable_rect(&w, &h); origin(w/2, 0); sets the graphics origin at the top center of the drawable area. Successive origin() calls don’t combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).
Parameters

| in  | x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from Fl_Widget_Surface.

31.123.3.3 printable_rect()

```cpp
virtual int Fl_SVG_File_Surface::printable_rect (
    int * w,
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

31.123.3.4 translate()

```cpp
virtual void Fl_SVG_File_Surface::translate ( 
    int x,
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.

Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.

Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following file:

- Fl_SVG_File_Surface.H
31.124  **Fl_SVG_Image Class Reference**

The **Fl_SVG_Image** class supports loading, caching and drawing of scalable vector graphics (SVG) images.

```cpp
#include <Fl_SVG_Image.H>
```

Inheritance diagram for Fl_SVG_Image:

```
Fl_Image
    Fl_RGB_Image
        Fl_SVG_Image
```

Public Member Functions

- virtual **Fl_SVG_Image** * as_svg_image ()
  
  *Returns whether an image is an Fl_SVG_Image or not.*

- 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 (int W, int H)
  
  *Creates a resized copy of the specified image.*

- **Fl_Image** * copy ()

- virtual void desaturate ()
  
  *The desaturate() method converts an image to grayscale.*

- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  *Draws the image to the current drawing surface with a bounding box.*

- void draw (int X, int Y)

- **Fl_SVG_Image** (const char *filename, const char *svg_data=NULL)
  
  *The constructor loads the SVG image from the given .svg/.svgz filename or in-memory data.*

- virtual void normalize ()
  
  *Makes sure the object is fully initialized.*

- void resize (int width, int height)
  
  *Have the svg data (re-)rasterized using the given width and height values.*

- virtual ~**Fl_SVG_Image** ()
  
  *The destructor frees all memory and server resources that are used by the SVG image.*

Public Attributes

- bool proportional
  
  *Set this to false to allow image re-scaling that alters the image aspect ratio.*

Generated by Doxygen
Additional Inherited Members

31.124.1 Detailed Description

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images.

The FLTK library performs parsing and rasterization of SVG data using a modified version of the nanosvg software ([https://github.com/memononen/nanosvg](https://github.com/memononen/nanosvg)). The software modification allows the option to change the image ratio while performing rasterization.

Use Fl_Image::fail() to check if the Fl_SVG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, and ERR_FORMAT if the SVG format could not be decoded. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Rasterization is not done until the image is first drawn or resize() or normalize() is called. Therefore, array is NULL until then. The delayed rasterization ensures an Fl_SVG_Image is always rasterized to the exact screen resolution at which it is drawn.

The Fl_SVG_Image class draws images computed by nanosvg with the following known limitations

- text between <text> and </text> marks,
- image elements, and
- <use> statements

are not rendered.

The FLTK library can optionally be built without SVG support; in that case, class Fl_SVG_Image is unavailable.

Example of displaying a hard-coded svg file:

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

// A black rotated rectangle
const char *svg_data = "<svg viewBox="0 0 200 200" version = "1.1">\n  <rect x="25" y="50" width="150" height="100" fill="black" transform="rotate(45 100 100)"/>\n</svg>\n";

int main(int argc, char **argv) {
  Fl_SVG_Image *svg = new Fl_SVG_Image(0, svg_data); // create SVG object
  Fl_Window *win = new Fl_Window(720, 486, "svg test");
  Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
  box->image(svg); // assign svg object to Fl_Box
  win->end();
  win->show(argc, argv);
  return(Fl::run());
}
```

Example of displaying an svg image from a file:
```c
#include <errno.h> // errno
#include <string.h> // strerror
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/fl_message.H>

int main(int argc, char **argv) {
    Fl_Window *win = new Fl_Window(720, 486, "svg test");
    Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());

    // Load svg image from disk, assign to a box
    const char *svgpath = "/var/tmp/simple.svg";
    Fl_SVG_Image *svg = new Fl_SVG_Image(svgpath); // load SVG object from disk
    switch (svg->fail()) {
        case Fl_Image::ERR_FILE_ACCESS:
            // File couldn't load? show path + os error to user
            fl_alert("%s: %s", svgpath, strerror(errno));
            return 1;
        case Fl_Image::ERR_FORMAT:
            // Parsing error
            fl_alert("%s: couldn't decode image", svgpath);
            return 1;
    }
    box->image(svg); // assign svg object to box
    win->end();
    win->show(argc, argv);
    return(Fl::run());
}
```

Example of fitting an svg image to a resizable Fl_Box:

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

class resizable_box : public Fl_Box {
public:
    resizable_box(int w, int h) : Fl_Box(0, 0, w, h, NULL) {};
    virtual void resize(int x, int y, int w, int h) {
        // p3 = proportional, p4 = can expand
        Fl_Box::resize(x, y, w, h);
    }
};

int main(int argc, char **argv) {
    Fl_Window *win = new Fl_Window(130, 130);
    resizable_box *box = new resizable_box(win->w(), win->h());
    Fl_SVG_Image *svg = new Fl_SVG_Image("/path/to/image.svg");
    box->image(svg);
    svg->scale(box->w(), box->h());
    win->end();
    win->resizable(win);
    win->show(argc, argv);
    return Fl::run();
}
```

### 31.124.2 Constructor & Destructor Documentation

#### 31.124.2.1 Fl_SVG_Image()

```c
Fl_SVG_Image::Fl_SVG_Image {
    const char * filename,
    const char * svg_data = NULL
}
```

The constructor loads the SVG image from the given .svg/.svgz filename or in-memory data.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of a .svg or .svgz file, or NULL.</td>
</tr>
<tr>
<td>svg_data</td>
<td>A pointer to the memory location of the SVG image data. This parameter allows to load an SVG image from in-memory data, and is used when filename is NULL.</td>
</tr>
</tbody>
</table>

Note

In-memory SVG data is parsed by the object constructor and is not used after construction.

31.124.2.2 \(~\text{Fl}_{-}\text{SVG}_{-}\text{Image}()\)

\text{Fl}_{-}\text{SVG}_{-}\text{Image}::\sim\text{Fl}_{-}\text{SVG}_{-}\text{Image}()\ [\text{virtual}]

The destructor frees all memory and server resources that are used by the SVG image.

31.124.3 Member Function Documentation

31.124.3.1 as_svg_image()

\text{virtual Fl}_{-}\text{SVG}_{-}\text{Image}\ast \text{Fl}_{-}\text{SVG}_{-}\text{Image}::\text{as}_{-}\text{svg}_{-}\text{image}()\ [\text{inline}], [\text{virtual}]

Returns whether an image is an \text{Fl}_{-}\text{SVG}_{-}\text{Image} or not.

This virtual method returns a pointer to the \text{Fl}_{-}\text{SVG}_{-}\text{Image} if this object is an instance of \text{Fl}_{-}\text{SVG}_{-}\text{Image} or NULL if not.

Reimplemented from \text{Fl}_{-}\text{RGB}_{-}\text{Image}.

31.124.3.2 color_average()

\text{void Fl}_{-}\text{SVG}_{-}\text{Image}::\text{color}_{-}\text{average}\ (\text{Fl}_{-}\text{Color}\ c,\ \text{float}\ i)\ [\text{virtual}]

The \text{color}_{-}\text{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 \text{Fl}_{-}\text{RGB}_{-}\text{Image}.

31.124.3.3 copy()

\text{Fl}_{-}\text{Image}\ast \text{Fl}_{-}\text{SVG}_{-}\text{Image}::\text{copy}\ (\text{int}\ W,\ \text{int}\ H)\ [\text{virtual}]

Creates a resized copy of the specified image.

The image should be deleted (or in the case of \text{Fl}_{-}\text{Shared}_{-}\text{Image}, released) when you are done with it.
Parameters

| W, H | width and height of the returned copied image |

Reimplemented from Fl_RGB_Image.

31.124.3.4 desaturate()

```cpp
void Fl_SVG_Image::desaturate () [virtual]
```

The `desaturate()` method converts an image to grayscale.

If the image contains an alpha channel (depth = 4), the alpha channel is preserved.

An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

Reimplemented from Fl_RGB_Image.

31.124.3.5 draw()

```cpp
void Fl_SVG_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box.

Arguments `X, Y, W, H` specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the `cx` and `cy` arguments.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at `X-cx, Y-cy` and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from Fl_RGB_Image.

31.124.3.6 normalize()

```cpp
void Fl_SVG_Image::normalize ( ) [virtual]
```

Makes sure the object is fully initialized.

This function rasterizes the SVG image, and consequently initializes its array member, if that was not done before.

Reimplemented from Fl_RGB_Image.
### 31.124.3.7 resize()

```cpp
void Fl_SVG_Image::resize (  
    int width,  
    int height  
)
```

Have the svg data (re-)rasterized using the given `width` and `height` values.

By default, the resulting image `w()` and `h()` will be close to `width` and `height` while preserving the width/height ratio of the SVG data. If `proportional` was set to `false`, the image is rasterized to the exact `width` and `height` values. In both cases, `data_w()` and `data_h()` values are set to `w()` and `h()`, respectively.

### 31.124.4 Member Data Documentation

#### 31.124.4.1 proportional

```cpp
bool Fl_SVG_Image::proportional
```

Set this to `false` to allow image re-scaling that alters the image aspect ratio.

Upon object creation, proportional is set to `true`, and the aspect ratio is kept constant.

The documentation for this class was generated from the following files:

- FL_SVG_Image.H
- FL_SVG_Image.cxx

### 31.125 Fl_Sys_Menu_Bar Class Reference

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

```cpp
#include <Fl_Sys_Menu_Bar.H>
```

Inheritance diagram for Fl_Sys_Menu_Bar:
Public Types

- enum window_menu_style_enum { no_window_menu = 0, tabbing_mode_none, tabbing_mode_automatic, tabbing_mode_preferred }

  Possible styles of the Window menu in the system menu bar.

Public Member Functions

- int add (const char *label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)

  Add a new menu item to the system menu bar.

- 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 *str)

  Forms-compatible procedure to add items to the system menu bar.

- void clear ()

  Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.

- int clear_submenu (int index)

  Clears the specified submenu pointed to by index of all menu items.

- Fl_Sys_Menu_Bar (int x, int y, int w, int h, const char *l=0)

  The constructor.

- int insert (int index, const char *label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)

  Insert in the system menu bar a new menu item

- 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.

- 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

- void mode (int i, int fl)

  Sets the flags of item i.

- int mode (int i) const

  Gets 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.

- virtual void update ()

  Updates the menu bar after any change to its items.

- virtual ~Fl_Sys_Menu_Bar ()

  The destructor.
Static Public Member Functions

- static void about (Fl_Callback ∗ cb, void ∗ data)
  
  Attaches a callback to the "About myprog" item of the system application menu.

- static void create_window_menu ()
  
  Adds a Window menu, to the end of the system menu bar.

- static window_menu_style_enum window_menu_style ()
  
  Get the style of the Window menu in the system menu bar.

- static void window_menu_style (window_menu_style_enum style)
  
  Set the desired style of the Window menu in the system menu bar.

Protected Member Functions

- virtual void draw ()
  
  Draws the widget.

Additional Inherited Members

31.125.1 Detailed Description

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

On other than macOS platforms, Fl_Sys_Menu_Bar is a synonym of class Fl_Menu_Bar.

On the macOS platform, replace Fl_Menu_Bar with Fl_Sys_Menu_Bar, and a system menu at the top of the screen will be available. This menu will match an array of Fl_Menu_Item's exactly as in all other FLTK menus (except for the submenu with the application's own name and the 'Window' menu; see below). There is, though, an important difference between an Fl_Sys_Menu_Bar object under macOS and under other platforms: only a single object from this class can be created, because macOS uses a single system menu bar. Therefore, porting to macOS an app that creates, on other platforms, several Fl_Menu_Bar objects, one for each of several windows, is more complex than just replacing Fl_Menu_Bar by Fl_Sys_Menu_Bar.

On the macOS platform, the system menu bar of any FLTK app begins with the Application menu which the FLTK library automatically constructs. Functions Fl_Mac_App_Menu::custom_application_menu_items() and Fl_Sys__Menu_Bar::about() can be used to further customize the Application menu. The FLTK library also automatically constructs and handles a Window menu which can be further customized (or even removed) calling Fl_Sys_Menu__Bar::window_menu_style(window_menu_style_enum style). Other member functions of this class allow the app to generate the rest of the system menu bar. It is recommended to localize the system menu bar using the standard Mac OS X localization procedure (see Internationalization).

Changes to the menu state are immediately visible in the menubar when they are made using member functions of the Fl_Sys_Menu_Bar class. Other changes (e.g., by a call to Fl_Menu_Item::set()) should be followed by a call to update() to be visible in the menubar across all platforms. macOS global variable fl_sys_menu_bar points to the unique, current system menu bar.

A few FLTK menu features are not supported by the Mac System menu:

- no symbolic labels
- no embossed labels
- no font sizes
As described above, the submenu with the application's own name (usually the second submenu from the left, immediately following the "Apple" submenu) is a special case, and can be managed with `Fl_Mac_App_Menu::custom_application_menu_items()`. For example, to make your own "Appname - Preferences" dialog, you might use:

```cpp
#include <FL/platform.H> // for Fl_Mac_App_Menu class
#include <FL/Fl_Sys_Menu_Bar.H> // for Fl_Menu_Item

void prefs_cb(Fl_Widget *w, void *data) {
    // ..Open your preferences dialog here..
}

int main(...) {
    // Items to add to the application menu
    static Fl_Menu_Item appitems[] = {
        { "Preferences", 0, prefs_cb, 0, 0 },
        { 0 }, { 0 }
    };
    Fl_Mac_App_Menu::custom_application_menu_items(appitems);
    // adds it
}
```

..the result being:

![Figure 31.40 Mac Application submenu](image)

31.125.2 Member Enumeration Documentation

31.125.2.1 window_menu_style_enum

```cpp
enum Fl_Sys_Menu_Bar::window_menu_style_enum
```

Possible styles of the Window menu in the system menu bar.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_window_menu</td>
<td>No Window menu in the system menu bar.</td>
</tr>
<tr>
<td>tabbing_mode_none</td>
<td>No tabbed windows, but the system menu bar contains a Window menu.</td>
</tr>
<tr>
<td>tabbing_mode_automatically</td>
<td>Windows are created by themselves but can be tabbed later.</td>
</tr>
<tr>
<td>tabbing_mode_preferred</td>
<td>Windows are tabbed when created.</td>
</tr>
</tbody>
</table>
31.125.3 Constructor & Destructor Documentation

31.125.3.1 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().

31.125.4 Member Function Documentation

31.125.4.1 about()

void Fl_Sys_Menu_Bar::about (  
   Fl_Callback * cb,  
   void * data ) [static]

Attaches a callback to the "About myprog" item of the system application menu.

This cross-platform function is effective only under the MacOS platform.

Parameters

<table>
<thead>
<tr>
<th>cb</th>
<th>a callback that will be called by &quot;About myprog&quot; menu item with NULL 1st argument.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>a pointer transmitted as 2nd argument to the callback.</td>
</tr>
</tbody>
</table>

31.125.4.2 add() [1/3]

int Fl_Sys_Menu_Bar::add (  
   const char * label,  
   int shortcut,  
   Fl_Callback * cb,  
   void * user_data = 0,  
   int flags = 0 )

Add a new menu item to the system menu bar.
Add to the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.
Parameters

<table>
<thead>
<tr>
<th>label</th>
<th>new menu item's label</th>
</tr>
</thead>
<tbody>
<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)

31.125.4.3 add() [2/3]

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)

31.125.4.4 add() [3/3]

int Fl_Sys_Menu_Bar::add (const char* str)

Forms-compatible procedure to add items to the system menu bar.

Returns

the index into the menu() array, where the entry was added

See also

Fl_Menu_::add(const char* str)
31.125.4.5  clear()

void Fl_Sys_Menu_Bar::clear ( )

Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.

See also
    Fl_Menu::clear()

31.125.4.6  clear_submenu()

int Fl_Sys_Menu_Bar::clear_submenu ( int index )

Clears the specified submenu pointed to by index of all menu items.

See also
    Fl_Menu::clear_submenu(int index)

31.125.4.7  create_window_menu()

void Fl_Sys_Menu_Bar::create_window_menu ( ) [static]

Adds a Window menu, to the end of the system menu bar.

FLTK apps typically don't need to call this function which is automatically called by the library the first time a window is shown. The default system menu bar contains a Window menu with a "Merge All Windows" item. Other Window menu styles can be obtained calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum) before the first Fl_Window::show(). Alternatively, an app can call create_window_menu() after having populated the system menu bar, for example with menu(const Fl_Menu_Item *), and before the first Fl_Window::show().

This function does nothing on non MacOS platforms.

Version
    1.4
31.125.4.8 draw()

```
void Fl_Sys_Menu_Bar::draw ( )  [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Menu_Bar.

31.125.4.9 insert() [1/2]

```
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)
```
31.125.4.10 insert()

```cpp
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

```cpp
Fl_Menu_::insert(int index, const char* label, const char* shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
```

31.125.4.11 menu()

```cpp
void Fl_Sys_Menu_Bar::menu (  
    const Fl_Menu_Item * m )
```

create a system menu bar using the given list of menu structs

Author
Matthias Melcher

Parameters

| m | Zero-ending list of Fl_Menu_Item's |

31.125.4.12 mode()

```cpp
void Fl_Sys_Menu_Bar::mode (  
    int i,  
    int fl )
```

Sets the flags of item i.

See also

```cpp
Fl_Menu_::mode(int i, int fl)
```
31.125.4.13  remove()

```cpp
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 |

31.125.4.14  replace()

```cpp
void Fl_Sys_Menu_Bar::replace (  
    int index,  
    const char * name  
)
```
rename an item from the system menu bar

**Parameters**

| index | the index of the item to rename |
| name  | the new item name as a UTF8 string |

31.125.4.15  setonly()

```cpp
void Fl_Sys_Menu_Bar::setonly (  
    Fl_Menu_Item * item  
)
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

31.125.4.16  update()

```cpp
void Fl_Sys_Menu_Bar::update ( ) [virtual]
```

Updates the menu bar after any change to its items.

This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.

Reimplemented from Fl_Menu_Bar.
void Fl_Sys_Menu_Bar::window_menu_style (Fl_Sys_Menu_Bar::window_menu_style_enum style) [static]

Set the desired style of the Window menu in the system menu bar.

This function, to be called before the first call to Fl_Window::show(), allows to control whether the system menu bar should contain a Window menu, and if yes, whether new windows should be displayed in tabbed form. These are the effects of various values for style:

- `no_window_menu`: don't add a Window menu to the system menu bar
- `tabbing_mode_none`: add a simple Window menu to the system menu bar
- `tabbing_mode_automatic`: the window menu also contains "Merge All Windows" to group all windows in a single tabbed display mode. This is the default Window menu style for FLTK apps.
- `tabbing_mode_preferred`: new windows are displayed in tabbed mode when first created

The Window menu, if present, is entirely created and controlled by the FLTK library. Mac OS version 10.12 or later must be running for windows to be displayed in tabbed form. Under non MacOS platforms, this function does nothing.

Version
1.4

The documentation for this class was generated from the following files:

- `Fl_Sys_Menu_Bar.H`
- `Fl_Sys_Menu_Bar.cxx`

31.126 Fl_Table Class Reference

A table of widgets or other content.

#include <Fl_Table.H>

Inheritance diagram for Fl_Table:

```
Fl_Widget
  Fl_Group
    Fl_Table
      Fl_Table_Row
```
Public Types

- enum TableContext {
  CONTEXT_NONE = 0, CONTEXT_STARTPAGE = 0x01, CONTEXT_ENDPAGE = 0x02, CONTEXT_ROWR_HEADER = 0x04,
  CONTEXT_COL_HEADER = 0x08, CONTEXT_CELL = 0x10, CONTEXT_TABLE = 0x20, CONTEXT_RC_RESIZE = 0x40
}

The context bit flags for Fl_Table related callbacks.

Public Member Functions

- void add (Fl_Widget &wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table’s group.
- void add (Fl_Widget *wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table’s group.
- Fl_Widget *const *array ()
  Returns a pointer to the array of children.
- 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.
- void col_header_color (Fl_Color val)
  Sets the color for column headers and redraws the table.
- Fl_Color col_header_color ()
  Gets the color for column headers.
- void col_header_height (int height)
  Sets the color for column headers.
- int col_header_height ()
  Gets the height in pixels for column headers and redraws the table.
- int col_position ()
  Gets the column header height.
- void col_position (int col)
  Sets the horizontal scroll position so ‘col’ is at the left, and causes the screen to redraw.
- int col_position ()
  Returns the current column scroll position as a column number.
- int col_resize ()
  Returns if column resizing by the user is allowed.
• void col_resize (int flag)
  Allows/disallows column resizing by the user.

• void col_resize_min (int val)
  Sets the current column minimum resize value.

• 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.

• virtual void cols (int val)
  Set the number of columns in the table and redraw.

• void end ()

• void find (const Fl_Widget *wgt) const

• int find (const Fl_Widget &wgt) const

• Fl_Table (int X, int Y, int W, int H, const char *l=0)
  The constructor for Fl_Table.

• void get_selection (int &row_top, int &col_left, int &row_bot, int &col_right)
  Gets the region of cells selected (highlighted).

• void init_sizes ()
  Resets the internal array of widget sizes and positions.

• void insert (Fl_Widget &wgt, int n)
  The specified widget is removed from its current group (if any) and inserted into the Fl_Table's group at position 'n'.

• void insert (Fl_Widget &wgt, Fl_Widget *w2)
  The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'.

• 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.

• void move_cursor (int R, int C, int shiftselect)
  Moves the selection cursor a relative number of rows/columns specified by R/C.

• void move_cursor (int R, int C)
  Same as move_cursor(R,C,1);.

• void remove (Fl_Widget &wgt)
  The specified widget is removed from Fl_Table's group.

• void resize (int X, int Y, int W, int H)
  Handle resize events if user resizes parent window.

• int row_header ()
  Returns if row headers are enabled or not.

• void row_header (int flag)
  Enables/disables showing the row headers.

• void row_header_color (Fl_Color val)
  Sets the row header color and causes the screen to redraw.

• Fl_Color row_header_color ()
  Returns the current row header color.
void row_header_width (int width)
Sets the row header width to n and causes the screen to redraw.

int row_header_width ()
Returns the current row header width (in pixels).

void row_height (int row, int height)
Sets the height of the specified row in pixels, and the table is redrawn.

int row_height (int row)
Returns the current height of the specified row as a value in pixels.

void row_height_all (int height)
Convenience method to set the height of all rows to the same value, in pixels.

void row_position (int row)
Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.

int row_position ()
Returns the current row scroll position as a row number.

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.

virtual void rows (int val)
Sets the number of rows in the table, and the table is redrawn.

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).

void tab_cell_nav (int val)
Flag to control if Tab navigates table cells or not.

int tab_cell_nav () const
Get state of table's 'Tab' key cell navigation flag.

void table_box (Fl_Boxtype val)
Sets the kind of box drawn around the data table, the default being FL_NO_BOX.

Fl_Boxtype table_box (void)
Returns the current box type used for the data table.

void top_row (int row)
Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

int top_row ()
Returns the current top row shown in the table.

void visible_cells (int &r1, int &r2, int &c1, int &c2)
Returns the range of row and column numbers for all visible and partially visible cells in the table.

void when (Fl_When flags)
The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

~Fl_Table ()
The destructor for Fl_Table.
Protected Types

- **enum ResizeFlag {**
  - RESIZE_NONE = 0,
  - RESIZE_COL_LEFT = 1,
  - RESIZE_COL_RIGHT = 2,
  - RESIZE_ROW ABOVE = 3,
  - RESIZE_ROW BELOW = 4 }

Protected Member Functions

- void change_cursor (Fl_Cursor newcursor)
  - Change mouse cursor to different type.
- long col_scroll_position (int col)
  - Returns the scroll position (in pixels) of the specified column 'col'.
- TableContext cursor2rowcol (int &R, int &C, ResizeFlag &resizeflag)
  - Find row/col for the recent mouse event.
- void damage_zone (int r1, int c1, int r2, int c2, int r3=0, int c3=0)
  - Sets the damage zone to the specified row/col values.
- void draw ()
  - Draws the entire Fl_Table.
- virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)
  - Subclass should override this method to handle drawing the cells.
- int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
  - Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.
- void get_bounds (TableContext context, int &X, int &Y, int &W, int &H)
  - Returns the (X,Y,W,H) bounding region for the specified 'context'.
- int handle (int e)
  - Handle FLTK events.
- int is_fltk_container ()
  - Does the table contain any child fltk widgets?
- void recalc_dimensions ()
  - Recalculate the dimensions of the table, and affect any children.
- void redraw_range (int topRow, int botRow, int leftCol, int rightCol)
  - Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).
- int row_col_clamp (TableContext context, int &R, int &C)
  - Return specified row/col values R and C to within the table's current row/col limits.
- long row_scroll_position (int row)
  - Returns the scroll position (in pixels) of the specified 'row'.
- void table_resized ()
  - Call this if table was resized, to recalculate internal data.
- void table_scrolled ()
  - Recalculate internals after a scroll.

Static Protected Member Functions

- static void scroll_cb (Fl_Widget *, void *)
  - Callback for when someone moves a scrollbar.
Protected Attributes

- **int botrow**
  bottom row# of currently visible table on screen
- **int current_col**
  selection cursor's current column (-1 if none)
- **int current_row**
  selection cursor's current row (-1 if none)
- **Fl_Scrollbar ∗ hscrollbar**
  child horizontal scrollbar widget
- **int leftcol**
  left column# of currently visible table on screen
- **int leftcol_scrollpos**
  precomputed scroll position for left column
- **int rightcol**
  right column# of currently visible table on screen
- **int select_col**
  extended selection column (-1 if none)
- **int select_row**
  extended selection row (-1 if none)
- **Fl_Scroll ∗ table**
  child Fl_Scroll widget container for child ftk widgets (if any)
- **int table_h**
  table's virtual height (in pixels)
- **int table_w**
  table's virtual width (in pixels)
- **int tih**
  Data table's inner h dimension, inside bounding box. See Table Dimension Diagram.
- **int tiw**
  Data table's inner w dimension, inside bounding box. See Table Dimension Diagram.
- **int tix**
  Data table's inner x dimension, inside bounding box. See Table Dimension Diagram.
- **int tiy**
  Data table's inner y dimension, inside bounding box. See Table Dimension Diagram.
- **int toh**
  Data table's outer h dimension, outside bounding box. See Table Dimension Diagram.
- **int torow**
  top row# of currently visible table on screen
- **int toprow_scrollpos**
  precomputed scroll position for top row
- **int tow**
  Data table's outer w dimension, outside bounding box. See Table Dimension Diagram.
- **int tox**
  Data table's outer x dimension, outside bounding box. See Table Dimension Diagram.
- **int toy**
  Data table's outer y dimension, outside bounding box. See Table Dimension Diagram.
- **Fl_Scrollbar ∗ vscrollbar**
  child vertical scrollbar widget
- **int wih**
  Table widget's inner h dimension, inside bounding box. See Table Dimension Diagram.
- **int wiw**
Table widget's inner \emph{w} dimension, inside bounding box. See Table Dimension Diagram.

- \texttt{int wix}
  
  Table widget's inner \emph{x} dimension, inside bounding box. See Table Dimension Diagram.

- \texttt{int wiy}
  
  Table widget's inner \emph{y} dimension, inside bounding box. See Table Dimension Diagram.

Additional Inherited Members

31.126.1 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 \textit{not} handle the data in the table. The \texttt{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. \textit{Not} recommended for large tables.

Figure 31.41 table-simple example

Figure 31.42 table-as-container example
When acting as part of a custom widget, events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the setting for `Fl_Table::when()`.

When acting as a container for FLTK widgets, the FLTK widgets maintain themselves. Although the `draw_cell()` method must be overridden, its contents can be very simple. See the `draw_cell()` code in examples/table-simple.cxx.

The following variables are available to classes deriving from `Fl_Table`:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x/y/w/h</td>
<td><code>Fl_Table</code> widget's outer dimension. The outer edge of the border of the <code>Fl_Table</code>. (Red in the diagram above)</td>
</tr>
<tr>
<td>wix/wiy/wiw/wih</td>
<td><code>Fl_Table</code> widget's inner dimension. The inner edge of the border of the <code>Fl_Table</code>. eg. if the <code>Fl_Table</code>'s <code>box()</code> is <code>FL_NO_BOX</code>, these values are the same as x/y/w/h. (Yellow in the diagram above)</td>
</tr>
<tr>
<td>tox/toy/tow/toh</td>
<td>The table's outer dimension. The outer edge of the border around the cells, but inside the row/col headings and scrollbars. (Green in the diagram above)</td>
</tr>
<tr>
<td>tix/tiy/tiw/tih</td>
<td>The table's inner dimension. The inner edge of the border around the cells, but inside the row/col headings and scrollbars. AKA the table's clip region. eg. if the <code>table_box()</code> is <code>FL_NO_BOX</code>, these values are the same as tox/toy/tow/toh. (Blue in the diagram above)</td>
</tr>
</tbody>
</table>

### CORE DEVELOPERS

- Greg Ercolano : 12/16/2002 - initial implementation 12/16/02. `Fl_Table`, `Fl_Table_Row`, docs.
- Jean-Marc Lienher : 02/22/2004 - added keyboard nav + mouse selection, and ported `Fl_Table` into fltk-utf8-1.1.4

### OTHER CONTRIBUTORS

- Inspired by the Feb 2000 version of FLVW's Flvw_Table widget. Mucho thanks to those folks.
- Mister Satan : 04/07/2003 - MinGW porting mods, and singleinput.cxx; a cool `Fl_Input` oriented spreadsheet example
• Marek Paliwoda : 01/08/2003 - Porting mods for Borland
• Ori Berger : 03/16/2006 - Optimizations for >500k rows/cols

LICENSE

Greg kindly gave his permission to integrate Fl_Table and Fl_Table_Row into FLTK, allowing FLTK license to apply while his widgets are part of the library. [updated by Greg, 04/26/17]

31.126.2 Member Enumeration Documentation

31.126.2.1 TableContext

enum Fl_Table::TableContext

The context bit flags for Fl_Table related callbacks.

Should be used in draw_cell() to determine what's being drawn, or in a callback() to determine where a recent event occurred.

<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 the table is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ENDPAGE</td>
<td>after the table is redrawn</td>
</tr>
<tr>
<td>CONTEXT_ROW_HEADER</td>
<td>drawing or event occurred in the row header</td>
</tr>
<tr>
<td>CONTEXT_COL_HEADER</td>
<td>drawing or event occurred in the col header</td>
</tr>
<tr>
<td>CONTEXT_CELL</td>
<td>drawing or event occurred in a cell</td>
</tr>
<tr>
<td>CONTEXT_TABLE</td>
<td>drawing or event occurred in a dead zone of table</td>
</tr>
<tr>
<td>CONTEXT_RC_RESIZE</td>
<td>column or row is being resized</td>
</tr>
</tbody>
</table>

31.126.3 Constructor & Destructor Documentation

31.126.3.1 Fl_Table()

Fl_Table::Fl_Table ( 
  int X,
  int Y,
  int W,
  int H,
  const char * l = 0 )

The constructor for Fl_Table.

This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.
31.126.3.2 \texttt{\sim Fl\_Table()}

\texttt{Fl\_Table::\sim Fl\_Table()}

The destructor for \texttt{Fl\_Table}.

Destroys the table and its associated widgets.

31.126.4 Member Function Documentation

31.126.4.1 \texttt{array()}

\texttt{Fl\_Widget* const\* Fl\_Table::array() [inline]}

Returns a pointer to the array of children.

\textit{This pointer is only valid until the next time a child is added or removed.}

31.126.4.2 \texttt{callback()}

\texttt{void Fl\_Table::callback (}
\begin{verbatim}
    Fl\_Widget *,
    void *
\end{verbatim}

Callbacks will be called depending on the setting of \texttt{Fl\_Widget::when()}. Callback functions should use the following functions to determine the context/row/column:

- \texttt{Fl\_Table::callback\_row()} returns current row
- \texttt{Fl\_Table::callback\_col()} returns current column
- \texttt{Fl\_Table::callback\_context()} returns current table context

\texttt{callback\_row()} and \texttt{callback\_col()} will be set to the row and column number the event occurred on. If someone clicked on a row header, \texttt{col} will be 0. If someone clicked on a column header, \texttt{row} will be 0.

\texttt{callback\_context()} will return one of the following:

| \texttt{Fl\_Table::CONTEXT\_ROW\_HEADER} | Someone clicked on a row header. Excludes resizing. |
| \texttt{Fl\_Table::CONTEXT\_COL\_HEADER} | Someone clicked on a column header. Excludes resizing. |
| \texttt{Fl\_Table::CONTEXT\_CELL} | Someone clicked on a cell. To receive callbacks for FL\_RELEASE events, you must set when(FL\_WHEN\_RELEASE). |
| \texttt{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
  MyTable() {
[..]
    table.callback(event_callback, (void*)this); // setup callback
    table.when(FL_WHEN_CHANGED|FL_WHEN_RELEASE); // when to call it
  }
};

31.126.4.3  callback_col()

int Fl_Table::callback_col ( ) [inline]

Returns the current column the event occurred on.

This function should only be used from within the user's callback function.

31.126.4.4  callback_context()

TableContext Fl_Table::callback_context ( ) [inline]

Returns the current 'table context'.

This function should only be used from within the user's callback function.

31.126.4.5  callback_row()

int Fl_Table::callback_row ( ) [inline]

Returns the current row the event occurred on.

This function should only be used from within the user's callback function.
31.126.4.6 child()

Fl_Widget* Fl_Table::child (int n) const [inline]

Returns the child widget by an index.

When using the Fl_Table as a container for FLTK widgets, this method returns the widget pointer from the internal array of widgets in the container.

Typically used in loops, eg:

```cpp
for ( int i=0; i<children(); i++ ) {
    Fl_Widget *w = child(i);
    ...
}
```

31.126.4.7 children()

int Fl_Table::children () const [inline]

Returns the number of children in the table.

When using the Fl_Table as a container for FLTK widgets, this method returns how many child widgets the table has.

See also

child(int)

31.126.4.8 clear()

virtual void Fl_Table::clear () [inline], [virtual]

Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert/etc.

See also

rows(int), cols(int)

Reimplemented in Fl_Table_Row.
### col_header()

```cpp
void Fl_Table::col_header ( int flag ) [inline]
```

Enable or disable column headers.

If changed, the table is redrawn.

### col_resize()

```cpp
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.

### col_resize_min()

```cpp
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 $\geq 1$.

### col_width()

```cpp
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`.

### col_width_all()

```cpp
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.
31.126.4.14 cursor2rowcol()

```cpp
Fl_Table::TableContext Fl_Table::cursor2rowcol (  
  int & R,  
  int & C,  
  ResizeFlag & resizeflag ) [protected]
```

Find row/col for the recent mouse event.

Returns the context, and the row/column values in R/C. Also returns 'resizeflag' if mouse is hovered over a resize boundary.

31.126.4.15 damage_zone()

```cpp
void Fl_Table::damage_zone (  
  int r1,  
  int c1,  
  int r2,  
  int c2,  
  int r3 = 0,  
  int c3 = 0 ) [protected]
```

Sets the damage zone to the specified row/col values.

Calls `redraw_range()`.

31.126.4.16 do_callback()

```cpp
void Fl_Table::do_callback (  
  TableContext context,  
  int row,  
  int col ) [inline]
```

Calls the widget callback.

Saves the specified 'context', 'row', and 'col' values, so that the user's callback can then access them with the member functions `callback_context()`, `callback_row()` and `callback_col()`.

31.126.4.17 draw()

```cpp
void Fl_Table::draw (  
  void ) [protected], [virtual]
```

Draws the entire `Fl_Table`.

Lets `fltk` widgets draw themselves first, followed by the cells via calls to `draw_cell()`.

Reimplemented from `Fl_Group`. 

---

Generated by Doxygen
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 visibleCells()) |
| 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'.
// 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) {  // Fl_Table telling us it’s starting to draw
        case CONTEXT_STARTPAGE:  // Fl_Table telling us it’s starting to draw
            fl_font(Fl_HELVETICA, 16);
            return;
        case CONTEXT_ROW_HEADER: // Fl_Table telling us to draw row/col
            case CONTEXT_COL_HEADER:
                fl_push_clip(X, Y, W, H);
                { fl_draw_box(FL_THIN_UP_BOX, X, Y, W, H,
                    color());
                    fl_color(Fl_BLACK);
                    fl_draw(s, X, Y, W, H, FL_ALIGN_CENTER);
                }
                fl_pop_clip();
                return;
        case CONTEXT_CELL:  // Fl_Table telling us to draw cells
            fl_push_clip(X, Y, W, H);
            {  // BG COLOR
                fl_color(row_selected(R) ? selection_color() : FL_WHITE);
                fl_rectf(X, Y, W, H);
                // TEXT
                fl_color(Fl_BLACK);
                fl_draw(s, X, Y, W, H, FL_ALIGN_CENTER);
                // BORDER
                fl_color(Fl_LIGHT2);
                fl_rect(X, Y, W, H);
            }
            fl_pop_clip();
            return;
        default:
            return;
    }
    //NOTREACHED
}

31.126.4.19 find_cell()

int Fl_Table::find_cell (TableContext context,
    int R,
    int C,
    int & X,
    int & Y,
    int & W,
    int & H) [protected]

Find a cell’s X/Y/W/H region for the specified cell in row ‘R’, column ‘C’.

Returns

- 0 – on success, XYWH returns the region of the specified cell.
- -1 – if R or C are out of range, and X/Y/W/H will be set to zero.
31.126.4.20 get_selection()

```cpp
void Fl_Table::get_selection ( 
    int & row_top, 
    int & col_left, 
    int & row_bot, 
    int & col_right )
```

Gets the region of cells selected (highlighted).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_top</td>
<td>Returns the top row of selection area</td>
</tr>
<tr>
<td>col_left</td>
<td>Returns the left column of selection area</td>
</tr>
<tr>
<td>row_bot</td>
<td>Returns the bottom row of selection area</td>
</tr>
<tr>
<td>col_right</td>
<td>Returns the right column of selection area</td>
</tr>
</tbody>
</table>

31.126.4.21 init_sizes()

```cpp
void Fl_Table::init_sizes ( ) [inline]
```

Resets the internal array of widget sizes and positions.

**See also**

- `Fl_Group::init_sizes()`

31.126.4.22 insert()

```cpp
void Fl_Table::insert ( 
    Fl_Widget & wgt, 
    Fl_Widget * w2 ) [inline]
```

The specified widget is removed from its current group (if any) and inserted into `Fl_Table`'s group before widget 'w2'.

This will append if 'w2' is not in `Fl_Table`'s group.

31.126.4.23 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()`.
### 31.126.4.24 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.

### 31.126.4.25 move_cursor()

```cpp
int Fl_Table::move_cursor ( int R, int C, int shiftselect )
```

Moves the selection cursor a relative number of rows/columns specified by R/C.

- R/C can be positive or negative, depending on the direction to move. A value of 0 for R or C prevents cursor movement on that axis.

- If shiftselect is set, the selection range is extended to the new cursor position. If clear, the cursor is simply moved, and any previous selection is cancelled.

- Used mainly by keyboard events (e.g. Fl_Right, FL_Home, FL_End..) to let the user keyboard navigate the selection cursor around.

- The scroll positions may be modified if the selection cursor traverses into cells off the screen's edge.

**Examples:**
- R=1, C=0  -- moves the selection cursor one row downward.
- R=5, C=0  -- moves the selection cursor 5 rows downward.
- R=-5, C=0 -- moves the cursor 5 rows upward.
- R=2, C=2  -- moves the cursor 2 rows down and 2 columns to the right.

### 31.126.4.26 recalc_dimensions()

```cpp
void Fl_Table::recalc_dimensions ( ) [protected]
```

Recalculate the dimensions of the table, and affect any children.

- Internally, Fl_Group::resize() and init_sizes() are called.
31.126.4.27 redraw_range()

void Fl_Table::redraw_range (  
    int topRow,  
    int botRow,  
    int leftCol,  
    int rightCol ) [inline], [protected]

Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).

Extends any previously defined range to redraw.

31.126.4.28 resize()

void Fl_Table::resize (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]

Handle resize events if user resizes parent window.

This changes the size of Fl_Table, causing it to redraw.

Reimplemented from Fl_Group.

31.126.4.29 row_col_clamp()

int Fl_Table::row_col_clamp (  
    TableContext context,  
    int & R,  
    int & C ) [protected]

Return specified row/col values R and C to within the table's current row/col limits.

Returns

0 if no changes were made, or 1 if they were.

31.126.4.30 row_header()

void Fl_Table::row_header (  
    int flag ) [inline]

Enables/disables showing the row headers.

1=enabled, 0=disabled. If changed, the table is redrawn.
31.126.4.31 row_height()

void Fl_Table::row_height (  
   int row,  
   int height )

Sets the height of the specified row in pixels, and the table is redrawn.

callback() will be invoked with CONTEXT_RC_RESIZE if the row's height was actually changed, and when() is FL_WHEN_CHANGED.

31.126.4.32 row_height_all()

void Fl_Table::row_height_all (  
   int height ) [inline]

Convenience method to set the height of all rows to the same value, in pixels.

The screen is redrawn.

31.126.4.33 row_resize()

void Fl_Table::row_resize (  
   int flag ) [inline]

Allows/disallows row resizing by the user.

1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the row headers, row_header() must also be enabled to allow resizing.

31.126.4.34 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.

31.126.4.35 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)
scrollbar_size() [2/2]

```cpp
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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>newSize</code></td>
<td>Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global <code>Fl::scrollbar_size()</code></td>
</tr>
</tbody>
</table>

**See also**

`Fl::scrollbar_size()`

---

set_selection()

```cpp
void Fl_Table::set_selection (  
    int row_top,  
    int col_left,  
    int row_bot,  
    int col_right )
```

Sets the region of cells to be selected (highlighted).

So for instance, `set_selection(0,0,0,0)` selects the top/left cell in the table. And `set_selection(0,0,1,1)` selects the four cells in rows 0 and 1, column 0 and 1.

To deselect all cells, use `set_selection(-1,-1,-1,-1);`

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>row_top</code></td>
<td>Top row of selection area</td>
</tr>
<tr>
<td>in</td>
<td><code>col_left</code></td>
<td>Left column of selection area</td>
</tr>
<tr>
<td>in</td>
<td><code>row_bot</code></td>
<td>Bottom row of selection area</td>
</tr>
<tr>
<td>in</td>
<td><code>col_right</code></td>
<td>Right column of selection area</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If val is 1, Tab key navigates cells in table, not fltk widgets.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If val is 0, Tab key will advance focus to the next fltk widget (default), and does not navigate cells in table.</td>
</tr>
</tbody>
</table>

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)

void Fl_Table::table_box ( Fl_Boxtype val ) [inline]

Sets the kind of box drawn around the data table, the default being FL_NO_BOX.

Changing this value will cause the table to redraw.

void Fl_Table::table_resized ( ) [protected]

Call this if table was resized, to recalculate internal data.

Calls recall_dimensions(), and recalculates scrollbar sizes.
31.126.4.42  table_scrolled()

void Fl_Table::table_scrolled ( ) [protected]
Recalculate internals after a scroll.
Call this if table has been scrolled or resized. Does not handle redraw(). TODO: Assumes t[x|y|w|h] has already been recalculated.

31.126.4.43  top_row()[1/2]

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.

31.126.4.44  top_row()[2/2]

int Fl_Table::top_row ( ) [inline]
Returns the current top row shown in the table.
This row may be partially obscured.

31.126.4.45  visible_cells()

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.

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

31.126.4.46  when()

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:
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 (ie. `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.cxx`

### 31.127 Fl_Table_Row Class Reference

A table with row selection capabilities.

```c
#include <Fl_Table_Row.H>
```

Inheritance diagram for `Fl_Table_Row`:

```
Fl_Widget
   ↓
Fl_Group
      ↓
Fl_Table
         ↓
Fl_Table_Row
```

**Public Types**

- `enum TableRowSelectMode { SELECT_NONE, SELECT_SINGLE, SELECT_MULTI }`

**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 *l=0)`
  
  The constructor for the `Fl_Table_Row`.

- `int row_selected (int row)`
  
  Checks to see if 'row' is selected.

- `void rows (int val)`
  
  Sets the number of rows in the table, and the table is redrawn.
• int rows ()
• 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'.
• void type (TableRowSelectMode val)
   
   Sets the table selection mode.
• TableRowSelectMode type () const
• ~Fl_Table_Row ()
   
   The destructor for the Fl_Table_Row.

Protected Member Functions

• int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
• int handle (int event)
   
   Handle FLTK events.

Additional Inherited Members

31.127.1 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().

31.127.2 Constructor & Destructor Documentation

31.127.2.1 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.
31.127.2.2 \texttt{\textasciitilde Fl\_Table\_Row()} \\

\texttt{Fl\_Table\_Row::\textasciitilde Fl\_Table\_Row ( ) [inline]}

The destructor for the \texttt{Fl\_Table\_Row}.

Destroys the table and its associated widgets.

### 31.127.3 Member Function Documentation

#### 31.127.3.1 clear()

\texttt{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

- \texttt{rows(int), cols(int)}

Reimplemented from \texttt{Fl\_Table}.

#### 31.127.3.2 row_selected()

\texttt{int Fl\_Table\_Row::row\_selected ( 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)

#### 31.127.3.3 select\_all\_rows()

\texttt{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.
31.127.3.4 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.

31.127.3.5 type()

```cpp
void Fl_Table_Row::type (
    TableRowSelectMode val )
```

Sets the table selection mode.

- Fl_Table_Row::SELECT_NONE - No selection allowed
- Fl_Table_Row::SELECT_SINGLE - Only single rows can be selected
- Fl_Table_Row::SELECT_MULTI - Multiple rows can be selected

The documentation for this class was generated from the following files:

- Fl_Table_Row.H
- Fl_Table_Row.cxx

31.128 Fl_Tabs Class Reference

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.

```cpp
#include <Fl_Tabs.H>
```

Inheritance diagram for Fl_Tabs:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Tabs
```
Public Member Functions

- void **client_area** (int &rx, int &ry, int &rw, int &rh, int tabh=0)
  
  Returns the position and size available to be used by its children.

- **Fl_Tabs** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Tabs widget using the given position, size, and label string.

- int **handle** (int)
  
  Handles the specified event.

- **Fl_Widget * push () const**
  
  Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.

- int push (Fl_Widget *)
  
  This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.

- void **tab_align** (Fl_Align a)
  
  Sets the tab label alignment.

- **Fl_Align tab_align () const**
  
  Gets the tab label alignment.

- **Fl_Widget * value ()**
  
  Gets the currently visible widget/tab.

- int value (Fl_Widget *)
  
  Sets the widget to become the current visible widget/tab.

- virtual **Fl_Widget * which (int event_x, int event_y)**
  
  Return the widget of the tab the user clicked on at event_x/event_y.

Protected Member Functions

- virtual void **clear_tab_positions ()**
  
- void **draw ()**
  
  Draws the widget.

- virtual void **draw_tab (int x1, int x2, int W, int H, Fl_Widget *o, int sel=0)**
  
- virtual void **redraw_tabs ()**
  
- virtual int **tab_height ()**
  
- virtual int **tab_positions ()**

Protected Attributes

- **Fl_Align tab_align_**
  
- int **tab_count**
  
- int * **tab_pos**
  
- int * **tab_width**
Additional Inherited Members

31.128.1 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 31.44 Fl_Tabs](image)

Clicking the tab makes a child visible() by calling show() on it, and all other children are made invisible by calling hide() on them. Usually the children are Fl_Group widgets containing several widgets themselves.

Each child makes a card, and its label() is printed on the card tab, including the label font and style. The selection color of that child is used to color the tab, while the color of the child determines the background color of the pane. ‘&’ in labels are used to prefix a shortcut that is drawn underlined and that activates the corresponding tab; repeated ‘&&’ avoids that.

The size of the tabs is controlled by the bounding box of the children (there should be some space between the children and the edge of the Fl_Tabs), and the tabs may be placed "inverted" on the bottom - this is determined by which gap is larger. It is easiest to lay this out in fluid, using the fluid browser to select each child group and resize them until the tabs look the way you want them to.

The background area behind and to the right of the tabs is "transparent", exposing the background detail of the parent. The value of Fl_Tabs::box() does not affect this area. So if Fl_Tabs is resized by itself without the parent, force the appropriate parent (visible behind the tabs) to redraw() to prevent artifacts.

See "Resizing Caveats" below on how to keep tab heights constant. See "Callback's Use Of when()" on how to control the details of how clicks invoke the callback().

A typical use of the Fl_Tabs widget:

```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();
}
```

Generated by Doxygen
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 31.45 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++
void MyTabCallback(Fl_Widget *, void*) {
    Fl_Tabs *tabs = (Fl_Tabs*)w;
    // When tab changed, make sure it has same color as its group
    tabs->selection_color( (tab->value())->color() );
}
```

The result of the above looks like:

![Figure 31.46 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.

```c++
void MyTabCallback(Fl_Widget *w, void*) {
    Fl_Tabs *tabs = (Fl_Tabs*)w;
    // When tab changed, make sure it has same color as its group
    tabs->selection_color( (tab->value())->color() );
}
```

```c++
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 31.47 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.

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:

1. The above flags can be logically OR-ed (|) or added (+) to combine behaviors.
2. The default value for \texttt{when()} is \texttt{FL\_WHEN\_RELEASE} (inherited from \texttt{Fl\_Widget}).

3. If \texttt{FL\_WHEN\_RELEASE} is the \textit{only} flag specified, the behavior will be as if \texttt{(FL\_WHEN\_RELEASE|FL\_WHEN\_CHANGED)} was specified.

4. The value of \texttt{changed()} will be valid during the callback.

5. If both \texttt{FL\_WHEN\_CHANGED} and \texttt{FL\_WHEN\_NOT\_CHANGED} are specified, the callback is invoked whether the tab has been changed or not. The \texttt{changed()} method can be used to determine the cause.

6. \texttt{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.

### 31.128.2 Constructor & Destructor Documentation

#### 31.128.2.1 \texttt{Fl\_Tabs()}

\texttt{Fl\_Tabs::Fl\_Tabs ( int X, int Y, int W, int H, const char * L = 0 )}

Creates a new \texttt{Fl\_Tabs} widget using the given position, size, and label string.

The default boxtype is \texttt{FL\_THIN\_UP\_BOX}.

Use \texttt{add(Fl\_Widget\*)} to add each child, which are usually \texttt{Fl\_Group} widgets. The children should be sized to stay away from the top or bottom edge of the \texttt{Fl\_Tabs} widget, which is where the tabs will be drawn.

All children of \texttt{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 \texttt{Fl\_Tabs} is set to "resizable()", the riders will not resize when the tabs are resized.

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. A kludge has been done so the \texttt{Fl\_Tabs} and all of its children can be automatic (local) variables, but you must declare the \texttt{Fl\_Tabs} widget \textit{first} so that it is destroyed last.

### 31.128.3 Member Function Documentation
31.128.3.1 client_area()

```c
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><code>tabh</code></th>
<th>position and optional height of tabs (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>rx, ry, rw, rh</code></td>
<td>(x,y,w,h) of client area for children</td>
</tr>
</tbody>
</table>

Since

FLTK 1.3.0

31.128.3.2 draw()

```c
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:
Reimplemented from Fl_Group.

31.128.3.3 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**

<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_Group.

31.128.3.4 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*).
31.128.3.5 push() [2/2]

```
int Fl_Tabs::push (
    Fl_Widget * o )
```

This is called by the tab widget's `handle()` method to set the tab group widget the user last FL_PUSH'ed on.
Set back to zero on FL_RELEASE.

As of this writing, the value is mainly used by `draw_tab()` to determine whether or not to draw a 'down' box for the tab when it's clicked, and to turn it off if the user drags off it.

See also

    push().

31.128.3.6 tab_align() [1/2]

```
void Fl_Tabs::tab_align ( 
    Fl_Align a ) [inline]
```

Sets the tab label alignment.

The default is FL_ALIGN_CENTER so tab labels are centered, but since the label space is measured (per label) to fit the labels, there wouldn't be any difference if labels were aligned left or right.

If you want to show an image (icon) next to the group's label you can set a different label alignment. FL_ALIGN_IMAGE_NEXT_TO_TEXT is the recommended alignment to show the icon left of the text.

31.128.3.7 tab_align() [2/2]

```
Fl_Align Fl_Tabs::tab_align ( ) const [inline]
```

Gets the tab label alignment.

See also

    tab_align(Fl_Align)

31.128.3.8 value() [1/2]

```
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.
31.128.3.9  value() [2/2]

    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)

31.128.3.10  which()

    Fl_Widget * Fl_Tabs::which ( int event_x, int event_y ) [virtual]

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

31.129  Fl_Text_Buffer Class Reference

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

#include <Fl_Text_Buffer.H>

Generated by Doxygen
Public Member Functions

- `void add_modify_callback (Fl_Text_Modify_Callback bufModifiedCB, void *cbArg)`
  Adds a callback function that is called whenever the text buffer is modified.

- `void add_predelete_callback (Fl_Text_Predelete_Callback bufPredelCB, void *cbArg)`
  Adds a callback routine to be called before text is deleted from the buffer.

- `const char * address (int pos) const`
  Converts a byte offset in buffer into a memory address.

- `char * address (int pos)`
  Converts a byte offset in buffer into a memory address.

- `void append (const char * t)`
  Appends the text string to the end of the buffer.

- `int appendfile (const char * file, int buflen=128 *1024)`
  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 ()`
  Calls all modify callbacks that have been registered using the `add_modify_callback()` method.

- `void call_predelete_callbacks ()`
  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

- `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.

- `void highlight (int start, int end)`
  Highlights the specified text within the buffer.

- `int highlight ()`
  Returns a non-zero value if text has been highlighted, 0 otherwise.

- `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 the named file at position pos.
Inserts a file at the specified position.

- **bool is_word_separator (int pos) const**
  
  Returns whether character at position *pos* is a word separator.

- **int length () const**
  
  Returns the number of bytes in the buffer.

- **int line_end (int pos) const**
  
  Finds and returns the position of the end of the line containing position *pos* (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).

- **int line_start (int pos) const**
  
  Returns the position of the start of the line containing position *pos*.

- **char * line_text (int pos) const**
  
  Returns the text from the entire line containing the specified character position.

- **int loadfile (const char *file, int buflen=128 *1024)**
  
  Loads a text file into the buffer.

- **int next_char (int ix) const**
  
  Returns the index of the next character.

- **int next_char_clipped (int ix) const**

- **int outputfile (const char *file, int start, int end, int buflen=128 *1024)**
  
  Writes the specified portions of the text buffer to a file.

- **int prev_char (int ix) const**
  
  Returns the index of the previous character.

- **int prev_char_clipped (int ix) const**

- **const Fl_Text_Selection * primary_selection () const**
  
  Returns the primary selection.

- **Fl_Text_Selection * primary_selection ()**
  
  Returns the primary selection.

- **void printf (const char *fmt, ...)**
  
  Appends printf formatted messages to the end of the buffer.

- **void remove (int start, int end)**
  
  Deletes a range of characters in the buffer.

- **void remove_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)**
  
  Removes a modify callback.

- **void remove_predelete_callback (Fl_Text_Predelete_Cb predelCB, void *cbArg)**
  
  Removes a callback routine *bufPreDeleteCB* associated with argument *cbArg* to be called before text is deleted from the buffer.

- **void remove_secondary_selection ()**
  
  Removes the text from the buffer corresponding to the secondary text selection object.

- **void remove_selection ()**
  
  Removes the text in the primary selection.

- **void replace (int start, int end, const char *text)**
  
  Deletes the characters between *start* and *end*, and inserts the null-terminated string *text* in their place in the buffer.

- **void replace_secondary_selection (const char *text)**
  
  Replaces the text from the buffer corresponding to the secondary text selection object with the new string *text*.

- **void replace_selection (const char *text)**
  
  Replaces the text in the primary selection.

- **int rewind_lines (int startPos, int nLines)**
  
  Finds and returns the position of the first character of the line *nLines* backwards from *startPos* (not counting the character pointed to by *startPos* if that is a newline) in the buffer.

- **int savefile (const char *file, int buflen=128 *1024)**
  
  Saves a text file from the current buffer.

- **int search_backward (int startPos, const char *searchString, int *foundPos, int matchCase=0) const**
Search backwards in buffer for string `searchString`, starting with the character at `startPos`, returning the result in `foundPos`.

- `int search_forward` (int startPos, const char ∗searchString, int ∗foundPos, int matchCase=0) const
  Search forwards in buffer for string `searchString`, starting with the character `startPos`, and returning the result in `foundPos`.

- `void secondary_select` (int start, int end)
  Selects a range of characters in the secondary selection.

- `int secondary_selected` ()
  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.

- `void vprintf` (const char ∗fmt, va_list ap)
  Can be used by subclasses that need their own printf() style functionality.

- `int word_end` (int pos) const
Fl_Text_Buffer Class Reference

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 f_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
  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.

31.129.1 Detailed Description

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

All text in `Fl_Text_Buffer` must be encoded in UTF-8. All indices used in the function calls must be aligned to the start of a UTF-8 sequence. All indices and pointers returned will be aligned. All functions that return a single character will return that in an unsigned int in UCS-4 encoding.

The `Fl_Text_Buffer` class is used by the `Fl_Text_Display` and `Fl_Text_Editor` to manage complex text data and is based upon the excellent NEdit text editor engine - see [https://sourceforge.net/projects/nedit/](https://sourceforge.net/projects/nedit/).
31.129.2 Constructor & Destructor Documentation

31.129.2.1 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>

31.129.3 Member Function Documentation

31.129.3.1 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);
```

31.129.3.2 address()[1/2]

const char* Fl_Text_Buffer::address (  
   int pos  
) const [inline]

Convert a byte offset in buffer into a memory address.
Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address

31.129.3.3 address()

```c
char* Fl_Text_Buffer::address ( int pos ) [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address

31.129.3.4 append()

```c
void Fl_Text_Buffer::append ( const char * t ) [inline]
```

Appends the text string to the end of the buffer.

Parameters

| t | UTF-8 encoded and nul terminated text |

31.129.3.5 appendfile()

```c
int Fl_Text_Buffer::appendfile ( const char * file,
                                 int buflen = 128*1024 ) [inline]
```

Appends the named file to the end of the buffer.

See also insertfile().
31.129.3.6 byte_at()

char Fl_Text_Buffer::byte_at ( int pos ) const

Returns the raw byte at the specified position pos in the buffer. Positions start at 0.

Parameters

| pos | byte offset into buffer |

Returns

unencoded raw byte

31.129.3.7 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

31.129.3.8 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fromBuf</code></td>
<td>source text buffer, may be the same as this</td>
</tr>
<tr>
<td><code>fromStart</code></td>
<td>byte offset into buffer</td>
</tr>
<tr>
<td><code>fromEnd</code></td>
<td>byte offset into buffer</td>
</tr>
<tr>
<td><code>toPos</code></td>
<td>destination byte offset into buffer</td>
</tr>
</tbody>
</table>

#### 31.129.3.9 `count_displayed_characters()`

```c
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.

#### 31.129.3.10 `count_lines()`

```c
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.

#### 31.129.3.11 `findchar_backward()`

```c
int Fl_Text_Buffer::findchar_backward ( int startPos, unsigned int searchChar, int * foundPos ) const
```

Search backwards in buffer `buf` for character `searchChar`, starting with the character before `startPos`, returning the result in `foundPos`.

Returns 1 if found, 0 if not. The difference between this and `search_backward()` is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startPos</code></td>
<td>byte offset to start position</td>
</tr>
<tr>
<td><code>searchChar</code></td>
<td>UCS-4 character that we want to find</td>
</tr>
<tr>
<td><code>foundPos</code></td>
<td>byte offset where the character was found</td>
</tr>
</tbody>
</table>
31.129 Fl_Text_Buffer Class Reference 945

Returns

1 if found, 0 if not

31.129.3.12 findchar_forward()

```c
int Fl_Text_Buffer::findchar_forward (  
   int startPos,  
   unsigned searchChar,  
   int * foundPos ) const
```

Finds the next occurrence of the specified character.

Search forwards in buffer for character `searchChar`, starting with the character `startPos`, and returning the result in `foundPos`. Returns 1 if found, 0 if not. The difference between this and `search_forward()` is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchChar</td>
<td>UCS-4 character that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the character was found</td>
</tr>
</tbody>
</table>

Returns

1 if found, 0 if not

31.129.3.13 highlight_text()

```c
char * Fl_Text_Buffer::highlight_text ( )
```

Returns the highlighted text.

When you are done with the text, free it using the `free()` function.

31.129.3.14 insert()

```c
void Fl_Text_Buffer::insert (  
   int pos,  
   const char * text )
```

Inserts null-terminated string `text` at position `pos`.
Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>insertion position as byte offset (must be UTF-8 character aligned)</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>UTF-8 encoded and nul terminated text</td>
</tr>
</tbody>
</table>

31.129.3.15 insert()

int Fl_Text_Buffer::insert_ (  
    int pos,  
    const char * text ) [protected]

Internal (non-redisplaying) version of insert().

Returns the length of text inserted (this is just strlen(text), however this calculation can be expensive and the length will be required by any caller who will continue on to call redisplay). pos must be contiguous with the existing text in the buffer (i.e. not past the end).

Returns

the number of bytes inserted

31.129.3.16 insertfile()

int Fl_Text_Buffer::insertfile (  
    const char * file,  
    int pos,  
    int buflen = 128*1024 )

Inserts a file at the specified position.

Returns

• 0 on success
• non-zero on error (strerror() contains reason)
• 1 indicates open for read failed (no data loaded)
• 2 indicates error occurred while reading data (data was partially loaded)

File can be UTF-8 or CP1252 encoded. If the input file is not UTF-8 encoded, the Fl_Text_Buffer widget will contain data transcoded to UTF-8. By default, the message Fl_Text_Buffer::file_encoding_warning_message will warn the user about this.

See also

input_file_was_transcoded and transcoding_warning_action.
31.129.3.17  is_word_separator()

bool Fl_Text_Buffer::is_word_separator (  
    int pos   ) const

Returns whether character at position \texttt{pos} is a word separator.
Pos must be at a character boundary.

31.129.3.18  length()

int Fl_Text_Buffer::length ( ) const [inline]

Returns the number of bytes in the buffer.
Returns  
\text{size of text in bytes}

31.129.3.19  line_end()

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
\begin{tabular}{|c|c|}
\hline
\texttt{pos} & byte index into buffer \\
\hline
\end{tabular}

Returns  
\text{byte offset to line end}

31.129.3.20  line_start()

int Fl_Text_Buffer::line_start (  
    int pos   ) const

Returns the position of the start of the line containing position \texttt{pos}.

Parameters
\begin{tabular}{|c|c|}
\hline
\texttt{pos} & byte index into buffer \\
\hline
\end{tabular}
31.129.3.21  line_text()

char * Fl_Text_Buffer::line_text (  
    int pos ) const

Returns the text from the entire line containing the specified character position.
When you are done with the text, free it using the free() function.

Parameters

pos  byte index into buffer

Returns

copy of UTF-8 text, must be free'd

31.129.3.22  loadfile()

int Fl_Text_Buffer::loadfile (  
    const char * file,  
    int buflen = 128*1024 ) [inline]

Loads a text file into the buffer.
See also insertfile().

31.129.3.23  next_char()

int Fl_Text_Buffer::next_char (  
    int ix ) const

Returns the index of the next character.

Parameters

ix  index to the current character
### 31.129.3.24 outputfile()

```cpp
int Fl_Text_Buffer::outputfile (const char * file, int start, int end, int buflen = 128*1024 )
```

Writers 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)`

### 31.129.3.25 prev_char()

```cpp
int Fl_Text_Buffer::prev_char (int ix) const
```

Returns the index of the previous character.

**Parameters**

| ix     | index to the current character |

### 31.129.3.26 printf()

```cpp
void Fl_Text_Buffer::printf (const char * fmt, ... )
```

Appends printf formatted messages to the end of the buffer.

**Example:**

Generated by Doxygen
int main(..) {
   // Create a text display widget and assign it a text buffer
   Fl_Text_Display *tdsp = new Fl_Text_Display(..);
   Fl_Text_Buffer *tbuf = new Fl_Text_Buffer();
   tdsp->buffer(tbuf);
   // Append three lines of formatted text to the buffer
   tbuf->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
   tbuf->printf("The current PID is %ld.\n", (long)getpid());
}

Note
The expanded string is currently limited to 1024 characters.

Parameters

\begin{tabular}{|l|l|}
\hline
\textbf{in} & \textbf{fmt} \quad \text{is a printf format string for the message text.} \\
\hline
\end{tabular}

31.129.3.27 remove()

\begin{verbatim}
void Fl_Text_Buffer::remove (  
   int start,  
   int end )
\end{verbatim}

Deletes a range of characters in the buffer.

Parameters

\begin{tabular}{|l|l|}
\hline
\textbf{start} & byte offset to first character to be removed \\
\textbf{end} & byte offset to character after last character to be removed \\
\hline
\end{tabular}

31.129.3.28 remove()

\begin{verbatim}
void Fl_Text_Buffer::remove_ (  
   int start,  
   int end ) [protected]
\end{verbatim}

Internal (non-redisplaying) version of \texttt{remove()}.

Removes the contents of the buffer between \texttt{start} and \texttt{end} (and moves the gap to the site of the delete).
31.129.3.29 replace()

```cpp
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>

31.129.3.30  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 \texttt{nLines} backwards from \texttt{startPos} (not counting the character pointed to by \texttt{startpos} if that is a newline) in the buffer.

\texttt{nLines == 0} means find the beginning of the line.

31.129.3.31  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 (\texttt{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

\texttt{outputfile(const char *file, int start, int end, int buflen)}

31.129.3.32  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 \texttt{searchString}, starting with the character at \texttt{startPos}, returning the result in \texttt{foundPos}.

Returns 1 if found, 0 if not.
31.129 Fl_Text_Buffer Class Reference

Parameters

| startPos | byte offset to start position |
| searchString | UTF-8 string that we want to find |
| foundPos | byte offset where the string was found |
| matchCase | if set, match character case |

Returns

1 if found, 0 if not

31.129.3.33 search_forward()

```cpp
int Fl_Text_Buffer::search_forward ( int startPos, const char * searchString, int * foundPos, int matchCase = 0 ) const
```

Search forwards in buffer for string `searchString`, starting with the character `startPos`, and returning the result in `foundPos`.

Returns 1 if found, 0 if not.

Parameters

| startPos | byte offset to start position |
| searchString | UTF-8 string that we want to find |
| foundPos | byte offset where the string was found |
| matchCase | if set, match character case |

Returns

1 if found, 0 if not

31.129.3.34 secondary_selection_text()

```cpp
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.
31.129.3.35 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.

31.129.3.36 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

31.129.3.37 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.

31.129.3.38 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
void Fl_Text_Buffer::text ( const char * text )

Replaces the entire contents of the text buffer.
**text_range()**

```cpp
char * Fl_Text_Buffer::text_range ( int start, int end ) const
```

Get a copy of a part of the text buffer.

Return a copy of the text between `start` and `end` character positions from text buffer `buf`. Positions start at 0, and the range does not include the character pointed to by `end`. When you are done with the text, free it using the `free()` function.

**Parameters**

<table>
<thead>
<tr>
<th>start</th>
<th>byte offset to first character</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>byte offset after last character in range</td>
</tr>
</tbody>
</table>

**Returns**

- newly allocated text buffer - must be free'd, text is UTF-8

---

**vprintf()**

```cpp
void Fl_Text_Buffer::vprintf ( const char * fmt, va_list ap )
```

Can be used by subclasses that need their own `printf()` style functionality.

e.g. `Fl_Simple_Terminal::printf()` would wrap around this method.

**Note**

The expanded string is currently limited to 1024 characters.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>is a printf format string for the message text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ap</td>
<td>is a va_list created by <code>va_start()</code> and closed with <code>va_end()</code>, which the caller is responsible for handling.</td>
</tr>
</tbody>
</table>
31.129.3.42 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

31.129.3.43 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

31.129.4 Member Data Documentation

31.129.4.1 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.
31.129.4.2 mPredeleteProcs

Fl_Text_Predelete_Cb* Fl_Text_Buffer::mPredeleteProcs [protected]

procedure to call before text is deleted from the buffer; at most one is supported.

31.129.4.3 mTabDist

int Fl_Text_Buffer::mTabDist [protected]

equiv.

number of characters in a tab

31.129.4.4 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

31.130 Fl_Text_Display Class Reference

Rich text display widget.

#include <Fl_Text_Display.H>

Inheritance diagram for Fl_Text_Display:

```
Fl_Widget
  Fl_Group
    Fl_Text_Display
      Fl_Simple_Terminal  Fl_Text_Editor
```

Generated by Doxygen
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

- void buffer (Fl_Text_Buffer *buf)
  
  Attach a text buffer to display, replacing the current buffer (if any).

- void buffer (Fl_Text_Buffer &buf)
  
  Sets the current text buffer associated with the text widget.

- Fl_Text_Buffer * buffer () const
  
  Gets the current text buffer associated with the text widget.

- 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.

- int cursor_style () const

- 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.

• void insert_position (int newPos)
  
  Sets the position of the text insertion cursor for text display.

• int insert_position () const
  
  Gets 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
  
  Returns the beginning of a line.

• void linenumber_align (Fl_Align val)
  
  Set alignment for line numbers (if enabled).

• Fl_Align linenumber_align () const
  
  Returns the alignment used for line numbers (if enabled).

• void linenumberbgcolor (Fl_Color val)
  
  Set the background color used for line numbers (if enabled).

• Fl_Color linenumberbgcolor () const
  
  Returns the background color used for line numbers (if enabled).

• void linenumberfgcolor (Fl_Color val)
  
  Set the foreground color used for line numbers (if enabled).

• Fl_Color linenumberfgcolor () const
  
  Returns the foreground color used for line numbers (if enabled).

• void linenumber_font (Fl_Font val)
  
  Set the font used for line numbers (if enabled).

• Fl_Font linenumber_font () const
  
  Returns the font used for line numbers (if enabled).

• void linenumber_format (const char *val)
  
  Sets the printf() style format string used for line numbers.

• const char *linenumber_format () const
  
  Returns the line number printf() format string.

• void linenumber_size (Fl_Fontsize val)
  
  Set the font size used for line numbers (if enabled).

• Fl_Fontsize linenumber_size () const
  
  Returns the font size used for line numbers (if enabled).

• void linenumber_width (int width)
  
  Set width of screen area for line numbers.

• int linenumber_width () const
  
  Returns the screen area width provided for line numbers.

• int move_down ()
  
  Moves the current insert position down one line.

• int move_left ()
  
  Moves the current insert position left one character.

• int move_right ()
  
  Moves the current insert position right one character.

• int move_up ()
  
  Moves the current insert position up one line.

• void next_word (void)
  
  Moves the current insert position right one word.

• void overstrike (const char *text)
Replaces text at the current insert position.

- `int position_style (int lineStartPos, int lineLen, int lineIndex) const`
  Find the correct style for a character.

- `int position_to_xy (int pos, int ∗x, int ∗y) const`
  Convert a character index into a pixel position.

- `void previous_word (void)`
  Moves the current insert position left one word.

- `virtual void recalc_display ()`
  Recalculate the display’s visible lines and scrollbar sizes.

- `void redisplay_range (int start, int end)`
  Marks text from start to end as needing a redraw.

- `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_size () const`
  Gets the current size of the scrollbars’ troughs, in pixels.

- `void scrollbar_size (int newSize)`
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.

- `int scrollbar_width () const`
  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

- `void scrollbar_width (int width)`
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

- `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.

Protected Types

  • enum {
    DRAW_LINE, FIND_INDEX, FIND_INDEX_FROM_ZERO, GET_WIDTH,
    FIND_CURSOR_INDEX }

Protected Member Functions

  • void absolute_top_line_number (int oldFirstChar)
    Re-calculate absolute top line number for a change in scroll position.
  • void calc_last_char ()
    Update last display character index.
  • void calc_line_starts (int startLine, int endLine)
    Update the line starts array.
  • void clear_rect (int style, int x, int y, int width, int height) const
    Clear a rectangle with the appropriate background color for style.
  • void display_insert ()
    Scroll the display to bring insertion cursor into view.
  • 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 pos_is_line_start, int *lineEnd, int *nextLineStart) const
Finds both the end of the current line and the start of the next line.

- **void find_wrap_range** (const char *deletedText, int pos, int nInserted, int nDeleted, int *modRangeStart, int *modRangeEnd, int *linesInserted, int *linesDeleted)
  Wrapping calculations.

- **int find_x** (const char *s, int len, int style, int x) const
  Find the index of the character that lies at the given x position / closest cursor position.

- **int get_absolute_top_line_number** () const
  Returns the absolute (non-wrapped) line number of the first line displayed.

- **int handle_vline** (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const
  Universal pixel machine.

- **int longest_vline** () const
  Find the longest line of all visible lines.

- **void maintain_absolute_top_line_number** (int state)
  Line numbering stuff, currently unused.

- **int maintaining_absolute_top_line_number** () const
  Returns true if a separate absolute top line number is being maintained.

- **void measure_deleted_lines** (int pos, int nDeleted)
  Wrapping calculations.

- **double measure_proportional_character** (const char *s, int colNum, int pos) const
  Wrapping calculations.

- **int measure_vline** (int visLineNum) const
  Returns the width in pixels of the displayed line pointed to by "visLineNum".

- **void offset_line_starts** (int newTopLineNum)
  Offset line start counters for a new vertical scroll position.

- **int position_to_line** (int pos, int *lineNum) const
  Convert a position index into a line number offset.

- **int position_to_linecol** (int pos, int *lineNum, int *column) const
  Find the line and column number of position pos.

- **void reset_absolute_top_line_number** ()
  Reestablish the absolute (non-wrapped) top line number.

- **int scroll_** (int topLineNum, int horizOffset)
  Scrolls the current buffer to start at the specified line and column.

- **double string_width** (const char *string, int length, int style) const
  Find the width of a string in the font of a particular style.

- **void update_h_scrollbar** ()
  Update horizontal scrollbar.

- **void update_line_starts** (int pos, int 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 wrapUsesCharacter** (int lineEndPos) const
  Check if the line break is caused by a newline or by line wrapping.

- **void wrapped_line_counter** (Fi_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosLineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLastLineMissingNewLine=true) const
  Wrapping calculations.

- **int xy_to_position** (int x, int y, int PosType=CHARACTER_POS) const
  Translate a pixel position into a character index.

- **void xy_to_rowcol** (int x, int y, int *row, int *column, int PosType=CHARACTER_POS) const
  Translate pixel coordinates into row and column.
Static Protected Member Functions

- static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText, void *cbArg)
  
  This is called whenever the buffer is modified.

- static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)
  
  This is called before any characters are deleted.

- static void h_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)
  
  Callback for drag or valueChanged on horizontal scrollbar.

- static void scroll_timer_cb (void *cbArg)
  
  Timer callback for scroll events.

- static void v_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)
  
  Callback for drag or valueChanged on vertical scrollbar.

Protected Attributes

- int damage_range1_end
- int damage_range1_start
- int damage_range2_end
- int damage_range2_start
- int display_insert_position_hint
- int dragging
- int dragPos
- int dragType
- Fl_Align linenumber_align_
- Fl_Color linenumberbgcolor_
- Fl_Color linenumberfgcolor_
- Fl_Font linenumber_font_
- const char *linenumber_format_
- Fl_Fontsize linenumber_size_
- int mAbsTopLineNum
- Fl_Text_Buffer *mBuffer
- double mColumnScale
- int mContinuousWrap
- Fl_Color mCursor_color
- int mCursorOldY
- int mCursorOn
- int mCursorPos
- int mCursorPreferredXPos
- int mCursorStyle
- int mCursorToHint
- int mFirstChar
- void *mHighlightCBArg
- int mHorizOffset
- int mHorizOffsetHint
- Fl_Scrollbar *mHScrollbar
- int mLastChar
- int mLineNumLeft
- int mLineNumWidth
- int mLineStarts
- int mMaxsize
- int mModifyingTabDistance
- int mNBufferLines

Generated by Doxygen
31.130 Fl_Text_Display Class Reference

- `int mNeedAbsTopLineNum`
- `int mLinesDeleted`
- `int mStyles`
- `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`
- `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**

**31.130.1 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.

**Figure 31.48 Fl_Text_Display widget**

Generated by Doxygen
Example Use

```c
#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_fgc_color(), 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 FLTK 1.3.3.

See also

Fl_Widget::shortcut_label(int)

31.130.2 Member Enumeration Documentation

31.130.2.1 anonymous enum

anonymous enum

text display cursor shapes enumeration
## Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL_CURSOR</td>
<td>I-beam.</td>
</tr>
<tr>
<td>CARET_CURSOR</td>
<td>caret under the text</td>
</tr>
<tr>
<td>DIM_CURSOR</td>
<td>dim I-beam</td>
</tr>
<tr>
<td>BLOCK_CURSOR</td>
<td>unfilled box under the current character</td>
</tr>
<tr>
<td>HEAVY_CURSOR</td>
<td>thick I-beam</td>
</tr>
<tr>
<td>SIMPLE_CURSOR</td>
<td>as cursor as Fl_Input cursor</td>
</tr>
</tbody>
</table>

### 31.130.2.2 anonymous enum

Anonymous enum

Wrap types - used in `wrap_mode()`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAP_NONE</td>
<td>don't wrap text at all</td>
</tr>
<tr>
<td>WRAP_AT_COLUMN</td>
<td>wrap text at the given text column</td>
</tr>
<tr>
<td>WRAP_AT_PIXEL</td>
<td>wrap text at a pixel position</td>
</tr>
<tr>
<td>WRAP_AT_BOUNDS</td>
<td>wrap text so that it fits into the widget width</td>
</tr>
</tbody>
</table>

### 31.130.3 Constructor & Destructor Documentation

#### 31.130.3.1 Fl_Text_Display()

```cpp
Fl_Text_Display::Fl_Text_Display (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0 )
```

Creates a new text display widget.

**Parameters**

<table>
<thead>
<tr>
<th>X,Y,W,H</th>
<th>position and size of widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>label text, defaults to none</td>
</tr>
</tbody>
</table>
### 31.130.3.2 \~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.

See also

* Fl_Text_Display::buffer(Fl_Text_Buffer* buf)

### 31.130.4 Member Function Documentation

#### 31.130.4.1 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.

Does nothing if the absolute top line number is not being maintained.

#### 31.130.4.2 buffer()[1/3]

**void Fl_Text_Display::buffer ( Fl_Text_Buffer* buf )**

Attach a text buffer to display, replacing the current buffer (if any).

Multiple text widgets can be associated with the same text buffer.

**Note**

The caller is responsible for the old (replaced) buffer (if any). This method does not delete the old buffer.

**Parameters**

| buf | attach this text buffer |
31.130.4.3 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

See also

Fl_Text_Display::buffer(Fl_Text_Buffer* buf)

31.130.4.4 buffer() [3/3]

Fl_Text_Buffer* Fl_Text_Display::buffer ( ) const [inline]

Gets the current text buffer associated with the text widget.

Multiple text widgets can be associated with the same text buffer.

Returns

current text buffer

See also

Fl_Text_Display::buffer(Fl_Text_Buffer* buf)  
Fl_Text_Display::buffer(Fl_Text_Buffer& buf)

31.130.4.5 buffer_modified_cb()    

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>

31.130.4.6  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>

31.130.4.7  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.

31.130.4.8  calc_line_starts()

```c
void Fl_Text_Display::calc_line_starts (  
    int startLine,  
    int endLine ) [protected]
```
Update the line starts array.

Scan through the text in the Text Display's buffer and recalculate the line starts array values beginning at index "startLine" and continuing through (including) "endLine". It assumes that the line starts entry preceding "startLine" (or mFirstChar if startLine is 0) is good, and re-counts newlines to fill in the requested entries. Out of range values for "startLine" and "endLine" are acceptable.
### clear_rect()

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>index into style table</td>
</tr>
<tr>
<td>X,Y,width,height</td>
<td>size and position of background area</td>
</tr>
</tbody>
</table>

### col_to_x()

```cpp
double Fl_Text_Display::col_to_x (  
    double col ) const
```

Convert a column number into an x pixel position.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>an approximate column number based on the main font</td>
</tr>
</tbody>
</table>

**Returns**

number of pixels from the left margin to the left of an average sized character

### count_lines()

```cpp
int Fl_Text_Display::count_lines (  
    int startPos,  
    int endPos,  
    bool startPosIsLineStart ) const
```

Generated by Doxygen
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

31.130.4.12  cursor_color() [1/2]

Fl_Color Fl_Text_Display::cursor_color ( ) const [inline]

Gets the text cursor color.

Returns

cursor color

31.130.4.13  cursor_color() [2/2]

void Fl_Text_Display::cursor_color ( Fl_Color n ) [inline]

Sets the text cursor color.

Parameters

| n | new cursor color |

31.130.4.14  cursor_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

<table>
<thead>
<tr>
<th>style</th>
<th>new cursor style</th>
</tr>
</thead>
</table>

31.130.4.15  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?

31.130.4.16  draw()

`void Fl_Text_Display::draw ( ) [protected], [virtual]`

Draw the widget.

This function tries to limit drawing to smaller areas if possible.

Reimplemented from Fl_Group.

Reimplemented in Fl_Simple_Terminal.

31.130.4.17  draw_cursor()

`void Fl_Text_Display::draw_cursor ( int X, int Y ) [protected]`

Draw a cursor with top center at X, Y.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>cursor position in pixels</td>
</tr>
</tbody>
</table>

31.130.4.18 `draw_line_numbers()`

```cpp
void Fl_Text_Display::draw_line_numbers {
    bool clearAll ) [protected]
```

Refresh the line number area.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearAll</td>
<td>– (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.</td>
</tr>
</tbody>
</table>

31.130.4.19 `draw_range()`

```cpp
void Fl_Text_Display::draw_range ( 
    int startpos,
    int endpos ) [protected]
```

Draw a range of text.

Refresh all of the text between buffer positions `startpos` and `endpos` not including the character at the position `endpos`.

If `endpos` points beyond the end of the buffer, refresh the whole display after `startpos`, including blank lines which are not technically part of any range of characters.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startpos</td>
<td>index of first character to draw</td>
</tr>
<tr>
<td>Endpos</td>
<td>index after last character to draw</td>
</tr>
</tbody>
</table>

31.130.4.20 `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>style</th>
<th>index into style lookup table</th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y</td>
<td>drawing origin</td>
</tr>
<tr>
<td>toX</td>
<td>rightmost position if this is a fill operation</td>
</tr>
<tr>
<td>string</td>
<td>text if this is a drawing operation</td>
</tr>
<tr>
<td>nChars</td>
<td>number of characters to draw</td>
</tr>
</tbody>
</table>

31.130.4.21 draw_text()

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>left,top</th>
<th>are in coordinates of the text drawing window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>width,height</td>
<td>size in pixels</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visLineNum</td>
<td>index of line in the visible line number lookup</td>
</tr>
<tr>
<td>leftClip, rightClip</td>
<td>pixel position of clipped area</td>
</tr>
<tr>
<td>leftCharIndex, rightCharIndex</td>
<td>index into line of segment that we want to draw</td>
</tr>
</tbody>
</table>

31.130.4.23 empty_vlines()

```cpp
int Fl_Text_Display::empty_vlines ( ) const [protected]
```

Return true if there are lines visible with no corresponding buffer text.

Returns

1 if there are empty lines

31.130.4.24 extend_range_for_styles()

```cpp
void Fl_Text_Display::extend_range_for_styles ( int * startPos, int * endpos ) [protected]
```

I don’t know what this does!

Extend the range of a redraw request (from `*start` to `*end`) with additional redraw requests resulting from changes to the attached style buffer (which contains auxiliary information for coloring or styling text).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPos</td>
<td></td>
</tr>
<tr>
<td>endpos</td>
<td></td>
</tr>
</tbody>
</table>

Todo Unicode?

31.130.4.25 find_line_end()

```cpp
void Fl_Text_Display::find_line_end ( int startPos, bool startPosIsLineStart, int * lineEnd, int * nextLineStart ) const [protected]
```
Finds both the end of the current line and the start of the next line.

Why? In continuous wrap mode, if you need to know both, figuring out one from the other can be expensive or error prone. The problem comes when there's a trailing space or tab just before the end of the buffer. To translate an end of line value to or from the next lines start value, you need to know whether the trailing space or tab is being used as a line break or just a normal character, and to find that out would otherwise require counting all the way back to the beginning of the line.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPosIsLineStart</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>lineEnd</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>nextLineStart</td>
</tr>
</tbody>
</table>

31.130.4.26 `find_wrap_range()`

```cpp
void Fl_Text_Display::find_wrap_range (const char * deletedText, int pos, int nInserted, int nDeleted, int * modRangeStart, int * modRangeEnd, int * linesInserted, int * linesDeleted) [protected]
```

Wrapping calculations.

When continuous wrap is on, and the user inserts or deletes characters, wrapping can happen before and beyond the changed position. This routine finds the extent of the changes, and counts the deleted and inserted lines over that range. It also attempts to minimize the size of the range to what has to be counted and re-displayed, so the results can be useful both for delimiting where the line starts need to be recalculated, and for deciding what part of the text to redisplay.

Parameters

| deletedText |
| pos |
| nInserted |
| nDeleted |
| modRangeStart |
| modRangeEnd |
| linesInserted |
| linesDeleted |

31.130.4.27 `find_x()`

```cpp
int Fl_Text_Display::find_x (```

Generated by Doxygen
const char * s,
int len,
int style,
int x ) const [protected]

Find the index of the character that lies at the given x position / closest cursor position.

Parameters

<table>
<thead>
<tr>
<th>s</th>
<th>UTF-8 text string</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>length of string</td>
</tr>
<tr>
<td>style</td>
<td>index into style lookup table</td>
</tr>
<tr>
<td>x</td>
<td>position in pixels - negative returns closest cursor position</td>
</tr>
</tbody>
</table>

Returns

index into buffer

31.130.4.28 get_absolute_top_line_number()

int Fl_Text_Display::get_absolute_top_line_number ( ) const [protected]

Returns the absolute (non-wrapped) line number of the first line displayed.

Returns 0 if the absolute top line number is not being maintained.

31.130.4.29 handle_vline()

int Fl_Text_Display::handle_vline ( int mode,
int lineStartPos,
int lineLen,
int leftChar,
int rightChar,
int Y,
int bottomClip,
int leftClip,
int rightClip ) const [protected]

Universal pixel machine.

We use a single function that handles all line layout, measuring, and drawing

- draw a text range
- return the width of a text range in pixels
- return the index of a character that is at a pixel position
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX</td>
</tr>
<tr>
<td>lineStartPos</td>
<td>index of first character</td>
</tr>
<tr>
<td>lineLen</td>
<td>size of string in bytes</td>
</tr>
<tr>
<td>leftChar, rightChar</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>drawing position</td>
</tr>
<tr>
<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

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAW_LINE</td>
<td>index of last drawn character</td>
</tr>
<tr>
<td>GET_WIDTH</td>
<td>width in pixels of text segment if we would draw it</td>
</tr>
<tr>
<td>FIND_INDEX</td>
<td>index of character at given x position in window coordinates</td>
</tr>
<tr>
<td>FIND_INDEX_FROM_ZERO</td>
<td>index of character at given x position without scrolling and widget offsets</td>
</tr>
</tbody>
</table>

Todo

- we need to handle hidden hyphens and tabs here!
- we handle all styles and selections
- we must provide code to get pixel positions of the middle of a character as well

31.130.4.30 highlight_data()

```java
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>an optional argument for the callback above, usually a pointer to the Text Display.</td>
</tr>
</tbody>
</table>

**Todo** "extendRangeForStyleMods" does not exist (might be a hangover from the port from nedit). Find the correct function.

### 31.130.4.31 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

### 31.130.4.32 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 insert(insert_position(),text) and then moving the insert position after the newly inserted text, except that it's optimized to do less redrawing.

**Parameters**

| text | new text in UTF-8 encoding. |
31.130.4.33 insert_position() [1/2]

```cpp
text
void Fl_Text_Display::insert_position (   
    int newPos )
```

Sets the position of the text insertion cursor for text display.

Moves the insertion cursor in front of the character at `newPos`. This function may trigger a redraw.

Parameters

| newPos | new caret position |

31.130.4.34 insert_position() [2/2]

```cpp
text
int Fl_Text_Display::insert_position ( ) const [inline]
```

Gets the position of the text insertion cursor for text display.

The insert position is the byte count (offset) from the beginning of the text buffer (starting with 0). Returns 0 (zero) if no buffer is associated to the text display. Returns `buffer() -> length()` if the insert position is at the end of the buffer.

Returns

- insert position index into text buffer

See also

- `insert_position(int)`

31.130.4.35 line_end()

```cpp
text
int Fl_Text_Display::line_end (   
    int startPos,   
    bool startPosIsLineStart ) const
```

Returns the end of a line.

Same as `buffer() -> line_end(startPos)`, but takes into account line breaks when wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as `true` to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Note that the definition of the end of a line is less clear when continuous wrap is on. With continuous wrap off, it's just a pointer to the newline that ends the line. When it's on, it's the character beyond the last `displayable` character on the line, where a whitespace character which has been "converted" to a newline for wrapping is not considered displayable. Also note that a line can be wrapped at a non-whitespace character if the line had no whitespace. In this case, this routine returns a pointer to the start of the next line. This is also consistent with the model used by `visLineLength`. 

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

Returns

new position as index

31.130.4.36  line_start()

```c
int Fl_Text_Display::line_start ( int pos ) const
```

Return the beginning of a line.

Same as `buffer()->line_start(pos)`, but returns the character after last wrap point rather than the last newline.

Parameters

| pos | index to starting character |

Returns

new position as index

31.130.4.37  linenumber_align()

```c
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
Class Documentation

31.130.4.38  linenumberbgcolor()

void Fl_Text_Display::linenumberbgcolor (  
    Fl_Color val  )

Set the background color used for line numbers (if enabled).

Version
   1.3.3


31.130.4.39  linenumberfgcolor()

void Fl_Text_Display::linenumberfgcolor (  
    Fl_Color val  )

Set the foreground color used for line numbers (if enabled).

Version
   1.3.3


31.130.4.40  linenumberfont()

void Fl_Text_Display::linenumberfont (  
    Fl_Font val  )

Set the font used for line numbers (if enabled).

Version
   1.3.3


31.130.4.41  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
31.130.42 linenumber_size()

void Fl_Text_Display::linenumber_size (
    Fl_Fontsize val )

Set the font size used for line numbers (if enabled).

Version
    1.3.3

31.130.43 linenumber_width()

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) |

31.130.44 longest_vline()

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

31.130.45 maintain_absolute_top_line_number()

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.
Todo  TextDPosToLineAndCol does not exist (nedit port?)

31.130.46  maintaining_absolute_top_line_number()

```cpp
int Fl_Text_Display::maintaining_absolute_top_line_number ( ) const [protected]
```

Returns true if a separate absolute top line number is being maintained.

The absolute top line number is used for displaying line numbers in continuous wrap mode or showing in the statistics line (the latter is currently not available in FLTK).

31.130.47  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

<table>
<thead>
<tr>
<th>pos</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeleted</td>
<td></td>
</tr>
</tbody>
</table>

31.130.48  measure_proportional_character()

```cpp
double Fl_Text_Display::measure_proportional_character ( const char * s, int xPix, int pos ) const [protected]
```

Wrapping calculations.

Measure the width in pixels of the first character of string "s" at a particular column "colNum" and buffer position "pos". This is for measuring characters in proportional or mixed-width highlighting fonts.

A note about proportional and mixed-width fonts: the mixed width and proportional font code in nedit does not get much use in general editing, because nedit doesn't allow per-language-mode fonts, and editing programs in a proportional font is usually a bad idea, so very few users would choose a proportional font as a default. There are still probably mixed-width syntax highlighting cases where things don't redraw properly for insertion/deletion, though static display and wrapping and resizing should now be solid because they are now used for online help display.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>text string</td>
</tr>
<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

31.130.4.49 measure_vline()

```cpp
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

31.130.4.50 move_down()

```cpp
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

31.130.4.51 move_left()

```cpp
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
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

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

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

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.

31.130.4.56  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

31.130.4.57  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 \texttt{pos} relative to the first line of displayed text, counting from 0 to \textit{visible lines} - 1. The line number is returned in \texttt{lineNum}.

Returns 0 if the line is not displayed. In this case \texttt{lineNum} is 0 as well.

Returns 1 if the line is displayed. In this case \texttt{lineNum} is the relative line number.
31.130 Fl_Text_Display Class Reference

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pos</th>
<th>byte position in buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>lineNum</td>
<td>relative line number of byte pos in buffer</td>
</tr>
</tbody>
</table>

Returns

whether the character at byte position pos is currently displayed

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>pos is not displayed; lineNum is invalid (zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pos is displayed; lineNum is valid</td>
</tr>
</tbody>
</table>

31.130.4.58 position_to_linecol()

```c
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.

31.130.4.59 position_to_xy()

```c
int Fl_Text_Display::position_to_xy (  
  int pos,
```

Generated by Doxygen
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>out</th>
<th>pos</th>
<th>character index</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<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

31.130.4.60 redisplay_range()

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>

31.130.4.61 reset_absolute_top_line_number()

void Fl_Text_Display::reset_absolute_top_line_number ( ) [protected]

Reestablish the absolute (non-wrapped) top line number.

Count lines from the beginning of the buffer to reestablish the absolute (non-wrapped) top line number. If mode is not continuous wrap, or the number is not being maintained, does nothing.
31.130.4.62 resize()

```cpp
void Fl_Text_Display::resize (  
    int X,  
    int Y,  
    int W,  
    int H ) [virtual]
```

Change the size of the displayed text area.

Calling this function will trigger a recalculation of all visible lines and of all scrollbar sizes.

Parameters

| X, Y, W, H | new position and size of this widget |

Reimplemented from Fl_Group.

31.130.4.63 rewind_lines()

```cpp
int Fl_Text_Display::rewind_lines (  
    int startPos,  
    int nLines )
```

Skip a number of lines back.

Same as buffer() -> rewind_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on.

Parameters

<table>
<thead>
<tr>
<th>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

31.130.4.64 scroll()

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topLineNum</td>
<td>top line number</td>
</tr>
<tr>
<td>horizOffset</td>
<td>column number</td>
</tr>
</tbody>
</table>

**Todo**  Column numbers make little sense here.

31.130.4.65  scroll()

```cpp
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topLineNum</td>
<td>top line number</td>
</tr>
<tr>
<td>horizOffset</td>
<td>in pixels</td>
</tr>
</tbody>
</table>

Returns

0 if nothing changed, 1 if we scrolled

31.130.4.66  scroll_timer_cb()

```cpp
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.

31.130.4.67  scrollbar_align(){1/2}

```cpp
Fl_Align Fl_Text_Display::scrollbar_align ( ) const [inline]
```

Gets the scrollbar alignment type.

Returns

scrollbar alignment
31.130.4.68 scrollbar_align() [2/2]

void Fl_Text_Display::scrollbar_align (Fl_Align a) [inline]

Sets the scrollbar alignment type.

Parameters

| a | new scrollbar alignment |

31.130.4.69 scrollbar_size() [1/2]

int Fl_Text_Display::scrollbar_size () const [inline]

Gets the current size of the scrollbars' troughs, in pixels.

If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

31.130.4.70 scrollbar_size() [2/2]

void Fl_Text_Display::scrollbar_size (int newSize) [inline]

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. 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()

31.130.4.71 scrollbar_width() [1/2]

int Fl_Text_Display::scrollbar_width ( ) const [inline]

Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width has been set.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed.

31.130.4.72 scrollbar_width() [2/2]

void Fl_Text_Display::scrollbar_width ( int width ) [inline]

Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed

31.130.4.73 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

31.130.4.74 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 |

31.130.4.75  show_cursor()

```cpp
tvoid 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). |

31.130.4.76  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()`.

31.130.4.77  skip_lines()

```cpp
int Fl_Text_Display::skip_lines ( 
    int startPos,
    int nLines,
    bool startPosIsLineStart )
```

Skip a number of lines forward.

Same as `Fl_Text_Buffer::skip_lines(startPos, nLines)`, but takes into account line breaks when wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

| startPos | index to starting character |
| nLines   | number of lines to skip ahead |
| startPosIsLineStart | avoid scanning back to the line start |
Returns

new position as index

31.130.4.78  string_width()

double Fl_Text_Display::string_width (const char * string, int length, int style) const [protected]

Find the width of a string in the font of a particular style.

Parameters

<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

31.130.4.79  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

31.130.4.80  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 |
fl_font

Gets the default font used when drawing text in the widget.

Returns

current text font face unless overridden by a style

fl_font

Sets the default font used when drawing text in the widget.

Parameters

- `s` default text font face

fl_fontsize

Gets the default size of text in the widget.

Returns

current text height unless overridden by a style

fl_fontsize

Sets the default size of text in the widget.
### Parameters

| s | new text size |

#### 31.130.4.85 update_h_scrollbar()

```cpp
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.

#### 31.130.4.86 update_line_starts()

```cpp
void Fl_Text_Display::update_line_starts ( int pos,
                                        int charsInserted,
                                        int charsDeleted,
                                        int linesInserted,
                                        int linesDeleted,
                                        int * scrolled ) [protected]
```

Update line start arrays and variables.

Update the line starts array, mTopLineNum, mFirstChar and lastChar for this text display after a modification to the text buffer, given by the position `pos` where the change began, and the numbers of characters and lines inserted and deleted.

**Parameters**

<table>
<thead>
<tr>
<th>pos</th>
<th>index into buffer of recent changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>charsInserted</td>
<td>number of bytes(!) inserted</td>
</tr>
<tr>
<td>charsDeleted</td>
<td>number of bytes(!) deleted</td>
</tr>
<tr>
<td>linesInserted</td>
<td>number of lines</td>
</tr>
<tr>
<td>linesDeleted</td>
<td>number of lines</td>
</tr>
<tr>
<td>scrolled</td>
<td>set to 1 if the text display needs to be scrolled</td>
</tr>
</tbody>
</table>

#### 31.130.4.87 update_v_scrollbar()

```cpp
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.
31.130.4.88 vline_length()

```cpp
int Fl_Text_Display::vline_length ( int visLineNum ) const [protected]
```
Count number of bytes in a visible line.

Return the length of a line (number of bytes) by examining entries in the line starts array rather than by scanning for newlines.

**Parameters**

| visLineNum | index of line in visible line array |

**Returns**

number of bytes in this line

31.130.4.89 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

31.130.4.90 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 |
31.130.4.91  

void Fl_Text_Display::wrap_mode ( 
    int wrap, 
    int wrapMargin )

Set the new text wrap mode.

If wrap mode is not zero, this call enables automatic word wrapping at column wrapMargin. Word-wrapping does not change the text buffer itself, only the way the text is displayed. Different Text Displays can have different wrap modes, even if they share the same Text Buffer.

Valid wrap modes are:

- WRAP_NONE : don't wrap text at all
- WRAP_AT_COLUMN : wrap text at the given text column
- WRAP_AT_PIXEL : wrap text at a pixel position
- WRAP_AT_BOUNDS : wrap text so that it fits into the widget width

Parameters

<table>
<thead>
<tr>
<th>wrap</th>
<th>new wrap mode (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapMargin</td>
<td>in WRAP_AT_COLUMN mode, text will wrap at the n'th character. For variable width fonts, an average character width is calculated. The column width is calculated using the current textfont or the first style when this function is called. If the font size changes, this function must be called again. In WRAP_AT_PIXEL mode, this is the pixel position.</td>
</tr>
</tbody>
</table>

31.130.4.92  

int Fl_Text_Display::wrap_uses_character ( 
    int lineEndPos ) const [protected]

Check if the line break is caused by a newline or by line wrapping.

Line breaks in continuous wrap mode usually happen at newlines (\n) or whitespace. This line-terminating character is not included in line width measurements and has a special status as a non-visible character. However, lines with no whitespace are wrapped without the benefit of a line terminating character, and this distinction causes endless trouble with all of the text display code which was originally written without continuous wrap mode and always expects to wrap at a newline character.

Given the position of the end of the line, as returned by 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

Todo  TextDEndOfLine and BufEndOfLine functions don't exist (nedit port?)

31.130.4.93  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? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Todo  Unicode?

31.130.4.94  wrapped_line_counter()

```cpp
void Fl_Text_Display::wrapped_line_counter (  
    Fl_Text_Buffer * buf,  
    int startPos,  
    int maxPos,
```

Generated by Doxygen
int maxLines,
bool startPosIsLineStart,
int styleBufOffset,
int * retPos,
int * retLines,
int * retLineStart,
int * retLineEnd,
bool countLastLineMissingNewLine = true ) const [protected]

Wrapping calculations.

Count forward from startPos to either maxPos or maxLines (whichever is reached first), and return all relevant positions and line count. The provided textBuffer may differ from the actual text buffer of the widget. In that case it must be a (partial) copy of the actual text buffer and the styleBufOffset argument must indicate the starting position of the copy, to take into account the correct style information.

Parameters

| in | buf | The text buffer to operate on |
| in | startPos | Starting index position into the buffer |
| in | maxPos | Maximum index position into the buffer we'll reach |
| in | maxLines | Maximum number of lines we'll reach |
| in | startPosIsLineStart | Flag indicating if startPos is start of line. (If set, prevents our having to find the line start) |
| in | styleBufOffset | Offset index position into style buffer. |
| out | retPos | Position where counting ended. When counting lines, the position returned is the start of the line "maxLines" lines beyond "startPos". |
| out | retLines | Number of line breaks counted |
| out | retLineStart | Start of the line where counting ended |
| out | retLineEnd | End position of the last line traversed |
| out | countLastLineMissingNewLine |

31.130.4.95 wrapped_row()

int Fl_Text_Display::wrapped_row ( int row ) const

Nobody knows what this function does.

Correct a row number from an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to a straight number of newlines from the top line of the display. Because rectangular selections are based on newlines, rather than display wrapping, and anywhere a rectangular selection needs a row, it needs it in terms of un-wrapped lines.

Parameters

| row | 
Returns

something unknown

**Todo** What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

### x_to_col()

```cpp
double Fl_Text_Display::x_to_col (double x) const
```

Convert an x pixel position into a column number.

**Parameters**

- **x** number of pixels from the left margin

**Returns**

an approximate column number based on the main font

### xy_to_position()

```cpp
int Fl_Text_Display::xy_to_position (int X, int Y, int posType = CHARACTER_POS) const [protected]
```

Translate a pixel position into a character index.

Translate window coordinates to the nearest (insert cursor or character cell) text position. The parameter `posType` specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest cursor position, and CHARACTER_POS means return the position of the character closest to \((X, Y)\).

**Parameters**

- **X, Y** pixel position
- **posType** CURSOR_POS or CHARACTER_POS

**Returns**

index into text buffer

Generated by Doxygen
31.130.4.98  xy_to_rowcol()

```c
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>out</th>
<th>X,Y</th>
<th>pixel coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>row,column</td>
<td>nearest row and column</td>
<td></td>
</tr>
<tr>
<td>posType</td>
<td>CURSOR_POS or CHARACTER_POS</td>
<td></td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Text_Display.H
- Fl_Text_Display.cxx

31.131  **Fl_Text_Editor Class Reference**

This is the FLTK text editor widget.

```c
#include <Fl_Text_Editor.H>
```

Inheritance diagram for Fl_Text_Editor:

```
Fl_Widget
   |
   v
Fl_Group
   |
   v
Fl_Text_Display
   |
   v
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 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, Key_Binding **list)
  Adds a key of state state with the function function to an arbitrary key binding list list.

- void add_key_binding (int key, int state, Key_Func f)
  Adds a key of state state with the function f.

- Key_Func bound_key_function (int key, int state, Key_Binding *list) const
  Returns the function associated with a key binding.

- Key_Func bound_key_function (int key, int state) const
  Returns the function associated with a key binding.

- void default_key_function (Key_Func f)
  Sets the default key function for unassigned keys.

- Fl_Text_Editor (int X, int Y, int W, int H, const char *l=0)
  The constructor creates a new text editor widget.

- virtual int handle (int e)
  Event handling.

- void insert_mode (int b)
  Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.

- int insert_mode ()
  Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.

- void remove_all_key_bindings (Key_Binding **list)
  Removes all of the key bindings associated with the text editor or list.

- void remove_all_key_bindings ()
  Removes all of the key bindings associated with the text editor or list.

- 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.

- void remove_key_binding (int key, int state)
  Removes the key binding associated with the key "key" of state "state".

- void tab_nav (int val)
  Enables or disables Tab key focus navigation.

- int tab_nav () const
  Check if Tab focus navigation is enabled.

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_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_meta_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'.

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
Static Protected Attributes

- static Key_Binding * global_key_bindings

  Global key binding list.

Additional Inherited Members

31.131.1 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.

31.131.2 Member Typedef Documentation

31.131.2.1 Key_Func

typedef int(* Fl_Text_Editor::Key_Func) (int key, Fl_Text_Editor *editor)

Key function binding callback type.

31.131.3 Constructor & Destructor Documentation

31.131.3.1 Fl_Text_Editor()

Fl_Text_Editor::Fl_Text_Editor (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * l = 0 )

The constructor creates a new text editor widget.

31.131.4 Member Function Documentation

Generated by Doxygen
31.131.4.1 add_default_key_bindings()

```cpp
void Fl_Text_Editor::add_default_key_bindings (  
        Key_Binding ** list )
```

Adds all of the default editor key bindings to the specified key binding list.

31.131.4.2 add_key_binding() [1/2]

```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_key_bindings.

31.131.4.3 add_key_binding() [2/2]

```cpp
void Fl_Text_Editor::add_key_binding (  
        int key,  
        int state,  
        Key_Func f ) [inline]
```

Adds a key of state state with the function f.

31.131.4.4 bound_key_function() [1/2]

```cpp
Fl_Text_Editor::Key_Func Fl_Text_Editor::bound_key_function (  
        int key,  
        int state,  
        Key_Binding * list ) const
```

Returns the function associated with a key binding.

31.131.4.5 bound_key_function() [2/2]

```cpp
Key_Func Fl_Text_Editor::bound_key_function (  
        int key,  
        int state ) const [inline]
```

Returns the function associated with a key binding.
31.131.4.6 default_key_function()

void Fl_Text_Editor::default_key_function (  
    Key_Func f ) [inline]

Sets the default key function for unassigned keys.

31.131.4.7 insert_mode()[1/2]

void Fl_Text_Editor::insert_mode (  
    int b ) [inline]

Sets the current insert mode; if non-zero, new text is inserted before the current cursor position. 
Otherwise, new text replaces text at the current cursor position.

31.131.4.8 insert_mode()[2/2]

int Fl_Text_Editor::insert_mode ( ) [inline]

Gets the current insert mode; if non-zero, new text is inserted before the current cursor position. 
Otherwise, new text replaces text at the current cursor position.

31.131.4.9 kf_backspace()

int Fl_Text_Editor::kf_backspace (  
    int c,  
    Fl_Text_Editor * e ) [static]

Does a backspace for key 'c' in the current buffer of editor 'e'.

Any current selection is deleted. Otherwise, the character left is deleted and the cursor moved. The key value 'c' is currently unused.

31.131.4.10 kf_c_s_move()

int Fl_Text_Editor::kf_c_s_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction indicated by control key 'c' in editor 'e'.

See also
    kf_ctrl_move().
31.131.4.11 kf_copy()

```cpp
int Fl_Text_Editor::kf_copy ( int c, Fl_Text_Editor * e ) [static]
```

Does a copy of selected text or the current character in the current buffer of editor 'e'.

The key value 'c' is currently unused.

31.131.4.12 kf_ctrl_move()

```cpp
int Fl_Text_Editor::kf_ctrl_move ( int c, Fl_Text_Editor * e ) [static]
```

Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.

Supported values for 'c' are currently:

- **FL_Home** -- moves the cursor to the beginning of the document
- **FL_End** -- moves the cursor to the end of the document
- **FL_Left** -- moves the cursor left one word
- **FL_Right** -- moves the cursor right one word
- **FL_Up** -- scrolls up one line, without moving cursor
- **FL_Down** -- scrolls down one line, without moving cursor
- **FL_Page_Up** -- moves the cursor to the beginning of the top line on the current page
- **FL_Page_Down** -- moves the cursor to the beginning of the last line on the current page

31.131.4.13 kf_cut()

```cpp
int Fl_Text_Editor::kf_cut ( int c, Fl_Text_Editor * e ) [static]
```

Does a cut of selected text in the current buffer of editor 'e'.

The key value 'c' is currently unused.

31.131.4.14 kf_default()

```cpp
int Fl_Text_Editor::kf_default ( int c, Fl_Text_Editor * e ) [static]
```

Inserts the text associated with key 'c' in editor 'e'.

Honors the current selection and insert/overwrite mode.
31.131.4.15 kf_delete()

```c
int Fl_Text_Editor::kf_delete (  
    int c,  
    Fl_Text_Editor * e ) [static]
```

Does a delete of selected text or the current character in the current buffer of editor 'e'.

The key value 'c' is currently unused.

31.131.4.16 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.

31.131.4.17 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.

31.131.4.18 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.

31.131.4.19 kf_home()

```c
int Fl_Text_Editor::kf_home (  
    int ,  
    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.
31.131.4.20  kf_ignore()

```cpp
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.

31.131.4.21  kf_insert()

```cpp
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.

31.131.4.22  kf_left()

```cpp
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.

31.131.4.23  kf_m_s_move()

```cpp
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().
31.131.4.24 kf_meta_move()

```cpp
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

31.131.4.25 kf_move()

```cpp
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

31.131.4.26 kf_page_down()

```cpp
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.

31.131.4.27 kf_page_up()

```cpp
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.
31.131.4.28  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.

31.131.4.29  kf_right()

```c
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.

31.131.4.30  kf_select_all()

```c
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.

31.131.4.31  kf_shift_move()

```c
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()

31.131.4.32  kf_undo()

```c
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.
31.131.4.33  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.

31.131.4.34  remove_all_key_bindings() [1/2]

void Fl_Text_Editor::remove_all_key_bindings (  
    Key_Binding ** list )

Removes all of the key bindings associated with the text editor or list.

31.131.4.35  remove_all_key_bindings() [2/2]

void Fl_Text_Editor::remove_all_key_bindings ( ) [inline]

Removes all of the key bindings associated with the text editor or list.

31.131.4.36  remove_key_binding() [1/2]

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_key_bindings.

31.131.4.37  remove_key_binding() [2/2]

void Fl_Text_Editor::remove_key_binding (  
    int key,  
    int state ) [inline]

Removes the key binding associated with the key "key" of state "state".
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 desireable 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 desireable when Fl_Text_Editor is used e.g. in a database input form.

Currently, this method is implemented as a convenience method that adjusts the key bindings for the Tab key. This implementation detail may change in the future. Know that changing the editor's key bindings for Tab and Shift-Tab may affect tab navigation.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If val is 0, Tab inserts a tab character (default). If val is 1, Tab navigates widget focus.</td>
</tr>
</tbody>
</table>

See also

`tab_nav()`, `Fl::OPTION_ARROW_FOCUS`.

Version

1.3.4 ABI feature

31.131.4.39 tab_nav() [2/2]

```cpp
int Fl_Text_Editor::tab_nav () const
```

Check if Tab focus navigation is enabled.

If disabled (default), hitting Tab inserts a tab character into the editor buffer.

If enabled, hitting Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget.

Returns

if Tab inserts tab characters or moves the focus

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>Tab inserts tab characters (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tab navigation is enabled.</td>
</tr>
</tbody>
</table>

See also

`tab_nav(int)`, `Fl::OPTION_ARROW_FOCUS`.

Version

1.3.4 ABI feature

31.131.5 Member Data Documentation
31.131.5.1  global_key_bindings

`Key_Binding* Fl_Text_Editor::global_key_bindings [static], [protected]`

Global key binding list.

Derived classes can add key bindings for all Fl_Text_Editor widgets by adding a `Key_Binding` to this list.

See also

```cpp
        add_key_binding(int key, int state, Key_Func f, Key_Binding** list);
```

The documentation for this class was generated from the following files:

- Fl_Text_Editor.H
- Fl_Text_Editor.cxx

31.132  Fl_Text_Selection Class Reference

This is an internal class for Fl_Text_Buffer to manage text selections.

```cpp
#include <Fl_Text_Buffer.H>
```

Public Member Functions

- int `end () const`
  
  Returns the byte offset to the character after the last selected character.

- int `includes (int pos) const`
  
  Returns true if position `pos` is in the Fl_Text_Selection.

- int `length () const`
  
  Returns the size in bytes of the selection.

- int `position (int *startpos, int *endpos) const`
  
  Returns the status and the positions of this selection.

- bool `selected () const`
  
  Returns true if any text is selected.

- void `selected (bool b)`
  
  Modifies the 'selected' flag.

- void `set (int startpos, int endpos)`
  
  Sets the selection range.

- int `start () const`
  
  Returns the byte offset to the first selected character.

- void `update (int pos, int nDeleted, int nInserted)`
  
  Updates a selection after text was modified.
Protected Attributes

- int mEnd
  *byte offset to the character after the last selected character*
- bool mSelected
  *this flag is set if any text is selected*
- int mStart
  *byte offset to the first selected character*

Friends

- class Fl_Text_Buffer

31.132.1 Detailed Description

This is an internal class for Fl_Text_Buffer to manage text selections.

All methods use byte (not UTF-8 character) offsets and start at 0. This class works correctly with UTF-8 strings assuming that the parameters for all calls are on character boundaries.

If the selection is inactive (not currently used), then selected() returns false and start() and end() return 0 (zero).

The stored offsets are in ascending order, hence the following conditions are true (pseudo code):

```cpp
if (!selected()) : (start() == 0) && (end() == 0) && (start() == end())
if (selected()) : start() < end()
always : 0 <= start() <= end()
always : length() == end() - start()
```

The selection size in bytes can always (unconditionally) be computed by

```cpp
int size = sel->end() - sel->start();
```

See also

- length()

Note

The protected member variables mStart and mEnd are not necessarily 0 (zero) if mSelected == false because they are not cleared when selected(false) is called (as of Jul 2017). This may be changed in the future.

31.132.2 Member Function Documentation
31.132.2.1 end()

```cpp
int Fl_Text_Selection::end ( ) const [inline]
```

Returns the byte offset to the character after the last selected character.
The returned offset is only valid if `selected()` returns true (non-zero). The offset is 0 if no text is selected (since FLTK 1.4.0).

**Note**

In FLTK 1.3.x the returned offset could be non-zero even if `selected()` would have returned 0.

Returns

byte offset or 0 if not selected.

31.132.2.2 includes()

```cpp
int Fl_Text_Selection::includes ( int pos ) const
```

Returns true if position `pos` is in the `Fl_Text_Selection`.
`pos` must be at a character boundary.

31.132.2.3 length()

```cpp
int Fl_Text_Selection::length ( ) const [inline]
```

Returns the size in bytes of the selection.
This is a convenience method. It always returns the same as
`end() - start()`
and it returns 0 if `selected() == false`.

Returns

size in bytes or 0 if not selected.

Since

FLTK 1.4.0

31.132.2.4 position()

```cpp
int Fl_Text_Selection::position ( int * startpos,
                                 int * endpos ) const
```

Returns the status and the positions of this selection.
This method returns the same as `selected()` as an `int` (0 or 1) in its return value and the offsets to the start of the selection in `startpos` and to the byte after the last selected character in `endpos`, if `selected()` is true.
If `selected()` is `false`, both offsets are set to 0.

**Note**

In FLTK 1.3.x `startpos` and `endpos` were *not modified* if `selected()` was `false`. 
Parameters

<table>
<thead>
<tr>
<th>startpos</th>
<th>return byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpos</td>
<td>return byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

Returns

whether the selection is active (\texttt{selected()}) or not

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if not selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if selected</td>
</tr>
</tbody>
</table>

See also

\texttt{selected(), start(), end()}

31.132.2.5 \texttt{selected()} [1/2]

\begin{verbatim}
bool Fl_Text_Selection::selected ( ) const [inline]
\end{verbatim}

Returns true if any text is selected.

Returns

\texttt{true} if any text has been selected, or \texttt{false} if no text is selected.

31.132.2.6 \texttt{selected()} [2/2]

\begin{verbatim}
void Fl_Text_Selection::selected ( bool b ) [inline]
\end{verbatim}

Modifies the 'selected' flag.

Parameters

<table>
<thead>
<tr>
<th>( b )</th>
<th>new flag</th>
</tr>
</thead>
</table>
31.132.2.7 set()

void Fl_Text_Selection::set (  
    int startpos,  
    int endpos )

Sets the selection range.

startpos and endpos must be at a character boundary.

If startpos != endpos selected() is set to true, else to false.

If startpos is greater than endpos they are swapped so that startpos <= endpos.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>startpos</th>
<th>byte offset to first selected character</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>endpos</td>
<td>byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>

31.132.2.8 start()

int Fl_Text_Selection::start ( ) const [inline]

Returns the byte offset to the first selected character.

The returned offset is only valid if selected() returns true. If the selection is not valid the returned offset is 0 since FLTK 1.4.0.

Note

In FLTK 1.3.x the returned offset could be non-zero even if selected() would have returned 0.

Returns

byte offset or 0 if not selected.

31.132.2.9 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.
The Fl_Tile class lets you resize its children by dragging the border between them.

Inheritance diagram for Fl_Tile:

```
Fl_Widget

Fl_Group

Fl_Tile
```

Public Member Functions

- **Fl_Tile** (int X, int Y, int W, int H, const char ∗L=0)
  
  *Creates a new Fl_Tile widget using the given position, size, and label string.*

- int **handle** (int event)
  
  *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 Fl_Tile widget and its children.*

Additional Inherited Members

31.133.1 Detailed Description

The Fl_Tile class lets you resize its children by dragging the border between them.
For the tiling to work correctly, the children of an \texttt{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 \texttt{Fl\_Tile}.

\texttt{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 \texttt{FL\_DOWN\_BOX} creating the impression of "ridges" where the boxes touch. What you see are actually two adjacent \texttt{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.

\texttt{Fl\_Tile} allows objects to be resized to zero dimensions. To prevent this you can use the \texttt{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 \texttt{Fl\_Tile} can figure out their layout. If desired, call \texttt{position()} after creating the children but before displaying the window to set the borders where you want.

\textbf{Note on \texttt{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 \texttt{Fl\_Tile} widget, and thus resize objects on the edges to zero width or height. When the entire \texttt{Fl\_Tile} widget is resized, the \texttt{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 \texttt{resizable()} widget must be the first child of the \texttt{Fl\_Tile} widget group. Otherwise some events (e.g. \texttt{FL\_MOVE} and \texttt{FL\_ENTER}) might be consumed by the \texttt{resizable()} widget so that they are lost for widgets covered (overlapped) by the \texttt{resizable()} widget.

Note

You can still resize widgets \texttt{inside} the \texttt{resizable()} to zero width and/or height, i.e. box \texttt{2b} above to zero width and box \texttt{3a} to zero height.

See also

\texttt{void Fl\_Group::resizable(Fl\_Widget \\&w)}

Example for \texttt{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.
31.133.2 Constructor & Destructor Documentation

31.133.2.1 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

31.133.3 Member Function Documentation

31.133.3.1 handle()

int Fl_Tile::handle (  
    int event )  [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

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.

### 31.133.3.2 position()

```cpp
void Fl_Tile::position {
    int oldx,
    int oldy,
    int newx,
    int newy
}
```

Drag the intersection at \((\text{oldx}, \text{oldy})\) to \((\text{newx}, \text{newy})\).

This redraws all the necessary children.

Pass zero as \text{oldx} or \text{oldy} to disable drag in that direction.

### 31.133.3.3 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 \text{resize()} method. It does not use Fl_Group::\text{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
31.134 Fl_Tiled_Image Class Reference

This class supports tiling of images over a specified area.

```
#include <Fl_Tiled_Image.H>
```

Inheritance diagram for Fl_Tiled_Image:

```
    Fl_Tiled_Image
     |         
     V         
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.

- virtual Fl_Image * copy (int W, int H)
  
  Creates a resized copy of the specified image.

- Fl_Image * copy ()

- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.

- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws a tiled image.

- void draw (int X, int Y)

- 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.

Protected Attributes

- int alloc_image_

- Fl_Image * image_

Additional Inherited Members

31.134.1 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.
31.134.2 Constructor & Destructor Documentation

31.134.2.1 \texttt{Fl\_Tiled\_Image()}

\texttt{Fl\_Tiled\_Image::Fl\_Tiled\_Image (}
\texttt{   Fl\_Image \ast i,}
\texttt{   int \_W = 0,}
\texttt{   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 \texttt{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 \ast bg\_xpm = new Fl\_Pixmap(bg\_xpm);
Fl\_Tiled\_Image \ast bg\_tiled = new Fl\_Tiled\_Image(bg\_xpm,0,0);
Fl\_Box \ast box = new Fl\_Box(40,40,300,100,"\n");
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 \texttt{Fl::scheme()} initializes the window's label type and image. A possible workaround is to use another \texttt{Fl\_Group} as the only child widget and to set the background image for this group as described above.

Todo
Fix \texttt{Fl\_Tiled\_Image} as background image for widgets and windows and fix the implementation of \texttt{Fl\_::scheme(const char \ast)}.

31.134.3 Member Function Documentation

31.134.3.1 \texttt{color\_average()}

```cpp
void Fl\_Tiled\_Image::color\_average (}
\texttt{   Fl\_Color c,}
\texttt{   float i } ) \ [virtual]
```

The \texttt{color\_average()} method averages the colors in the image with the FLTK color value \texttt{c}.
The \texttt{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 \texttt{Fl\_Image}.
31.134.3.2 copy()

```cpp
Fl_Tiled_Image * Fl_Tiled_Image::copy (  
    int W,  
    int H  )  [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of Fl_Shared_Image, released) when you are done with it.

Parameters

- **W,H**  width and height of the returned copied image

Reimplemented from Fl_Image.

31.134.3.3 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.

31.134.3.4 draw()

```cpp
void Fl_Tiled_Image::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0  )  [virtual]
```

Draws a tiled image.

Tiled images can be used as background images for widgets and windows. However, due to implementation constraints, 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 >= 0 (negative values are ignored). If one of the values is greater than the image width or height resp. (**cx >= image()->w()** or **cy >= 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`

### 31.135 Fl_Timer Class Reference

This is provided only to emulate the Forms Timer widget.

```cpp
#include <Fl_Timer.H>
```

Inheritance diagram for `Fl_Timer`:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Timer</td>
</tr>
</tbody>
</table>
```

#### 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.
- `void value (double)`  
  Sets the current timer value.
- `double value () const`  
  See void `Fl_Timer::value(double)`
- `~Fl_Timer ()`  
  Destroys the timer and removes the timeout.
Protected Member Functions

• void draw ()
  
  Draws the widget.

Additional Inherited Members

31.135.1 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.

31.135.2 Constructor & Destructor Documentation

31.135.2.1 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.

31.135.3 Member Function Documentation

31.135.3.1 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().

Generated by Doxygen
31.135.3.2  direction()

```cpp
void Fl_Timer::direction (char d) [inline]
```

Gets or sets the direction of the timer.

If the direction is zero then the timer will count up, otherwise it will count down from the initial `value()`.

31.135.3.3  draw()

```cpp
void Fl_Timer::draw () [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

31.135.3.4  handle()

```cpp
int Fl_Timer::handle (int event) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**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 |
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.

31.135.3.5 suspended() [1/2]

char Fl_Timer::suspended ( ) const [inline]

Gets or sets whether the timer is suspended.

31.135.3.6 suspended() [2/2]

void Fl_Timer::suspended (  
    char d  )

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

31.136 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.

Additional Inherited Members

31.136.1 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().

31.136.2 Constructor & Destructor Documentation

31.136.2.1 Fl_Toggle_Button()

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>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Toggle_Button.H
- Fl_Button.cxx
The **Fl_Tooltip** class provides tooltip support for all FLTK widgets.

```cpp
#include <Fl_Tooltip.H>
```

## Static Public Member Functions

- **static Fl_Color color ()**
  
  Gets the background color for tooltips.

- **static void color (Fl_Color c)**
  
  Sets the background color for tooltips.

- **static Fl_Widget * current ()**
  
  Gets the current widget target.

- **static void current (Fl_Widget *)**
  
  Sets the current widget target.

- **static Fl_Window * current_window (void)**
  
  Returns the window that is used for tooltips.

- **static float delay ()**
  
  Gets the tooltip delay.

- **static void delay (float f)**
  
  Sets the tooltip delay.

- **static void disable ()**
  
  Same as enable(0), disables tooltips on all widgets.

- **static void enable (int b=1)**
  
  Enables tooltips on all widgets (or disables if b is false).

- **static int enabled ()**
  
  Returns non-zero if tooltips are enabled.

- **static void enter_area (Fl_Widget *w, int X, int Y, int W, int H, const char *tip)**
  
  You may be able to use this to provide tooltips for internal pieces of your widget.

- **static Fl_Font font ()**
  
  Gets the typeface for the tooltip text.

- **static void font (Fl_Font i)**
  
  Sets the typeface for the tooltip text.

- **static float hidedelay ()**
  
  Gets the time until an open tooltip hides again.

- **static void hidedelay (float f)**
  
  Sets the time until an open tooltip hides again.

- **static float hoverdelay ()**
  
  Gets the tooltip hover delay, the delay between tooltips.

- **static void hoverdelay (float f)**
  
  Sets the tooltip hover delay, the delay between tooltips.

- **static int margin_height ()**
  
  Gets the amount of extra space above and below the tooltip's text.

- **static void margin_height (int v)**
  
  Sets the amount of extra space above and below the tooltip's text.

- **static int margin_width ()**
  
  Gets the amount of extra space left/right of the tooltip's text.

- **static void margin_width (int v)**
  
  Sets the amount of extra space left/right of the tooltip's text.
• static Fl_Fontsize size ()
  Gets the size of the tooltip text.
• static void size (Fl_Fontsize s)
  Sets the size of the tooltip text.
• static Fl_Color textcolor ()
  Gets the color of the text in the tooltip.
• static void textcolor (Fl_Color c)
  Sets the color of the text in the tooltip.
• static int wrap_width ()
  Gets the maximum width for tooltip’s text before it word wraps.
• static void wrap_width (int v)
  Sets the maximum width for tooltip’s text before it word wraps.

Static Public Attributes

• static void(* enter)(Fl_Widget *w) = nothing
• static void(* exit)(Fl_Widget *w) = nothing

Friends

• class Fl_TooltipBox
• void Fl_Widget::copy_tooltip (const char *)
• void Fl_Widget::tooltip (const char *)

31.137.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets.

It contains only static methods.

![Figure 31.51 Fl_Tooltip Options](image)

31.137.2 Member Function Documentation
31.137.2.1 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.

31.137.2.2 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.

31.137.2.3 current()

void Fl_Tooltip::current ( Fl_Widget * w ) [static]

Sets the current widget target.
Acts as though enter(widget) was done but does not pop up a tooltip. This is useful to prevent a tooltip from reappearing when a modal overlapping window is deleted. FLTK does this automatically when you click the mouse button.

31.137.2.4 delay() [1/2]

static float Fl_Tooltip::delay ( ) [inline], [static]

Gets the tooltip delay.
The default delay is 1.0 seconds.

31.137.2.5 delay() [2/2]

static void Fl_Tooltip::delay ( float f ) [inline], [static]

Sets the tooltip delay.
The default delay is 1.0 seconds.

31.137.2.6 disable()

static void Fl_Tooltip::disable ( ) [inline], [static]

Same as enable(0), disables tooltips on all widgets.
31.137.2.7 enable()

static void Fl_Tooltip::enable (int b = 1) [inline], [static]

Enables tooltips on all widgets (or disables if \( b \) is false).

31.137.2.8 enabled()

static int Fl_Tooltip::enabled () [inline], [static]

Returns non-zero if tooltips are enabled.

31.137.2.9 enter_area()

void Fl_Tooltip::enter_area (Fl_Widget * wid, int x, int y, int w, int h, const char * t) [static]

You may be able to use this to provide tooltips for internal pieces of your widget.

Call this after setting \( \text{Fl::belowmouse()} \) 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).

31.137.2.10 font() [1/2]

static Fl_Font Fl_Tooltip::font () [inline], [static]

Gets the typeface for the tooltip text.

31.137.2.11 font() [2/2]

static void Fl_Tooltip::font (Fl_Font i) [inline], [static]

Sets the typeface for the tooltip text.
31.137.2.12  hidedelay() [1/2]

static float Fl_Tooltip::hidedelay ( ) [inline], [static]

Gets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

31.137.2.13  hidedelay() [2/2]

static void Fl_Tooltip::hidedelay ( float f ) [inline], [static]

Sets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

31.137.2.14  hoverdelay() [1/2]

static float Fl_Tooltip::hoverdelay ( ) [inline], [static]

Gets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

31.137.2.15  hoverdelay() [2/2]

static void Fl_Tooltip::hoverdelay ( float f ) [inline], [static]

Sets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

31.137.2.16  margin_height() [1/2]

static int Fl_Tooltip::margin_height ( ) [inline], [static]

Gets the amount of extra space above and below the tooltip's text.
Default is 3.

31.137.2.17  margin_height() [2/2]

static void Fl_Tooltip::margin_height ( int v ) [inline], [static]

Sets the amount of extra space above and below the tooltip's text.
Default is 3.
### margin_width()

*static int Fl_Tooltip::margin_width ( ) [inline], [static]*

Gets the amount of extra space left/right of the tooltip's text.

Default is 3.

#### margin_width()

*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.

### size()

*static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]*

Gets the size of the tooltip text.

#### size()

*static void Fl_Tooltip::size ( Fl_Fontsize s ) [inline], [static]*

Sets the size of the tooltip text.

### textcolor()

*static Fl_Color Fl_Tooltip::textcolor ( ) [inline], [static]*

Gets the color of the text in the tooltip.

The default is black.

#### textcolor()

*static void Fl_Tooltip::textcolor ( Fl_Color c ) [inline], [static]*

Sets the color of the text in the tooltip.

The default is black.
31.137.2.24  wrap_width() [1/2]

static int Fl_Tooltip::wrap_width ( ) [inline], [static]

Gets the maximum width for tooltip's text before it word wraps.

Default is 400.

31.137.2.25  wrap_width() [2/2]

static void Fl_Tooltip::wrap_width ( int v ) [inline], [static]

Sets the maximum width for tooltip's text before it word wraps.

Default is 400.

The documentation for this class was generated from the following files:

- Fl_Tooltip.H
- Fl.cxx
- Fl_Tooltip.cxx

31.138  Fl_Tree Class Reference

Tree widget.

#include <Fl_Tree.H>

Inheritance diagram for Fl_Tree:

```
Fl_Widget
   ⟹ Fl_Group
       ⟹ Fl_Tree
```

Generated by Doxygen
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.

- **void callback_item (Fl_Tree_Item *item)**
  Sets the item that was changed for this callback.

- **Fl_Tree_Item * callback_item ()**
  Gets the item that caused the callback.

- **void callback_reason (Fl_Tree_Reason reason)**
  Sets the reason for this callback.

- **Fl_Tree_Reason callback_reason () const**
  Gets 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 (Fl_Tree_Item *item, int docallback=1)**
  Closes the specified 'item'.

- **int close (const char *path, int docallback=1)**
  Closes the item specified by 'path'.

- **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 (Fl_Tree_Item *item, int docallback=1)**
  Deselect the specified 'item'.

- **int deselect (const char *path, int docallback=1)**
  Deselect an item specified by 'path'.

- **int deselect_all (Fl_Tree_Item *item=0, int docallback=1)**
  Deselect 'item' and all its children.

- **void display (Fl_Tree_Item *item)**
  Displays 'item', scrolling the tree as necessary.

- **int displayed (Fl_Tree_Item *item)**
  Displays 'item', scrolling the tree as necessary.
See if 'item' is currently displayed on-screen (visible within the widget).

- int extend_selection (Fl_Tree_Item ∗from, Fl_Tree_Item ∗to, int val=1, bool visible=false)
  Extend a selection between 'from' and 'to' depending on 'visible'.

- int extend_selection_dir (Fl_Tree_Item ∗from, Fl_Tree_Item ∗to, int dir, int val, bool visible)
  Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.

- const Fl_Tree_Item ∗find_clicked (int yonly=0) const
  Find the item that was last clicked on.

- Fl_Tree_Item ∗find_clicked (int yonly=0)
  Non-const version of Fl_Tree::find_clicked(int yonly) const.

- Fl_Tree_Item ∗find_item (const char ∗path)
  Non-const version of Fl_Tree::find_item(const char ∗path) const.

- const Fl_Tree_Item ∗find_item (const char ∗path) const
  Find the item, given a menu style path, e.g.

- Fl_Tree_Item ∗first ()
  Returns the first item in the tree, or 0 if none.

- Fl_Tree_Item ∗first_selected_item ()
  Returns the first selected item in the tree.

- Fl_Tree_Item ∗first_visible ()
  Returns the first open(), visible item in the tree, or 0 if none.

- Fl_Tree_Item ∗first_visible_item ()
  Returns the first open(), visible item in the tree, or 0 if none.

- Fl_Tree (int X, int Y , int W, int H, const char ∗L=0)
  Constructor.

- Fl_Tree_Item ∗get_item_focus () const
  Get the item that currently has keyboard focus.

- int get_selected_items (Fl_Tree_Item_Array &items)
  Returns the currently selected items as an array of 'ret_items'.

- int handle (int e)
  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 (Fl_Tree_Item ∗item) const
  See if the specified 'item' is closed.

- int is_close (const char ∗path) const
  See if item specified by 'path' is closed.

- int is_hscroll_visible () const
  See if the horizontal scrollbar is currently visible.

- int is_open (Fl_Tree_Item ∗item) const
  See if 'item' is open.

- int is_open (const char ∗path) const
  See if item specified by 'path' is open.

- int is_scrollbar (Fl_Widget ∗w)
  See if widget 'w' is one of the Fl_Tree widget's scrollbars.

- int is_selected (Fl_Tree_Item ∗item) const
See if the specified 'item' is selected.
• int is_selected (const char *path)

See if item specified by 'path' 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'.
• 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 background color used for creating new items.
• FL_Color item_labelfgcolor (void) const

Get the default label foreground color used for creating new items.
• void item_labelfgcolor (FL_Color val)

Set the default label foreground color used for creating new items.
• FL_Font item_labelfont () const

Get the default font face used for creating new items.
• void item_labelfont (FL_Font val)

Set the default font face used for creating new items.
• FL_Fontsize item_labelsize () const

Get the default label fontsize used for creating new items.
• void item_labelsize (FL_Fontsize val)

Set the default label font size used for creating new items.
• int item_pathname (char *pathname, int pathnamelen, const FL_Tree_Item *item) const

Return 'pathname' of size 'pathnamelen' for the specified 'item'.
• FL_Tree_Item_Reselect_Mode item_reselect_mode () const

Returns the current item re/selection mode.
• void item_reselect_mode (FL_Tree_Item_Reselect_Mode mode)

Sets the item re/selection mode.
• int labelmarginleft () const

Get the amount of white space (in pixels) that should appear to the left of the label text.
• void labelmarginleft (int val)

Set the amount of white space (in pixels) that should appear to the left of the label text.
• FL_Tree_Item * last ()

Returns the last item in the tree.
• FL_Tree_Item * last_selected_item ()

Returns the last selected item in the tree.
• FL_Tree_Item * last_visible ()

Returns the last open(), visible item in the tree.
• FL_Tree_Item * last_visible_item ()

Returns the last open(), visible item in the tree.
• int linespacing () const

Get the amount of white space (in pixels) that should appear between items in the tree.
• void linespacing (int val)

Sets the amount of white space (in pixels) that should appear between items in the tree.
**void load** (class Fl_Preferences &)

Load FLTK preferences.

**int marginbottom () const**

Get the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

**void marginbottom (int val)**

Sets the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

**int marginleft () const**

Get the amount of white space (in pixels) that should appear between the widget’s left border and the tree’s contents.

**void marginleft (int val)**

Set the amount of white space (in pixels) that should appear between the widget’s left border and the left side of the tree’s contents.

**int margintop () const**

Get the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

**void margintop (int val)**

Sets the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

**Fl_Tree_Item * next (Fl_Tree_Item *item=0)**

Return the next item after ‘item’, or 0 if no more items.

**Fl_Tree_Item * next_item (Fl_Tree_Item *item, int dir=FL_Down, bool visible=false)**

Returns next item after ‘item’ in direction ‘dir’ depending on ‘visible’.

**Fl_Tree_Item * next_selected_item (Fl_Tree_Item *item=0, int dir=FL_Down)**

Returns the next selected item above or below ‘item’, depending on ‘dir’.

**Fl_Tree_Item * next_visible_item (Fl_Tree_Item *start, int dir)**

Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified ‘item’, or 0 if no more items.

**int open (Fl_Tree_Item *item, int docallback=1)**

Open the specified ‘item’.

**int open (const char *path, int docallback=1)**

Opens the item specified by ‘path’.

**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.

**const Fl_Tree_Prefs & prefs () const**

* Fl_Image * openicon () const

Returns the icon to be used as the ‘open’ icon.

**void openicon (Fl_Image *val)**

Sets the icon to be used as the ‘open’ icon.

**const Fl_Tree_Prefs & prefs () const**

* Fl_Tree_Item * prev (Fl_Tree_Item *item=0)

Return the previous item before ‘item’, or 0 if no more items.

**void recalc_tree ()**

Schedule tree to recalc the entire tree size.

**int remove (Fl_Tree_Item *item)**

Remove the specified ‘item’ from the tree.

**void resize (int, int, int, int)**

Resizes the Fl_Group widget and all of its children.

* Fl_Tree_Item * root ()
1048 Class Documentation

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 (Fl_Tree_Item ∗item, int docallback=1)
  Select the specified 'item'.
• int select (const char ∗path, int docallback=1)
  Select the item specified by 'path'.
• 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, 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 (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_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

Generated by Doxygen
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.

Protected Member Functions

- void do_callback_for_item (Fl_Tree_Item *item, Fl_Tree_Reason reason)
  
  Do the callback for the specified ‘item’ using ‘reason’, setting the callback_item() and callback_reason().

- void draw ()
  
  Standard FLTK draw() method, handles drawing the tree widget.

- void item_clicked (Fl_Tree_Item *val)
  
  Set the item that was last clicked.

Protected Attributes

- Fl_Scrollbar * _hscroll
  
  Horizontal scrollbar.

- int _th
  
  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

31.138.1 Detailed Description

Tree widget.

```
#include <FL/Fl_Tree.H>
[...]
Fl_Tree tree(X,Y,W,H); // Top level widget
```

```
|--- Fl_Tree_Item // Items in the tree
|  |--- Fl_Tree_Prefs // Preferences for the tree
|  |  |--- Fl_Tree_Connector (enum) // Connection modes
|  |  |--- Fl_Tree_Select (enum) // Selection modes
|  |--- Fl_Tree_Sort (enum) // Sort behavior
```

Similar to *Fl_Browser*, *Fl_Tree* is a browser of *Fl_Tree_Item*'s arranged in a parented hierarchy, or 'tree'. Subtrees can be expanded or closed. Items can be added, deleted, inserted, sorted and re-ordered.

The tree items may also contain other FLTK widgets, like buttons, input fields, or even "custom" widgets.

The callback() is invoked depending on the value of when():

- FL_WHEN_RELEASE – callback invoked when left mouse button is released on an item
- FL_WHEN_CHANGED – callback invoked when left mouse changes selection state

The simple way to define a tree:

```
#include <FL/Fl_Tree.H>
[...]
Fl_Tree tree(X,Y,W,H);
```

```
tree.begin();
tree.add("Flintstones/Fred");
tree.add("Flintstones/Wilma");
tree.add("Flintstones/Pebbles");
tree.add("Simpsons/Homer");
tree.add("Simpsons/Marge");
tree.add("Simpsons/Bart");
tree.add("Simpsons/Lisa");
tree.end();
```
FEATURES

- Items can be added with `add()`, removed with `remove()`, completely cleared with `clear()`, inserted with `insert()` and `insert_above()`, selected/deselected with `select()` and `deselect()`, open/closed with `open()` and `close()`, positioned on the screen with `show_item_top()`, `show_item_middle()` and `show_item_bottom()`, item children can be swapped around with `Fl_Tree_Item::swap_children()`, items can be moved around with `Fl_Tree_Item::move()`, an item's children can be walked with `Fl_Tree_Item::first()` and `Fl_Tree_Item::next()`, an item's children can be indexed directly with `Fl_Tree_Item::child()` and `Fl_Tree_Item::children()`, items can be moved from one subtree to another with `Fl_Tree_Item::deparent()` and `Fl_Tree_Item::reparent()`, sorting can be controlled when items are `add()ed` via `sortorder()`. You can walk the entire tree with `first()` and `next()`. You can walk visible items with `first_visible_item()` and `next_visible_item()`. You can walk selected items with `first_selected_item()` and `next_selected_item()`. Items can be found by their pathname using `find_item(const char * )`, and an item's pathname can be found with `item_pathname()`. The selected items' colors are controlled by `selection_color()` (inherited from `Fl_Widget`).

A hook is provided to allow you to redefine how item's labels are drawn via `Fl_Tree::item_draw_callback()`. Items can be interactively dragged using `FL_TREE_SELECT_SINGLE_DRAGGABLE`.

SELECTION OF ITEMS

The tree can have different selection behaviors controlled by `selectmode()`. The background color used for selected items is the `Fl_Tree::selection_color()`. The foreground color for selected items is controlled internally with `fl_contrast()`.

CHILD WIDGETS

FLTK widgets (including custom widgets) can be assigned to tree items via `Fl_Tree_Item::widget()`.

When an `Fl_Tree_Item::widget()` is defined, the default behavior is for the widget() to be shown in place of the item's label (if it has one). Only the widget()'s width will be used; the widget()'s x() and y() position will be managed by the tree, and the h() will track the item's height. This default behavior can be altered (ABI 1.3.1): Setting `Fl_Tree::item_draw_mode()`'s `FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET` flag causes the label + widget to be displayed together in that order, and adding the `FL_TREE_ITEM_HEIGHT_FROM_WIDGET` flag causes widget's height to define the widget's height.

ICONS

The tree's open/close icons can be redefined with `Fl_Tree::openicon()`, `Fl_Tree::closeicon()`. User icons can either be changed globally with `Fl_Tree::usericon()`, or on a per-item basis with `Fl_Tree_Item::usericon()`.

Various default preferences can be globally manipulated via `Fl_Tree_Prefs`, including colors, margins, icons, connection lines, etc.

FONTS AND COLORS

When adding new items to the tree, the new items get the defaults for fonts and colors from:

- `Fl_Tree::item_labelfont()` – The default item label font (default: FL_HELVETICA)
• Fl_Tree::item_labelsize() – The default item label size (default: FL_NORMAL_SIZE)
• Fl_Tree::item_labelfgcolor() – The default item label foreground color (default: FL_FOREGROUND_COLOR)
• Fl_Tree::item_labelbgcolor() – The default item label background color (default: 0xffffffff, which tree uses as ‘transparent’)

Each item (Fl_Tree_Item) inherits a copy of these font/color attributes when created, and each item has its own methods to let the app change these values on a per-item basis using methods of the same name:

• Fl_Tree_Item::labelfont() – The item’s label font (default: FL_HELVETICA)
• Fl_Tree_Item::labelsize() – The item’s label size (default: FL_NORMAL_SIZE)
• Fl_Tree_Item::labelfgcolor() – The item’s label foreground color (default: FL_FOREGROUND_COLOR)
• Fl_Tree_Item::labelbgcolor() – The item’s label background color (default: 0xffffffff, which uses the tree’s own bg color)

CALLBACKS

The tree’s callback() will be invoked when items change state or are open/closed. when() controls when mouse/keyboard events invoke the callback. callback_item() and callback_reason() can be used to determine the cause of the callback. e.g.

```c
void MyTreeCallback(Fl_Widget *w, void *data) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = (Fl_Tree_Item*)tree->callback_item(); // get selected item
    switch ( tree->callback_reason() ) {
        case FL_TREE_REASON_SELECTED: [...] 
        case FL_TREE_REASON_DESELECTED: [...] 
        case FL_TREE_REASON_RESELECTED: [...] 
        case FL_TREE_REASON_OPENED: [...] 
        case FL_TREE_REASON_CLOSED: [...] 
    }
}
```

SIMPLE EXAMPLES

To find all the selected items:

```c
for ( Fl_Tree_Item *i=first_selected_item(); i; i=next_selected_item(i) )
    printf("Item %s is selected\n", i->label());
```

To get an item’s full menu pathname, use Fl_Tree::item_pathname(), e.g.

```c
char pathname[256] = "???";
tree->item_pathname(pathname, sizeof(pathname), item); // eg. "Parent/Child/Item"
```

To walk all the items of the tree from top to bottom:
// 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:

// Find all of the item's children and print an indented report of their labels
void my_print_all_children(Fl_Tree_Item *item, int indent=0) |
for ( int t=0; t<item->children(); t++ ) |
printf("%*s Item: %s\n", indent, "", item->child(t)->label()); |
my_print_all_children(item->child(t), indent+4); // recurse |
}

To change the default label font and color when creating new items:

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:

// 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 31.53 Fl_Tree elements
The following shows the protected dimension variables ‘tree inner’ (tix..) and ‘tree outer’ (tox..):

![Figure 31.54 Fl_Tree inner/outer dimensions](image)

**KEYBOARD BINDINGS**

The following table lists keyboard bindings for navigating the tree:

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>FL_TREE_SELECT__MULTI</th>
<th>FL_TREE_SELECT__SINGLE</th>
<th>FL_TREE_SELECT__NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-A (Linux/Windows)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Command-A (Mac)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Space</td>
<td>Selects item</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Ctrl-Space</td>
<td>Toggle item</td>
<td>Toggle item</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Space</td>
<td>Extends selection</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Ctrl-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Shift-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Right / Left</td>
<td>Open/Close item</td>
<td>Open/Close item</td>
<td>Open/Close item</td>
</tr>
<tr>
<td>Up / Down</td>
<td>Move focus box up/down</td>
<td>Move focus box up/down</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Up / Shift-Down</td>
<td>Extend selection up/down</td>
<td>Move focus up/down</td>
<td>N/A</td>
</tr>
<tr>
<td>Home / End</td>
<td>Move to top/bottom of tree</td>
<td>Move to top/bottom of tree</td>
<td>Move to top/bottom of tree</td>
</tr>
<tr>
<td>PageUp / PageDown</td>
<td>Page up/down</td>
<td>Page up/down</td>
<td>Page up/down</td>
</tr>
</tbody>
</table>

31.138.2 Member Function Documentation
31.138.2.1 add() [1/2]

```c
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**

| in | path | The path to the item, e.g. "Flintstone/Fred". |
|    | item | The new item to be added. If NULL, a new item is created with a name that is the last element in 'path'. |

**Returns**

The new item added, or 0 on error.

**Version**

1.3.3

31.138.2.2 add() [2/2]

```c
Fl_Tree_Item * Fl_Tree::add (  
    Fl_Tree_Item * parent_item,  
    const char * name )
```

Add a new child item labeled 'name' to the specified 'parent_item'.

**Parameters**

| in | parent_item | The parent item the new child item will be added to. Must not be NULL. |
|    | name | The label for the new item |

**Returns**

The new item added.
31.138.2.3 calc_dimensions()

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 resize()ed 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

31.138.2.4 calc_tree()

void Fl_Tree::calc_tree ( )

Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.

On return:

- _tree_w will be the overall pixel width of the entire viewable tree
- _tree_h will be the overall pixel height **
- scrollbar visibility and pan sizes are updated
- internal _tix/_tiy/_tiw/_tih dimensions are updated

_tree_w/_tree_h include the tree's margins (e.g. marginleft()), whether items are open or closed, label contents and font sizes, etc.

The tree hierarchy's size is managed separately from the widget's size as an optimization; this way resize() on the widget doesn't involve recalculating the tree's hierarchy needlessly, as widget size has no bearing on the tree hierarchy.

The tree hierarchy's size only changes when items are added/removed, open/closed, label contents or font sizes changed, margins changed, etc.

This calculation involves walking the entire tree from top to bottom, potentially a slow calculation if the tree has many items (potentially hundreds of thousands), and should therefore be called sparingly.

For this reason, recalc_tree() is used as a way to /schedule/ calculation when changes affect the tree hierarchy's size.

Apps may want to call this method directly if the app makes changes to the tree's geometry, then immediately needs to work with the tree's new dimensions before an actual redraw (and recalc) occurs. (This use by an app should only rarely be needed)
31.138.2.5 callback_item() [1/2]

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.

31.138.2.6 callback_item() [2/2]

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.

31.138.2.7 callback_reason() [1/2]

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.

31.138.2.8 callback_reason() [2/2]

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:

```c
void MyTreeCallback(Fl_Widget *w, void *userdata) {  
    Fl_Tree *tree = (Fl_Tree*)w;  
    Fl_Tree_Item *item = tree->callback_item(); // the item changed (can be 
        NULL if more than one item was changed!)  
    switch ( tree->callback_reason() ) { // reason callback was invoked  
        case FL_TREE_REASON_OPENED: ..item was opened..  
        case FL_TREE_REASON_CLOSED: ..item was closed..  
        case FL_TREE_REASON_SELECTED: ..item was selected..  
        case FL_TREE_REASON_RESELECTED: ..item was reselected (double-clicked, etc  
            );  
        case FL_TREE_REASON_DESELECTED: ..item was deselected..  
    }  
}
```

See also

    item_reselect_mode() – enables FL_TREE_REASON_RESELECTED events
31.138.2.9 clear()

void Fl_Tree::clear ( )

Clear the entire tree's children, including the root.

The tree will be left completely empty.

31.138.2.10 clear_children()

void Fl_Tree::clear_children ( Fl_Tree_Item * item )

Clear all the children for 'item'.

Item may not be NULL.

31.138.2.11 close()[1/2]

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>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_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()
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

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
</table>
| in | docallback | A flag that determines if the callback() is invoked or not:
|    |    | • 0 - callback() is not invoked
|    |    | • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED |

Returns

• 1 – OK: item closed
• 0 – OK: item was already closed, no change
• -1 – ERROR: item was not found

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()
Parameters

| in       | val               | – The new image, or zero to use the default [-] icon. |

31.138.2.15 connectorstyle()

void Fl_Tree::connectorstyle (  
    Fl_Tree_Connector val
)

Sets the line drawing style for inter-connecting items.

See Fl_Tree_Connector for possible values.

31.138.2.16 deselect() [1/2]

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

| in       | item | – the item to be deselected. Must not be NULL. |
| in       | docallback | – A flag that determines if the callback() is invoked or not:
|          |        | • 0 - the callback() is not invoked  
|          |        | • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED |

Returns

• 0 - item was already deselected, no change was made
• 1 - item's state was changed

31.138.2.17 deselect() [2/2]

int Fl_Tree::deselect (  
    const char * path,  
    int docallback = 1
)
Deselect an item specified by 'path'.

Invokes the callback depending on the value of optional parameter 'docallback'. Handles calling redraw() if anything changed.

Items or submenus that themselves contain slashes ('/' or '/') should be escaped, e.g. deselect("Holidays/12/25/2010").

The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

- 1 - OK: item's state was changed
- 0 - OK: item was already deselected, no change was made
- -1 - ERROR: item was not found

31.138.2.18 deselect_all()

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.

31.138.2.19 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. |

31.138.2.20 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

| in | item | The item to be checked. If NULL, first() is used. |

Returns

1 if displayed, 0 if scrolled off screen or no items are in tree.

31.138.2.21 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 \texttt{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 \texttt{redraw()} if anything changed.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>from</th>
<th>Starting item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>to</td>
<td>Ending item</td>
</tr>
<tr>
<td>in</td>
<td>val</td>
<td>Select or deselect items (0=deselect, 1=select, 2=toggle)</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=affect only open(), visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3 ABI feature

31.138.2.22 \texttt{extend selection dir()}

\begin{verbatim}
int Fl\_Tree::extend_selection_dir ( 
    Fl\_Tree\_Item * from, 
    Fl\_Tree\_Item * to, 
    int dir, 
    int val, 
    bool visible )
\end{verbatim}

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 \texttt{extend selection(Fl\_Tree\_Item*,Fl\_Tree\_Item*,int,bool)} should be used.

Handles calling \texttt{redraw()} if anything changed.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>from</th>
<th>Starting item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>to</td>
<td>Ending item</td>
</tr>
<tr>
<td>in</td>
<td>dir</td>
<td>Direction to extend selection (FL_Up or FL_Down)</td>
</tr>
<tr>
<td>in</td>
<td>val</td>
<td>0=deselect, 1=select, 2=toggle</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=affect only open(), visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

The number of items whose selection states were changed, if any.

Version

1.3.3

31.138.2.23  find_clicked()

```cpp
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

31.138.2.24  find_item()

```cpp
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\1\25\2010");  // Adds item "12/25/2010"
; tree->add("/Pathnames/c:\\Program Files\\\MyApp"); // Adds item "c:\Program Files\MyApp"
;```
Parameters

| in | path | – the tree item's pathname to be found (e.g. "Flintstones/Fred") |

Returns

The item, or NULL if not found.

See also

item_pathname()

31.138.2.25 first()

Fl_Tree_Item * Fl_Tree::first ()

Returns the first item in the tree, or 0 if none.

Use this to walk the tree in the forward direction, e.g.

```cpp
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) )
    printf("Item: %s\n", item->label());
```

Returns

First item in tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()

31.138.2.26 first_selected_item()

Fl_Tree_Item * Fl_Tree::first_selected_item ()

Returns the first selected item in the tree.

Use this to walk the tree from top to bottom looking for all the selected items, e.g.

```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\n", i->label());
```

Returns

The first selected item, or 0 if none.

See also

first_selected_item(), last_selected_item(), next_selected_item()
**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.

**first_visible_item()**

```cpp
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

**get_selected_items()**

```cpp
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;
// Manipulate the returned array
for ( int t=0; t<items.total(); t++ ) { //
    Fl_Tree_Item &item = items[t];
    ..do stuff with each selected item..
}
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ret_items</td>
</tr>
</tbody>
</table>

Returns

The number of items in the returned array.

See also

first_selected_item(), next_selected_item()

Version

1.3.3 ABI feature

31.138.2.30 handle()

```cpp
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.

31.138.2.31 hposition() [1/2]

```cpp
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.

31.138.2.32 hposition() [2/2]

```cpp
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.

31.138.2.33 insert()

```c
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:

```c
 tree->add("Aaa/000"); // "000" is index 0 in Aaa's children
 tree->add("Aaa/111"); // "111" is index 1 in Aaa's children
 tree->add("Aaa/222"); // "222" is index 2 in Aaa's children
 // How to use insert() to insert a new item between Aaa/111 + Aaa/222
 Fl_Tree_Item *item = tree->find_item("Aaa"); // get parent item Aaa
 if (item) tree->insert(item, "New item", 2); // insert as a child of Aaa at index #2
```

Parameters

| 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.

See also

insert_above()

31.138.2.34  insert_above()

Fl_Tree_Item * Fl_Tree::insert_above (  
    Fl_Tree_Item * above,  
    const char * name  
)

Inserts a new item 'name' above the specified Fl_Tree_Item 'above'.

Example:

```
:  
  tree->add("Aaa/000");  // "000" is index 0 in Aaa’s children  
  tree->add("Aaa/111");  // "111" is index 1 in Aaa’s children  
  tree->add("Aaa/222");  // "222" is index 2 in Aaa’s children  
  ...  
  // How to use insert_above() to insert a new item above Aaa/222  
  Fl_Tree_Item *item = tree->find_item("Aaa/222");  // get item Aaa/222  
  if (item) tree->insert_above(item, "New item");  // insert new item above it  
:  
```

Parameters

| 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

insert()

31.138.2.35  is_close() [1/2]

int Fl_Tree::is_close (  
    Fl_Tree_Item * item  
) const

See if the specified 'item' is closed.
Parameters

| in | item | – the item to be tested. Must not be NULL. |

Returns

- 1 : item is closed
- 0 : item is open

31.138.2.36 is_close()

```cpp
text
int Fl_Tree::is_close (const char *path) const
```

See if item specified by `path` is closed.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `is_close("Holidays/12\25\2010")`.

Parameters

| in | path | – the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

- 1 - OK: item is closed
- 0 - OK: item is open
- -1 - ERROR: item was not found

31.138.2.37 is_hscroll_visible()

```cpp
text
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.
31.138.2.38 is_open() [1/2]

```cpp
int Fl_Tree::is_open (  
    Fl_Tree_Item * item ) const
```

See if 'item' is open.

Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

| in  | item | – the item to be tested. Must not be NULL |

Returns

- 1: item is open
- 0: item is closed

31.138.2.39 is_open() [2/2]

```cpp
int Fl_Tree::is_open (  
    const char * path ) const
```

See if item specified by 'path' is open.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `is_open("← Holidays/12\25\2010")`.

Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

| in  | path | – the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

- 1 - OK: item is open
- 0 - OK: item is closed
- -1 - ERROR: item was not found

See also

`Fl_Tree_Item::visible_r()`
is_scrollbar()

```cpp
int Fl_Tree::is_scrollbar ( Fl_Widget * w )
```

See if widget 'w' is one of the Fl_Tree widget's scrollbars.

Use this to skip over the scrollbars when walking the `child()` array. Example:

```cpp
for ( int i=0; i<tree->children(); i++ ) { // walk children
    Fl_Widget *w = tree->child(i);
    if ( tree->is_scrollbar(w) ) continue; // skip scrollbars
    ..do work here..
}
```

**Parameters**

| in | w | Widget to test |

**Returns**

1 if w is a scrollbar, 0 if not.

**Todo** should be const

is_selected()

```cpp
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

is_selected()

```cpp
int Fl_Tree::is_selected ( const char * path )
```
See if item specified by 'path' is selected.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `is_selected("Holidays/12\25\2010")`.

**Parameters**

| in path | – the tree item's pathname (e.g. "Flintstones/Fred") |

**Returns**

- 1 : item selected
- 0 : item deselected
- -1 : item was not found

31.138.2.43 is_vscroll_visible()

```cpp
int Fl_Tree::is_vscroll_visible ( ) const
```

See if the vertical scrollbar is currently visible.

**Returns**

1 if scrollbar visible, 0 if not.

31.138.2.44 item_clicked() [1/2]

```cpp
void Fl_Tree::item_clicked ( Fl_Tree_Item * item ) [protected]
```

Set the item that was last clicked.

Should only be used by subclasses needing to change this value. Normally `Fl_Tree` manages this value.

**Deprecated** in 1.3.3 ABI – use `callback_item()` instead.
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.

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

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
item_draw_mode()

```cpp
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

item_labelbgcolor()

```cpp
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'.

```cpp
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)

item_labelfgcolor()

```cpp
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)

item_labelfont()

```cpp
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)
item_labelsize()

```cpp
void Fl_Tree::item_labelsize ( Fl_Fontsize val )
```

Sets the default label font size used for creating new items.

To change the font size on a per-item basis, use `Fl_Tree_Item::labelsize(Fl_Fontsize)`.

item_pathname()

```cpp
int Fl_Tree::item_pathname ( char * pathname, int pathnamelen, const Fl_Tree_Item * item ) const
```

Returns '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>out</th>
<th>pathname</th>
<th>The string to use to return the pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pathnamelen</td>
<td>The maximum length of the string (including NULL). Must not be zero.</td>
</tr>
<tr>
<td>in</td>
<td>item</td>
<td>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()`

item_reselect_mode()

```cpp
Fl_Tree_Item_Reselect_Mode Fl_Tree::item_reselect_mode ( ) const
```

Returns the current item re/selection mode.

**Version**

- 1.3.1 ABI feature
31.138.2.56 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

31.138.2.57 labelmarginleft() [1/2]

int Fl_Tree::labelmarginleft ( ) const

Get the amount of white space (in pixels) that should appear to the left of the label text.

31.138.2.58 labelmarginleft() [2/2]

void Fl_Tree::labelmarginleft ( int val )

Set the amount of white space (in pixels) that should appear to the left of the label text.

31.138.2.59 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.

```c
for ( Fl_Tree_Item *item = tree->last(); item; item = tree->prev() )
    printf("Item: %s\n", item->label());
```

Returns

Last item in the tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()
31.138.2.60  last_selected_item()

\texttt{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.

\begin{verbatim}
// 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());
\end{verbatim}

Returns

The last selected item, or 0 if none.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

31.138.2.61  last_visible()

\texttt{Fl_Tree_Item * Fl_Tree::last_visible ( )}

Returns the last \texttt{open()}, visible item in the tree.

\textbf{Deprecated} in 1.3.3 – use \texttt{last_visible_item()} instead.

31.138.2.62  last_visible_item()

\texttt{Fl_Tree_Item * Fl_Tree::last_visible_item ( )}

Returns the last \texttt{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
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 |

Return the next item after 'item', or 0 if no more items.

Use this code to walk the entire tree:

```c
for ( Fl_Tree_Item *i = tree->first(); i; i = tree->next(i) )
    printf("Item: %s\n", i->label());
```

Parameters

| in | item | The item to use to find the next item. If NULL, returns 0. |

Returns

Next item in tree, or 0 if at last item.

See also

first(), next(), last(), prev()
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>first_visible_item()</td>
</tr>
<tr>
<td>dir=FL_Down:</td>
<td>first_visible_item()</td>
</tr>
</tbody>
</table>

Example use:

```
// 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

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to use to find the next item. If NULL, returns 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>Can be FL_Up or FL_Down (default=FL_Down or 'next')</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=return only open(), visible items, false=return open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

Next item in tree in the direction and visibility specified, or 0 if no more items of specified visibility in that direction.

See also

first(), last(), next(),
first_visible_item(), last_visible_item(), next_visible_item(),
first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3
31.138.2.66 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.

```c
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i, FL_Down) )
    printf("Item: %s\n", i->label());

// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
    printf("Item: %s\n", i->label());
```

Parameters

| in | item | The item above or below which we'll find the next selected item. If NULL, first() is used if FL_Down, last() if FL_Up. (default=NULL) |
| in | dir | The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default) |

Returns

The next selected item, or 0 if there are no more selected items.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

31.138.2.67 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

31.138.2.68 open() [1/2]

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 <code>callback()</code> is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - <code>callback()</code> is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - <code>callback()</code> is invoked if item changed (default), <code>callback_reason()</code> will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>
Returns

- 1 – item was opened
- 0 – item was already open, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

31.138.2.69 open() (2/2)

int Fl_Tree::open (
  const char * path,
  int docallback = 1 )

Opens the item specified by 'path'.

This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. open("Holidays/12\25\2010").

The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>

Returns

- 1 – OK: item opened
- 0 – OK: item was already open, no change
- -1 – ERROR: item was not found

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()
31.138.2.70  open_toggle()

```cpp
void Fl_Tree::open_toggle (  
    Fl_Tree_Item * item,  
    int docallback = 1  
)
```

Toggle the open state of 'item'.

Invokes the callback depending on the value of optional parameter 'docallback'. Handles calling `redraw()` if anything changed.

The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>– the item whose open state is to be toggled. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the <code>callback()</code> is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - <code>callback()</code> is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - <code>callback()</code> is invoked (default), <code>callback_reason()</code> will be either</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED</td>
</tr>
</tbody>
</table>

See also

`open(), close(), is_open(), is_close(), callback_item(), callback_reason()`

31.138.2.71  openicon() [1/2]

```cpp
Fl_Image * Fl_Tree::openicon ( ) const
```

Returns the icon to be used as the 'open' icon.

If none was set, the internal default is returned, a simple '[+]' icon.

31.138.2.72  openicon() [2/2]

```cpp
void Fl_Tree::openicon (  
    Fl_Image * val  
)
```

Sets the icon to be used as the 'open' icon.

This overrides the built in default '[+] icon.

**Parameters**

| in  | val          | – The new image, or zero to use the default [+] icon. |
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. |

Returns

Previous item in tree, or 0 if at first item.

See also

first(), next(), last(), prev()

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.

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.
31.138.2.76 resize()

```cpp
void Fl_Tree::resize (
   int X,
   int Y,
   int W,
   int H ) [virtual]
```

Resizes the Fl_Group widget and all of its children.

The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

See also
- Fl_Group::resizable(Fl_Widget*)
- Fl_Group::resizable()
- Fl_Widget::resize(int,int,int)

Reimplemented from Fl_Group.

31.138.2.77 root()

```cpp
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

31.138.2.78 root_label()

```cpp
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'.
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)

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()
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

31.138.2.82 select() [2/2]

```c
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\12\3\25\12010").

The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>
31.138 Fl_Tree Class Reference

Returns

- 1: OK: item's state was changed
- 0: OK: item was already selected, no change was made
- -1: ERROR: item was not found

31.138.2.83 select_all()

```cpp
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.

31.138.2.84 select_only()

```cpp
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.

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>selitem</th>
<th>The item to be selected. If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

Returns

The number of items whose selection states were changed, if any.

31.138.2.85 select_toggle()

```cpp
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

<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 (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

31.138.2.86 selectbox() [1/2]

```cpp
Fl_Boxtype Fl_Tree::selectbox ( ) const
```

Sets the style of box used to draw selected items.

This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme()
31.138.2.87 selectbox() [2/2]

```c
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()

31.138.2.88 selectmode() [1/2]

```c
Fl_Tree_Select Fl_Tree::selectmode ( ) const
```

Gets the tree's current selection mode.

See Fl_Tree_Select for possible values.

31.138.2.89 selectmode() [2/2]

```c
void Fl_Tree::selectmode ( Fl_Tree_Select val )
```

Sets the tree's selection mode.

See Fl_Tree_Select for possible values.

31.138.2.90 set_item_focus()

```c
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. |

31.138.2.91 show_item() [1/2]

```c
void Fl_Tree::show_item ( Fl_Tree_Item * item, int yoff )
```

Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the Fl_Tree widget's display.
For instance, yoff=0 will position the item at the top. If yoff is larger than the vertical scrollbar's limit, the value will be clipped. So if yoff=100, but scrollbar's max is 50, then 50 will be used.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to be shown. If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<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()
31.138.2.94  show_item_middle()

```cpp
void Fl_Tree::show_item_middle ( 
    Fl_Tree_Item * item )
```

Adjust the vertical scrollbar so that 'item' is in the middle of the display.

**Parameters**

| in | item | The item to be shown. If NULL, first() is used. |

31.138.2.95  show_item_top()

```cpp
void Fl_Tree::show_item_top ( 
    Fl_Tree_Item * item )
```

Adjust the vertical scrollbar so that 'item' is at the top of the display.

**Parameters**

| in | item | The item to be shown. If NULL, first() is used. |

31.138.2.96  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

31.138.2.97  showcollapse() [1/2]

```cpp
int Fl_Tree::showcollapse ( ) const
```

Returns 1 if the collapse icon is enabled, 0 if not.

**See also**

- showcollapse(int)
Class Documentation

31.138.2.98 showcollapse() [2/2]

void Fl_Tree::showcollapse (  
    int val )

Set if we should show the collapse icon or not.

If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().

Parameters

| in | val | 1: shows collapse icons (default), 0: hides collapse icons. |

31.138.2.99 showroot()

void Fl_Tree::showroot (  
    int val )

Set if the root item should be shown or not.

Parameters

| in | val | 1 – show the root item (default) 0 – hide the root item. |

31.138.2.100 sortorder()

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.

31.138.2.101 usericon() [1/2]

Fl_Image * Fl_Tree::usericon ( ) const

Returns the Fl_Image being used as the default user icon for all newly created items.

Returns zero if no icon has been set, which is the default.

31.138.2.102 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.

31.138.2.103  usericonmarginleft() [1/2]

```cpp
int Fl_Tree::usericonmarginleft ( ) const
```

Get the amount of white space (in pixels) that should appear to the left of the usericon.

31.138.2.104  usericonmarginleft() [2/2]

```cpp
void Fl_Tree::usericonmarginleft ( int val )
```

Set the amount of white space (in pixels) that should appear to the left of the usericon.

31.138.2.105  vposition() [1/2]

```cpp
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)

31.138.2.106  vposition() [2/2]

```cpp
void Fl_Tree::vposition ( int pos )
```

Sets the vertical scroll offset to position 'pos'.

The position is how many pixels of the tree are scrolled off the top edge of the screen.
Parameters

|  in | pos | The vertical position (in pixels) to scroll the tree to. |

See also

- `vposition()`, `hposition()`, `hposition(int)`

```c
31.138.2.107    widgetmarginleft() [1/2]

int Fl_Tree::widgetmarginleft ( ) const

Get the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

```c
31.138.2.108    widgetmarginleft() [2/2]

void Fl_Tree::widgetmarginleft ( int val )

Set the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

The documentation for this class was generated from the following files:

- `Fl_Tree.H`
- `Fl_Tree.cxx`

31.139    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, const char *new_label, Fl_Tree_Item *newitem)
  Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, const char *new_label)
  Add a new child to this item with the name 'new_label' and defaults from 'prefs'.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, 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, char **arr)
  Descend into the path specified by 'arr', and add a new child there.
- Fl_Tree_Item * child (int index)
  Return the child item for the given 'index'.
- const Fl_Tree_Item * child (int t) const
  Return the const child item for the given 'index'.
- int children () const
  Return the number of children this item has.
- void clear_children ()
  Clear all the children for this item.
- void close ()
  Close this item and all its children.
- void deactivate ()
  Deactivate the item; the callback() won't be invoked when clicked.
- Fl_Tree_Item * deparent (int index)
  Deparent child at index position 'pos'.
- int depth () const
  Returns how many levels deep this item is in the hierarchy.
- void deselect ()
  Disable the item's selection state.
- int deselect_all ()
  Deselect item and all its children.
- void draw (int X, int &Y, int W, Fl_Tree_Item *itemfocus, int &tree_item_xmax, int lastchild=1, int render=1)
  Draw this item and its children.
- virtual int draw_item_content (int render)
  Draw the item content.
- int event_on_collapse_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'collapse' button of this item?
- int event_on_item (const Fl_Tree_Prefs &prefs) const
  Was event anywhere on the item?
- int event_on_label (const Fl_Tree_Prefs &prefs) const
  Was event on the label() of this item?
- int event_on_user_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'user icon' of this item, if any?
- int find_child (const char *name)
  Return the index of the immediate child of this item that has the label 'name'.
- int find_child (Fl_Tree_Item *item)
  Find the index number for the specified 'item' in the current item's list of children.
- 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_child_item (const char *name)
• 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 (char **arr)
  Non-const version of Fl_Tree_Item::find_child_item(char **arr) 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_clicked (const Fl_Tree_Prefs &prefs, int yonly=0)
  Non-const version of Fl_Tree_Item::find_clicked(const Fl_Tree_Prefs&, int) const.

• const Fl_Tree_Item * find_item (char **arr) const
  Find item by descending array of 'names'.

• Fl_Tree_Item * find_item (char **arr)
  Non-const version of Fl_Tree_Item::find_item(char **names) const.

• Fl_Tree_Item (const Fl_Tree_Prefs &prefs)
  Constructor.

• Fl_Tree_Item (Fl_Tree *tree)
  Constructor.

• Fl_Tree_Item (const Fl_Tree_Item *o)
  Copy 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.

• void label (const char *val)
  Set the label to 'name'.

• const char * label () const
  Return the label.

• 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.

• void labelbgcolor (Fl_Color val)
  Set item's label background color.

• Fl_Color labelbgcolor () const
  Return item's label background text color.

• void labelcolor (Fl_Color val)
  Set item's label text color. Alias for labelfgcolor(Fl_Color).

• Fl_Color labelcolor () const
  Return item's label text color. Alias for labelfgcolor() const).

• void labelfgcolor (Fl_Color val)
  Set item's label foreground text color.

• Fl_Color labelfgcolor () const
  Return item's label foreground text color.

• void labelfont (Fl_Font val)
  Set item's label font face.

• Fl_Font labelfont () const
  Get item's label font face.

• void labelsize (Fl_Fontsize val)
  Set item's label font size.

• Fl_Fontsize labelsize () const
  Get item's label font size.

• int move (int to, int from)
  Move the item 'from' to sibling position of 'to'.

• 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_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 (Fl_Tree_Item *item)`
  Remove `item` from the current item's children.

- `int remove_child (const char *new_label)`
  Remove immediate child (and its children) by its label `name`.

- `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.

- `void swap_children (int ax, int bx)`
  Swap two of our children, given two child index values `ax` and `bx`.

- `int swap_children (Fl_Tree_Item *a, Fl_Tree_Item *b)`
  Swap two of our immediate children, given item pointers.

- `const Fl_Tree * tree () const`
  Return the tree for this item.

- `Fl_Tree * tree ()`
  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 (void *data)`
  Set a user-data value for the item.

- `void * user_data () const`
  Retrieve the user-data value that has been assigned to the item.

- `void userdeicon (Fl_Image *val)`
  Set the usericon to draw when the item is deactivated.

- `Fl_Image * userdeicon () const`
  Return the deactivated version of the user icon, if any.

- `void usericon (Fl_Image *val)`
  Set the item's user icon to an `Fl_Image`.

- `Fl_Image * usericon () const`
  Get the item's user icon as an `Fl_Image`. Returns '0' if disabled.

- `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.

- void widget (Fl_Widget ∗val)
  Assign an FLTK widget to this item.

- Fl_Widget ∗widget () const
  Return FLTK widget assigned 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)
  Return the item's 'visible' height.

- int calc_item_height (const Fl_Tree_Prefs &prefs) const
  Return the item's 'visible' height.

- void draw_horizontal_connector (int x1, int x2, int y, const Fl_Tree_Prefs &prefs)
  Internal: Horizontal connector line based on preference settings.

- void draw_vertical_connector (int x, int y1, int y2, const Fl_Tree_Prefs &prefs)
  Internal: Vertical connector line based on preference settings.

- Fl_Color drawbgcolor () const
  Returns the recommended background color used for drawing this item.

- Fl_Color drawfgcolor () const
  Returns the recommended foreground color used for drawing this item.

- void hide_widgets ()
  Internal: Hide the FLTK widget() for this item and all children.

- int is_flag (unsigned short val) const
  See if flag set. Returns 0 or 1.

- void recalc_tree ()
  Call this when our geometry is changed.

- void set_flag (unsigned short flag, int val)
  Set a flag to an on or off value. val is 0 or 1.

- void show_widgets ()
  Internal: Show the FLTK widget() for this item and all children.

31.139.1 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:

![Fl_Tree_Item dimensions](image)

**Figure 31.55 Fl_Tree_Item's internal dimensions.**

### 31.139.2 Constructor & Destructor Documentation

#### 31.139.2.1 Fl_Tree_Item()

```cpp
def Fl_Tree_Item::Fl_Tree_Item (const Fl_Tree_Prefs &prefs)
```

Constructor.

Makes a new instance of Fl_Tree_Item using defaults from 'prefs'.

**Deprecated** in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree*) for proper horizontal scrollbar behavior.

#### 31.139.2.2 Fl_Tree_Item()

```cpp
def 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
31.139.3 Member Function Documentation

31.139.3.1 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.

31.139.3.2 add() [1/4]

```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

31.139.3.3 add() [2/4]

```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
31.139.3.4 add() [3/4]

```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

31.139.3.5 add() [4/4]

```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

31.139.3.6 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
- icon() height (if not NULL)
- usericon() height (if not NULL) Does NOT include Fl_Tree::linespacing();

Returns
- maximum pixel height
31.139.3.7  child()

const Fl_Tree_Item * Fl_Tree_Item::child ( int t ) const

Return the const child item for the given 'index'.

Return const child item for the specified 'index'.

31.139.3.8  deactivate()

void Fl_Tree_Item::deactivate ( ) [inline]

Deactivate the item; the callback() won't be invoked when clicked.  
Same as activate(0)

31.139.3.9  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)

31.139.3.10  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.

31.139.3.11  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".

Generated by Doxygen
31.139.3.12 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>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X</td>
<td>Horizontal position for item being drawn</td>
</tr>
<tr>
<td>in, out</td>
<td>Y</td>
<td>Vertical position for item being drawn, returns new position for next item</td>
</tr>
<tr>
<td>in</td>
<td>W</td>
<td>Recommended width for item</td>
</tr>
<tr>
<td>in</td>
<td>itemfocus</td>
<td>The tree’s current focus item (if any)</td>
</tr>
<tr>
<td>in, out</td>
<td>tree_item_xmax</td>
<td>The tree’s running xmax (right-most edge so far). Mainly used by parent tree when render==0 to calculate tree’s max width.</td>
</tr>
<tr>
<td>in</td>
<td>lastchild</td>
<td>Is this item the last child in a subtree?</td>
</tr>
<tr>
<td>in</td>
<td>render</td>
<td>Whether or not to render the item: 0: no rendering, just calculate size w/out drawing. 1: render item as well as size calc</td>
</tr>
</tbody>
</table>

Version

1.3.3 ABI feature: modified parameters

31.139.3.13 draw_horizontal_connector()

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>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x1</td>
<td>The left hand X position of the horizontal connector</td>
</tr>
<tr>
<td>in</td>
<td>x2</td>
<td>The right hand X position of the horizontal connector</td>
</tr>
<tr>
<td>in</td>
<td>y</td>
<td>The vertical position of the horizontal connector</td>
</tr>
<tr>
<td>in</td>
<td>prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>
31.139.3.14 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

| In | `render` | 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. |
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

31.139.3.15  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>

31.139.3.16  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
31.139.3.17  drawfgcolor()

```cpp
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

31.139.3.18  find_child() \[1/2\]

```cpp
int Fl_Tree_Item::find_child (  
    const char ∗ name )
```

Return the index of the immediate child of this item that has the label 'name'.

Returns
- index of found item, or -1 if not found.

Version
- 1.3.0 release

31.139.3.19  find_child() \[2/2\]

```cpp
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.
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

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

const Fl_Tree_Item * Fl_Tree_Item::find_clicked (const Fl_Tree_Prefs & prefs, int yonly = 0) const

Find the item that the last event was over.

If 'yonly' is 1, only check event's y value, don't care about x.

Parameters
<table>
<thead>
<tr>
<th>in</th>
<th>prefs</th>
<th>The parent tree's Fl_Tree_Prefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>yonly</td>
<td>– 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X.</td>
</tr>
</tbody>
</table>
Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

31.139.3.23 \texttt{find\_item()}

\begin{verbatim}
const Fl_Tree_Item * Fl_Tree_Item::find_item ( char ** names ) const
\end{verbatim}

Find item by descending array of \texttt{name}s.
Includes self in search. Only \texttt{Fl\_Tree} should need this method. Use \texttt{Fl\_Tree::find\_item()} instead.

Returns

const item, or 0 if not found

31.139.3.24 \texttt{hide\_widgets()}

\begin{verbatim}
void Fl_Tree_Item::hide_widgets ( ) [protected]
\end{verbatim}

Internal: Hide the FLTK \texttt{widget()} for this item and all children.
Used by \texttt{close()} to hide widgets.

31.139.3.25 \texttt{insert()}

\begin{verbatim}
Fl_Tree_Item * Fl_Tree_Item::insert ( const Fl_Tree_Prefs & prefs, const char * new\_label, int pos = 0 )
\end{verbatim}

Insert a new item named \texttt{new\_label} into current item's children at a specified position \texttt{pos}.
If \texttt{pos} is out of range the new item is

\begin{itemize}
\item prepended if \texttt{pos} < 0 or
\item appended if \texttt{pos} > \texttt{item->children()}.\end{itemize}

Returns

the new item inserted

See also

\texttt{Fl\_Tree::insert()}
31.139.3.26 insert_above()

```cpp
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.

31.139.3.27 label()

```cpp
void Fl_Tree_Item::label ( 
    const char * name )
```

Set the label to 'name'.

Makes and manages an internal copy of 'name'.

31.139.3.28 label_h()

```cpp
int Fl_Tree_Item::label_h ( ) const [inline]
```

The item's label height.

Version

1.3.3

31.139.3.29 label_w()

```cpp
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
31.139.3.30 label_x()

int Fl_Tree_Item::label_x ( ) const [inline]

The item's label x position relative to the window.

Version

1.3.3

31.139.3.31 label_y()

int Fl_Tree_Item::label_y ( ) const [inline]

The item's label y position relative to the window.

Version

1.3.3

31.139.3.32 labelbgcolor() [1/2]

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.

31.139.3.33 labelbgcolor() [2/2]

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.)
31.139.3.34 move() [1/2]

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)

See also

move_above(), move_below(), move_into(), move(Fl_Tree_Item*,int,int)

31.139.3.35 move() [2/2]

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.)

See also

move_above(), move_below(), move_into(), move(int,int)
31.139.3.36  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.

See also

move_below(), move_into(), move(int,int), move(Fl_Tree_Item+,int,int)

31.139.3.37  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.

See also

move_above(), move_into(), move(int,int), move(Fl_Tree_Item+,int,int)

31.139.3.38  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.

See also

move_above(), move_below(), move(int,int), move(Fl_Tree_Item+,int,int)
31.139.3.39  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.

31.139.3.40  next_displayed()

Fl_Tree_Item * Fl_Tree_Item::next_displayed(Fl_Tree_Prefs & prefs)

Same as next_visible().

Deprecated in 1.3.3 for confusing name, use next_visible() instead

31.139.3.41  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.

31.139.3.42  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
31.139.3.43 parent()

```cpp
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.

31.139.3.44 prefs()

```cpp
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

31.139.3.45 prev()

```cpp
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).

31.139.3.46 prev_displayed()

```cpp
Fl_Tree_Item * Fl_Tree_Item::prev_displayed ( Fl_Tree_Prefs & prefs )
```

Same as prev_visible().

**Deprecated** in 1.3.3 for confusing name, use prev_visible()
31.139.47 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.

31.139.48 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.

31.139.49 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
31.139.3.50  remove_child() [1/2]

```cpp
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.

31.139.3.51  remove_child() [2/2]

```cpp
int Fl_Tree_Item::remove_child ( const char * name )
```

Remove immediate child (and its children) by its label 'name'.

If more than one item matches 'name', only the first matching item is removed.

Parameters

| in               | name          | The label name of the immediate child to remove |

Returns

0 if removed, -1 if not found.

Version

1.3.3

31.139.3.52  reparent()

```cpp
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.
Fl_Tree_Item * Fl_Tree_Item::replace (  
        Fl_Tree_Item * newitem )

Replace the current item with a new item.

The current item is destroyed if successful. No checks are made to see if an item with the same name exists.

This method can be used to, for example, install ‘custom’ items into the tree derived from Fl_Tree_Item; see draw_item_content().

Parameters

| in  | newitem | The new item to replace the current item |

Returns

newitem on success, NULL if could not be replaced.

See also

Fl_Tree_Item::draw_item_content(), Fl_Tree::root(Fl_Tree_Item*)

Version

1.3.3 ABI feature

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

| in  | olditem | The item to be found and replaced |
| in  | newitem | The new item to take the place of 'olditem' |

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

31.139.3.55  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.

31.139.3.56  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".

31.139.3.57  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.

31.139.3.58  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()`

31.139.3.59  swap_children()[1/2]

```cpp
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

| in | ax, bx | the index of the items to swap |

### 31.139.3.60  
**swap_children()** [2/2]

```cpp
int Fl_Tree_Item::swap_children (  
    Fl_Tree_Item * a,  
    Fl_Tree_Item * b  
)
```

Swap two of our immediate children, given item pointers.

Use e.g. for sorting.

This method is SLOW because it involves linear lookups.
For speed, use `swap_children(int,int)` instead.

**Parameters**

| in | a, b | The item ptrs of the two items to swap. Both must be immediate children of the current item. |

**Returns**

- 0 : OK
- -1 : failed: item 'a' or 'b' is not our child.

### 31.139.3.61  
**tree()** [1/2]

```cpp
const Fl_Tree* Fl_Tree_Item::tree ( ) const [inline]
```

Return the tree for this item.

**Version**

1.3.3

### 31.139.3.62  
**tree()** [2/2]

```cpp
Fl_Tree* Fl_Tree_Item::tree ( ) [inline]
```

Return the tree for this item.

**Version**

1.3.4
31.139.3.63  update_prev_next()

```cpp
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**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>Our index# in the parent.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Special case if index=-1: become an orphan; null out all parent/sibling associations.</td>
</tr>
</tbody>
</table>

31.139.3.64  userdeicon() [1/2]

```cpp
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:

```cpp
// 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

31.139.3.65  userdeicon() [2/2]

```cpp
Fl_Image* Fl_Tree_Item::userdeicon () const [inline]
```

Return the deactivated version of the user icon, if any.

Returns 0 if none.
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*)

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 close()ed.

int Fl_Tree_Item::w ( ) const [inline]

The entire item's width to right edge of Fl_Tree's inner width within scrollbars.

The documentation for this class was generated from the following files:

- Fl_Tree_Item.H
- Fl_Tree_Item.cxx

31.140 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 (int new_chunksize=10)**
  Constructor; creates an empty array.

- **Fl_Tree_Item_Array (const Fl_Tree_Item_Array ∗o)**
  Copy constructor. Makes new copy of array, with new instances of each item.

- **void** `insert (int pos, Fl_Tree_Item ∗new_item)`
  Insert an item at index position `pos`.

- **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** `manage_item_destroy () const`

- **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)

- **void** `remove (int index)`
  Remove the item at.

- **int** `remove (Fl_Tree_Item ∗item)`
  Remove the item from the array.

- **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.

31.140.1 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).

31.140.2 Constructor & Destructor Documentation
31.140.2.1 Fl_Tree_Item_Array()

Fl_Tree_Item_Array::Fl_Tree_Item_Array (  
    int new_chunksize = 10  )

Constructor; creates an empty array.

The optional 'chunksize' can be specified to optimize memory allocation for potentially large arrays. Default chunksize is 10.

31.140.3 Member Function Documentation

31.140.3.1 add()

void Fl_Tree_Item_Array::add (  
    Fl_Tree_Item ∗ val  )

Add an item* to the end of the array.

Assumes the item was created with 'new', and will remain allocated.. Fl_Tree_Item_Array will handle calling the item's destructor when the array is cleared or the item remove()'ed.

31.140.3.2 clear()

void Fl_Tree_Item_Array::clear (  )

Clear the entire array.

Each item will be deleted (destructors will be called), and the array will be cleared. total() will return 0.

31.140.3.3 deparent()

int Fl_Tree_Item_Array::deparent (  
    int pos  )

Deparent item at 'pos' from our list of children.

Similar to a remove() without the destruction of the item. This creates an orphaned item (still allocated, has no parent) which soon after is typically reparented elsewhere.

\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)
31.140.3.4 insert()

```c
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 `pos >= total()`, the item is appended to the array. If `pos < 0`, the item is prepended (works like `pos == 0`).

31.140.3.5 manage_item_destroy()

```c
void Fl_Tree_Item_Array::manage_item_destroy (  
    int val ) [inline]
```

Option to control if `Fl_Tree_Item_Array`'s destructor will also destroy the `Fl_Tree_Item`'s.
If set: items and item array is destroyed. If clear: only the item array is destroyed, not items themselves.

31.140.3.6 move()

```c
int Fl_Tree_Item_Array::move (  
    int to,  
    int from )
```

Move item at 'from' to new position 'to' in the array.
Due to how the moving an item shuffles the array around, a positional 'move' implies things that may not be obvious:

- When 'from' moved lower in tree, appears BELOW item that was at 'to'.
- When 'from' moved higher in tree, appears ABOVE item that was at 'to'.

Returns
0 on success, -1 on range error (e.g. if 'to' or 'from' out of range)

31.140.3.7 remove() [1/2]

```c
void Fl_Tree_Item_Array::remove (  
    int index )
```

Remove the item at.

Parameters

```
| `index` | from the array |
```

Generated by Doxygen
The item will be delete'd (if non-NULL), so its destructor will be called.

31.140.3.8 remove() [2/2]

```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.

31.140.3.9 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)

31.140.3.10 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

31.141 Fl_Tree_Prefs Class Reference

Tree widget's preferences.

#include <Fl_Tree_Prefs.H>
Public Member Functions

- **Fl_Image * closedeicon () const**
  
  Return the deactivated version of the close icon, if any.

- **Fl_Image * closeicon () const**
  
  Gets the default 'close' icon Returns the Fl_Image of the icon, or 0 if none.

- **void closeicon (Fl_Image *val)**
  
  Sets the icon to be used as the 'close' icon.

- **Fl_Color connectorcolor () const**
  
  Get the connector color used for tree connection lines.

- **void connectorcolor (Fl_Color val)**
  
  Set the connector color used for tree connection lines.

- **Fl_Tree_Connector connectorstyle () const**
  
  Get the connector style.

- **void connectorstyle (Fl_Tree_Connector val)**
  
  Set the connector style.

- **void connectorstyle (int val)**
  
  Set the connector style [integer].

- **int connectorwidth () const**
  
  Get the tree connection line's width.

- **void connectorwidth (int val)**
  
  Set the tree connection line's width.

- **void do_item_draw_callback (Fl_Tree_Item *o) const**

- **Fl_Tree_Prefs ()**
  
  **Fl_Tree_Prefs constructor.**

- **void item_draw_callback (Fl_Tree_Item_Draw_Callback *cb, void *data=0)**
  
  **Fl_Tree_Item_Draw_Callback * item_draw_callback () 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)**
  
  Set the label's size in pixels to val.

- **Fl_Tree_Item_Reselect_Mode item_reselect_mode () const**
  
  Returns the current item re/selection mode.

- **void item_reselect_mode (Fl_Tree_Item_Reselect_Mode mode)**

Generated by Doxygen
Sets the item re/selection mode.

- **`Fl_Color labelbgcolor () const`**
  
  Obsolete: Get the default label background color. Please use `item_labelbgcolor()` instead.

- **`void labelbgcolor (Fl_Color val)`**
  
  Obsolete: Set the default label background color. Please use `item_labelbgcolor(Fl_Color)` instead.

- **`Fl_Color labelfgcolor () const`**
  
  Obsolete: Get the default label foreground color. Please use `item_labelfgcolor()` instead.

- **`void labelfgcolor (Fl_Color val)`**
  
  Obsolete: Set the default label foreground color. Please use `item_labelfgcolor(Fl_Color)` instead.

- **`Fl_Font labelfont () const`**
  
  Obsolete: Return the label's font. Please use `item_labelfont()` instead.

- **`void labelfont (Fl_Font val)`**
  
  Obsolete: Set the label's font to `val`. Please use `item_labelfont(Fl_Font)` instead.

- **`int labelmarginleft () const`**
  
  Get the label's left margin value in pixels.

- **`void labelmarginleft (int val)`**
  
  Set the label's left margin value in pixels.

- **`Fl_Fontsize labelsize () const`**
  
  Obsolete: Return the label's size in pixels. Please use `item_labelsize()` instead.

- **`void labelsize (Fl_Fontsize val)`**
  
  Obsolete: Set the label's size in pixels to `val`. Please use `item_labelsize(Fl_Fontsize)` instead.

- **`int linespacing () const`**
  
  Get the line spacing value in pixels.

- **`void linespacing (int val)`**
  
  Set the line spacing value in pixels.

- **`int marginbottom () const`**
  
  Get the bottom margin's value in pixels. This is the extra distance the vertical scroller lets you travel.

- **`void marginbottom (int val)`**
  
  Set the bottom margin's value in pixels.

- **`int marginleft () const`**
  
  Get the left margin's value in pixels.

- **`void marginleft (int val)`**
  
  Set the left margin's value in pixels.

- **`int margintop () const`**
  
  Get the top margin's value in pixels.

- **`void margintop (int val)`**
  
  Set the top margin's value in pixels.

- **`int openchild_marginbottom () const`**
  
  Get the margin below an open child in pixels.

- **`void openchild_marginbottom (int val)`**
  
  Set the margin below an open child in pixels.

- **`Fl_Imagex opendeicon () const`**
  
  Return the deactivated version of the open icon, if any.

- **`Fl_Imagex openicon () const`**
  
  Get the current default 'open' icon.

- **`void openicon (Fl_Imagex +val)`**
  
  Sets the default icon to be used as the 'open' icon when items are added 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`. 

Generated by Doxygen
• **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.

### 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.

### Member Function Documentation

Generated by Doxygen
31.141.2.1 closedeicon()

Fl_Image* Fl_Tree_Prefs::closedeicon() const [inline]

Return the deactivated version of the close icon, if any.

Returns 0 if none.

31.141.2.2 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. |

31.141.2.3 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.

31.141.2.4 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.

31.141.2.5 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().
31.141.2.6 marginbottom() [1/2]

```cpp
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.

31.141.2.7 marginbottom() [2/2]

```cpp
void Fl_Tree_Prefs::marginbottom (  
    int val ) [inline]
```

Set the bottom margin's value in pixels This is the extra distance the vertical scroller lets you travel.

31.141.2.8 opendeicon()

```cpp
Fl_Image* Fl_Tree_Prefs::opendeicon ( ) const [inline]
```

Return the deactivated version of the open icon, if any.
Returns 0 if none.

31.141.2.9 openicon() [1/2]

```cpp
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.

31.141.2.10 openicon() [2/2]

```cpp
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. |

Generated by Doxygen
31.141.2.11 selectmode()

```c
void Fl_Tree_Prefs::selectmode (
    Fl_Tree_Select val ) [inline]
```

Set the selection mode used for the tree to \texttt{val}.

This affects how items in the tree are selected when clicked on and dragged over by the mouse. See \texttt{Fl_Tree_Select} for possible values.

31.141.2.12 showcollapse()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>1: shows collapse icons (default), 0: hides collapse icons.</th>
</tr>
</thead>
</table>

31.141.2.13 showroot()

```c
void Fl_Tree_Prefs::showroot (
    int val ) [inline]
```

Set if the root item should be shown or not.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>1 – show the root item (default), 0 – hide the root item.</th>
</tr>
</thead>
</table>

31.141.2.14 sortorder()

```c
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 \texttt{Fl_Tree_Sort} for possible values.
### Fl_Tree_Prefs::userdeicon()

*Fl_Image* `Fl_Tree_Prefs::userdeicon ( ) const [inline]`

Returns 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

### Fl_Valuator Class Reference

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.

```
#include <Fl_Valuator.H>
```

Inheritance diagram for Fl_Valuator:

```
Fl_Widget
    Fl_Valuator
        Fl_Adjuster
        Fl_Counter
        Fl_Dial
        Fl_Roller
        Fl_Slider
        Fl_Value_Input
        Fl_Value_Output
        Fl_Simple_Counter
        Fl_Fill_Dial
        Fl_Line_Dial
        Fl_Fill_Slider
        Fl_Hor_Fill_Slider
        Fl_Hor_Nice_Slider
        Fl_Hor_Slider
        Fl_Nice_Slider
        Fl_Scrollbar
        Fl_Value_Slider
```

#### 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)`

---

Generated by Doxygen
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.

- **void step (int a)**
  
  See double Fl_Valuator::step() const.

- **void step (double a, int b)**
  
  See double Fl_Valuator::step() const.

- **void step (double s)**
  
  See double Fl_Valuator::step() const.

- **double step ()** const
  
  Gets or sets the step value.

- **double value ()** const
  
  Gets the floating point(double) value.

- **int value (double)**
  
  Sets the current value.

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.
Additional Inherited Members

31.142.1 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:

![Figure 31.56 Valuators derived from Fl_Valuators](image)

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.

31.142.2 Constructor & Destructor Documentation

31.142.2.1 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.
31.142.3 Member Function Documentation

31.142.3.1 bounds()

```cpp
void Fl_Valuator::bounds ( double a, double b ) [inline]
```

Sets the minimum (a) and maximum (b) values for the valuator widget.

31.142.3.2 clamp()

```cpp
double Fl_Valuator::clamp ( double v )
```

Clamps the passed value to the valuator range.

31.142.3.3 format()

```cpp
int Fl_Valuator::format ( char * buffer ) [virtual]
```

Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

The actual format used depends on the current step value. If the step value has been set to zero then a %g format is used. If the step value is non-zero, then a %.f format is used, where the precision is calculated to show sufficient digits for the current step value. An integer step value, such as 1 or 1.0, gives a precision of 0, so the formatted value will appear as an integer.

This method is used by the Fl_Valuator__ group of widgets to format the current value into a text string. The return value is the length of the formatted text. The formatted value is written into buffer. buffer should have space for at least 128 bytes.

You may override this function to create your own text formatting.

31.142.3.4 handle_drag()

```cpp
void Fl_Valuator::handle_drag ( double v ) [protected]
```

Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
31.142.3.5 handle_release()

```cpp
void Fl_Valuator::handle_release ( ) [protected]
```

Called after an FL_WHEN_RELEASE event is received and before the callback.

31.142.3.6 increment()

```cpp
double Fl_Valuator::increment (  
    double v,  
    int n)
```

Adds $n$ times the step value to the passed value.

If step was set to zero it uses \(\text{fabs(maximum()) - minimum()) / 100.}\)

31.142.3.7 maximum() [1/2]

```cpp
double Fl_Valuator::maximum ( ) const [inline]
```

Gets the maximum value for the valuator.

31.142.3.8 maximum() [2/2]

```cpp
void Fl_Valuator::maximum (  
    double a) [inline]
```

Sets the maximum value for the valuator.

31.142.3.9 minimum() [1/2]

```cpp
double Fl_Valuator::minimum ( ) const [inline]
```

Gets the minimum value for the valuator.

31.142.3.10 minimum() [2/2]

```cpp
void Fl_Valuator::minimum (  
    double a) [inline]
```

Sets the minimum value for the valuator.
31.142.3.11  precision()

void Fl_Valuator::precision (  
    int digits  )

Sets the step value to $1.0 / 10^{\text{digits}}$.

Precision $\text{digits}$ is limited to 0...9 to avoid internal overflow errors. Values outside this range are clamped.

Note

For negative values of $\text{digits}$ the step value is set to $A = 1.0$ and $B = 1$, i.e. $1.0/1 = 1$.

31.142.3.12  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 call \texttt{redraw()} the widget after changing the range.

31.142.3.13  round()

double Fl_Valuator::round (  
    double v  )

Round the passed value to the nearest step increment.

Does nothing if step is zero.

31.142.3.14  set_value()

void Fl_Valuator::set_value (  
    double v )  [inline],  [protected]

Sets the current floating point value.
31.142.3.15  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.

31.142.3.16  value() [1/2]

double Fl_Valuator::value ( ) const [inline]

Gets the floating point(double) value.

See int value(double)

31.142.3.17  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).

The documentation for this class was generated from the following files:

- Fl_Valuator.H
- Fl_Valuator.cxx

31.143  Fl_Value_Input Class Reference

The Fl_Value_Input widget displays a numeric value.

#include <Fl_Value_Input.H>

Inheritance diagram for Fl_Value_Input:

```
Fl_Widget
   |
   v
Fl_Valuator
   |
   v
Fl_Value_Input
```

Generated by Doxygen
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.

- `void soft (char s)`
  
  See void `Fl_Value_Input::soft(char s)`

- `char soft () const`
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- `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 Attributes

- `Fl_Input input`

Protected Member Functions

- `void draw ()`
  
  Draws the widget.
Additional Inherited Members

31.143.1 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.

See also

Fl_Widget::shortcut_label(int)

31.143.2 Constructor & Destructor Documentation

31.143.2.1 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.

31.143.3 Member Function Documentation
31.143.3.1 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.

31.143.3.2 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.

31.143.3.3 draw()

`void Fl_Value_Input::draw()` [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

31.143.3.4 handle()

`int Fl_Value_Input::handle(int event)` [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
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.

31.143.3.5   resize()

```cpp
void Fl_Value_Input::resize (  
    int x,  
    int y,  
    int w,  
    int h ) [virtual]
```

Changes the size or position of the widget.

This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the `redraw()` method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.

Some window managers under X11 call `resize()` a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.

position(X, Y) is a shortcut for `resize(X, Y, w(), h())`, and size(W, H) is a shortcut for `resize(x(), y(), W, H)`.

Parameters

| in | x,y | new position relative to the parent window |
| in | w,h | new size |

See also

- `position(int,int)`, `size(int,int)`

Reimplemented from Fl_Widget.
31.143.6  shortcut() [1/2]

```cpp
int Fl_Value_Input::shortcut ( ) const [inline]
```

Returns the current shortcut key for the Input.

See also

- `Fl_Value_Input::shortcut(int)`

31.143.7  shortcut() [2/2]

```cpp
void Fl_Value_Input::shortcut ( int s ) [inline]
```

Sets the shortcut key to `s`.

Setting this overrides the use of ‘&’ in the `label()`. The value is a bitwise OR of a key and a set of shift flags, for example `FL_ALT | 'a', FL_ALT | (FL_F + 10), or just 'a'. A value of 0 disables the shortcut.

The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.

The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a “don't care” setting).

31.143.8  soft()

```cpp
char Fl_Value_Input::soft ( ) const [inline]
```

If “soft” is turned on, the user is allowed to drag the value outside the range.

If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value. The default is true.

31.143.9  textcolor() [1/2]

```cpp
Fl_Color Fl_Value_Input::textcolor ( ) const [inline]
```

Gets the color of the text in the value box.

31.143.10  textcolor() [2/2]

```cpp
void Fl_Value_Input::textcolor ( Fl_Color n ) [inline]
```

Sets the color of the text in the value box.
31.143.3.11 textfont() [1/2]

`Fl_Font Fl_Value_Input::textfont() const [inline]`

Gets the typeface of the text in the value box.

31.143.3.12 textfont() [2/2]

`void Fl_Value_Input::textfont (Fl_Font s) [inline]`

Sets the typeface of the text in the value box.

31.143.3.13 textsize() [1/2]

`Fl_Fontsize Fl_Value_Input::textsize() const [inline]`

Gets the size of the text in the value box.

31.143.3.14 textsize() [2/2]

`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.hxx`

31.144 Fl_Value_Output Class Reference

The `Fl_Value_Output` widget displays a floating point value.

`#include <Fl_Value_Output.H>`

Inheritance diagram for `Fl_Value_Output`:

```
+------------------------+             +--------------------------+
| Fl_Widget              | <- | Fl_Valuator             |
|                        |     |                         |
|                        |     | Fl_Value_Output         |
```

Generated by Doxygen
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.

- **void soft (uchar s)**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **uchar soft () const**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **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)**

Protected Member Functions

- **void draw ()**
  
  Draws the widget.

Additional Inherited Members

31.144.1 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.

![Fl_Value_Output](image)

**Figure 31.58 Fl_Value_Output**

31.144.2 Constructor & Destructor Documentation
31.144 Fl_Value_Output Class Reference

31.144.2 Fl_Value_Output()

```cpp
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.

31.144.3 Member Function Documentation

31.144.3.1 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 = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw();  // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

31.144.3.2 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></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`.

31.144.3.3 `soft()` [1/2]

```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.

31.144.3.4 `soft()` [2/2]

```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.

31.144.3.5 `textcolor()` [1/2]

```cpp
Fl_Color Fl_Value_Output::textcolor ( ) const [inline]
```

Sets the color of the text in the value box.

31.144.3.6 `textcolor()` [2/2]

```cpp
void Fl_Value_Output::textcolor ( Fl_Color s ) [inline]
```

Gets the color of the text in the value box.
31.144.7 textfont() [1/2]

`Fl_Font Fl_Value_Output::textfont ( ) const [inline]`

Gets the typeface of the text in the value box.

31.144.8 textfont() [2/2]

`void Fl_Value_Output::textfont (Fl_Font s) [inline]`

Sets the typeface of the text in the value box.

31.144.9 textsize()

`Fl_Fontsize Fl_Value_Output::textsize ( ) const [inline]`

Gets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Output.H
- Fl_Value_Output.cxx

31.145 Fl_Value_Slider Class Reference

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

#include <Fl_Value_Slider.H>

Inheritance diagram for Fl_Value_Slider:

```
  Fl_Widget
   |  Fl_Valuator
   |  Fl_Slider
   |  Fl_Value_Slider
   |  Fl_Hor_Value_Slider
```

Generated by Doxygen
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.

Protected Member Functions

- `void draw ()`
  Draws the widget.

Additional Inherited Members

31.145.1 Detailed Description

The `Fl_Value_Slider` widget is a `Fl_Slider` widget with a box displaying the current value.
31.145.2.1 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.

31.145.3 Member Function Documentation

31.145.3.1 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:

```c
Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_ScrollView
s->draw();              // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Slider.

31.145.3.2 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

| 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_Slider.

31.145.3.3 textcolor() [1/2]

Fl_Color Fl_Value_Slider::textcolor ( ) const [inline]

Gets the color of the text in the value box.

31.145.3.4 textcolor() [2/2]

void Fl_Value_Slider::textcolor ( Fl_Color s ) [inline]

Sets the color of the text in the value box.

31.145.3.5 textfont() [1/2]

Fl_Font Fl_Value_Slider::textfont ( ) const [inline]

Gets the typeface of the text in the value box.

31.145.3.6 textfont() [2/2]

void Fl_Value_Slider::textfont ( Fl_Font s ) [inline]

Sets the typeface of the text in the value box.
31.145.7  `textsize()` [1/2]

```cpp
Fl_Fontsize Fl_Value_Slider::textsize ( ) const [inline]
```

Gets the size of the text in the value box.

31.145.8  `textsize()` [2/2]

```cpp
void Fl_Value_Slider::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_Slider.H
- Fl_Value_Slider.cxx

31.146  Fl_Widget Class Reference

`Fl_Widget` is the base class for all widgets in FLTK.

```cpp
#include <Fl_Widget.H>
```

Inheritance diagram for Fl_Widget:
Public Member Functions

- void _clear_fullscreen ()
- void _set_fullscreen ()
- void activate ()
  Activates the widget.
- unsigned int active () const
  Returns whether the widget is active.
- int active_r () const
  Returns whether the widget and all of its parents are active.
- Fl_Align align () const
  Gets the label alignment.
- void align (Fl_Align alignment)
  Sets the label alignment.
- long argument () const
  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)
  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual 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, 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)`
  Internal use only.

- `void deactivate ()`
  Deactivates the widget.

- `Fl_Image * deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.

- `const Fl_Image * deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.

- `void deimage (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.

- `void do_callback ()`
  Calls the widget callback function with default arguments.

- `void do_callback (Fl_Widget *widget, long arg)`
  Calls the widget callback function with arbitrary arguments.

- `void do_callback (Fl_Widget *widget, void *arg=0)`
  Calls the widget callback function with arbitrary arguments.

- `virtual void draw ()=0`
  Draws the widget.

- `void draw_label (int, int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- `int h () const`
  Gets the widget height.

- `virtual int handle (int event)`
  Handles the specified event.

- `virtual void hide ()`
  Makes a widget invisible.
• `Fl_Image * image ()`
  Gets the image that is used as part of the widget label when in the active state.

• `const Fl_Image * image () const`
  Gets the image that is used as part of the widget label when in the active state.

• `void image (Fl_Image *img)`
  Sets the image to use as part of the widget label when in the active state.

• `void image (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the active state.

• `int 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.

• void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.

• int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.

• virtual void show ()

  Makes a widget visible.

• void size (int W, int H)

  Changes the size of the widget.

• int take_focus ()

  Gives the widget the keyboard focus.

• unsigned int takesevents () const

  Returns if the widget is able to take events.

• int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.

• const char ∗ tooltip () const

  Gets the current tooltip text.

• void tooltip (const char ∗text)

  Sets the current tooltip text.

• Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.

• Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const

  Gets the widget type.

• void type (uchar t)

  Sets the widget type.

• int use_accents_menu ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void ∗ user_data () const

  Gets the user data for this widget.

• void user_data (void ∗v)

  Sets the user data for this widget.

• unsigned int visible () const

  Returns whether a widget is visible.

• void visible_focus (int v)

  Modifies keyboard focus navigation.

• 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.

Static Public Member Functions

• static void default_callback (Fl_Widget * widget, void * data)
  The default callback for all widgets that don’t set a callback.

• static unsigned int label_shortcut (const char * t)
  Returns the Unicode value of the ‘&x’ shortcut in a given text.

• static int test_shortcut (const char * t, const bool 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, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, 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

Generated by Doxygen
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.

**Friends**

- **void** Fl::focus (Fl_Widget *)
- class Fl_Group

### 31.146.1 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.

### 31.146.2 Member Enumeration Documentation

#### 31.146.2.1 anonymous enum

anonymous enum [protected]

flags possible values enumeration.

See activate(), output(), visible(), changed(), set_visible_focus()
31.146.3 Constructor & Destructor Documentation

31.146.3.1 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>

31.146.3.2 ~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

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.

31.146.4 Member Function Documentation

31.146.4.1 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

active(), active_r(), deactivate()

31.146.4.2 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()

31.146.4.3 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()

31.146.4.4 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

31.146.4.5 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

### 31.146.4.6 argument() [1/2]

```cpp
long Fl_Widget::argument ( ) const [inline]
```

Gets the current user data (long) argument that is passed to the callback function.

**Note**

On platforms with `sizeof(long) < sizeof(void*)`, particularly on Windows 64-bit platforms, this method can truncate stored addresses (void*) to the size of a long value. Use with care and only if you are sure that the stored user_data value fits in a long value because it was stored with argument(long) or another method using only long values. You may want to use user_data() instead.

See also

user_data()

**Todo** [Internal] The user_data value must be implemented using `fl_intptr_t` or similar to avoid 64-bit platform incompatibilities.

### 31.146.4.7 argument() [2/2]

```cpp
void Fl_Widget::argument ( long v ) [inline]
```

Sets the current user data (long) argument that is passed to the callback function.

See also

argument()

### 31.146.4.8 as_gl_window()

```cpp
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.

31.146.4.9 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 *arg) {
  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.
31.146.4.10  as_window()

virtual Fl_Window* Fl_Widget::as_window() [inline], [virtual]

Returns an Fl_Window pointer if this widget is an Fl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer to access its children or other Fl_Window-specific methods.

Return values

NULL if this widget is not derived from Fl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented in Fl_Window.

31.146.4.11  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

31.146.4.12  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

31.146.4.13 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

31.146.4.14 callback() [2/5]

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

| in  | cb   | new callback |
| in  | p    | user data   |

31.146.4.15 callback() [3/5]

void Fl_Widget::callback ( 
    Fl_Callback * cb ) [inline]

Sets the current callback function for the widget.

Each widget has a single callback.
### 31.146.4.16 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>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
</tr>
</tbody>
</table>

### 31.146.4.17 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>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
</tr>
</tbody>
</table>

### 31.146.4.18 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.
Note

do_callback() turns this flag off after the callback.

Return values

| 0 | if the value did not change |

See also

- set_changed(), clear_changed()
- do_callback(Fl_Widget *widget, void *data)

31.146.4.19 clear_active()

void Fl_Widget::clear_active ( ) [inline]

Marks the widget as inactive without sending events or changing focus.
This is mainly for specialized use, for normal cases you want deactivate().

See also

- deactivate()

31.146.4.20 clear_changed()

void Fl_Widget::clear_changed ( ) [inline]

Marks the value of the widget as unchanged.

See also

- changed(), set_changed()

31.146.4.21 clear_damage()

void Fl_Widget::clear_damage ( uchar c = 0 ) [inline]

Clears or sets the damage flags.
Damage flags are cleared when parts of the widget drawing is repaired.
The optional argument c specifies the bits that are set after the call (default: 0) and not the bits that are cleared!

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()

31.146.4.22 clear_output()

void Fl_Widget::clear_output ( ) [inline]

Sets a widget to accept input.

See also

set_output(), output()

31.146.4.23 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.

31.146.4.24 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)
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)

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)

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.
31.146 Fl_Widget Class Reference

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bg</th>
<th>background color</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sel</td>
<td>selection color</td>
</tr>
</tbody>
</table>

See also

color(unsigned), selection_color(unsigned)

31.146.4.28 color2() [1/2]

Fl_Color Fl_Widget::color2 ( ) const [inline]

For back compatibility only.

**Deprecated** Use selection_color() instead.

31.146.4.29 color2() [2/2]

void Fl_Widget::color2 ( unsigned a ) [inline]

For back compatibility only.

**Deprecated** Use selection_color(unsigned) instead.

31.146.4.30 contains()

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.

31.146.4.31 copy_label()

```c
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()

31.146.4.32 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()
31.146.4.33  damage() [1/3]

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)

31.146.4.34  damage() [2/3]

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)

31.146.4.35  damage() [3/3]

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)

31.146.4.36 damage_resize()

```c
int Fl_Widget::damage_resize ( int x, int y, int w, int h )
```

Internal use only.

31.146.4.37 deactivate()

```c
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()

31.146.4.38 default_callback()

```c
void Fl_Widget::default_callback ( Fl_Widget * widget, void * data ) [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(). This is the default for all widgets if you don't set a callback.

You can avoid the overhead of this default handling if you set the callback to NULL explicitly.

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>widget</th>
<th>the Fl_Widget given to the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>user data associated with that callback</td>
</tr>
</tbody>
</table>

See also

callback(), Fl::readqueue()
do_callback(Fl_Widget *widget, void *data)

31.146.4.39 deimage() [1/4]

Fl_Image* Fl_Widget::deimage ( ) [inline]

Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

31.146.4.40 deimage() [2/4]

const Fl_Image* Fl_Widget::deimage ( ) const [inline]

Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

31.146.4.41 deimage() [3/4]

void Fl_Widget::deimage ( 
    Fl_Image * img ) [inline]

Sets the image to use as part of the widget label when in the inactive state.

Parameters

| in  | img    | the new image for the deactivated widget |
Note

The caller is responsible for making sure `img` is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

### deimage() [4/4]

```c
void Fl_Widget::deimage (Fl_Image & img) [inline]
```

Sets the image to use as part of the widget label when in the inactive state.

**Parameters**

| in | `img` | the new image for the deactivated widget |

**See also**

- `void deimage(Fl_Image* img)`

### do_callback() [1/3]

```c
void Fl_Widget::do_callback () [inline]
```

Calls the widget callback function with default arguments.

This is the same as calling

```c
do_callback(this, user_data());
```

**See also**

- `callback`
- `do_callback(Fl_Widget *widget, void *data)`

### do_callback() [2/3]

```c
void Fl_Widget::do_callback (Fl_Widget * widget, long arg) [inline]
```

Calls the widget callback function with arbitrary arguments.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>in</td>
</tr>
</tbody>
</table>

See also

callback()
do_callback(Fl_Widget *widget, void *data)

do_callback() [3/3]

void Fl_Widget::do_callback (  
    Fl_Widget * widget,  
    void * arg = 0 )

Calls the widget callback function with arbitrary arguments.

All overloads of do_callback() call this method. It does nothing if the widget's callback is NULL. It clears the widget's changed flag after the callback was called unless the callback is the default callback. Hence it is not necessary to call clear_changed() after calling do_callback() in your own widget's handle() method.

Note

It is legal to delete the widget in the callback (i.e. in user code), but you must not access the widget in the handle() method after calling do_callback() if the widget was deleted in the callback. We recommend to use Fl_Widget_Tracker to check whether the widget was deleted in the callback.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>in</td>
</tr>
</tbody>
</table>

See also

default_callback()
callback()
class Fl_Widget_Tracker

draw() [3/3]

virtual void Fl_Widget::draw ( ) [pure virtual]

Draws the widget.

Generated by Doxygen
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```


### 31.146.4.47 draw_box() [1/2]

```cpp
void Fl_Widget::draw_box ( Fl_Boxtype t, Fl_Color c ) const [protected]
```

Draws a box of type `t`, of color `c` at the widget's position and size.

### 31.146.4.48 draw_box() [2/2]

```cpp
void Fl_Widget::draw_box ( Fl_Boxtype t, int X, int Y, int W, int H, Fl_Color c ) const [protected]
```

Draws a box of type `t`, of color `c` at the position `X,Y` and size `W,H`.

### 31.146.4.49 draw_focus()

```cpp
void Fl_Widget::draw_focus ( Fl_Boxtype B, int X, int Y, int W, int H ) const [protected]
```

Draws a focus box for the widget at the given position and size.
31.146.4.50  draw_label() [1/3]

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.

31.146.4.51  draw_label() [2/3]

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

31.146.4.52  draw_label() [3/3]

void Fl_Widget::draw_label (  
    int X,  
    int Y,  
    int W,  
    int H,  
    Fl_Align a ) const

Draws the label in an arbitrary bounding box with an arbitrary alignment.
Anybody can call this to force the label to draw anywhere.

31.146.4.53  h() [1/2]

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.

31.146.4.54  h() [2/2]

int Fl_Widget::h ( ) const [inline]

Gets the widget height.
Returns
    the height of the widget in pixels.
31.146.4.55 handle()

```cpp
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**

*event* the kind of event received

**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


31.146.4.56 hide()

```cpp
void Fl_Widget::hide ( ) [virtual]
```

Makes a widget invisible.

**See also**

show(), visible(), visible_r()

Reimplemented in Fl_Window, Fl_Browser, Fl_GL_Window, Fl_Overlay_Window, Fl_Double_Window, and Fl_Menu_Window.
Fl_Widget::image() [1/4]

Fl_Image* Fl_Widget::image () [inline]

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

Fl_Widget::image() [2/4]

const Fl_Image* Fl_Widget::image () const [inline]

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

Fl_Widget::image() [3/4]

void Fl_Widget::image ( Fl_Image* img ) [inline]

Sets the image to use as part of the widget label when in the active state.

Parameters

| in   | img         | the new image for the label |

Note

The caller is responsible for making sure img is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

Fl_Widget::image() [4/4]

void Fl_Widget::image ( Fl_Image & img ) [inline]

Sets the image to use as part of the widget label when in the active state.
Parameters

| in | *img* | the new image for the label |

See also

`void image(Fl_Image* img)`

### 31.146.4.61 inside()

```cpp
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()`

### 31.146.4.62 is_label_copied()

```cpp
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()`.

### 31.146.4.63 label () [1/3]

```cpp
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 *)

31.146.4.64  label() [2/3]

void Fl_Widget::label (  
    const char * text )

Sets the current label pointer.

The label is shown somewhere on or next to the widget. See Labels and Label Types for details about what can be put in a label. The passed pointer is stored unchanged in the widget (the string is not copied), so if you need to set the label to a formatted value, make sure the buffer is static, global, or allocated. The copy_label() method can be used to make a copy of the label string automatically.

Parameters

| in text | pointer to new label text |

See also

copy_label()

31.146.4.65  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)

31.146.4.66  label_shortcut()

unsigned int Fl_Widget::labelShortcut (  
    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.

31.146.4.67  labelcolor() [1/2]

\texttt{Fl\_Color Fl\_Widget::labelcolor () const [inline]}

Gets the label color.

The default color is \texttt{FL\_FOREGROUND\_COLOR}.

Returns

the current label color

31.146.4.68  labelcolor() [2/2]

\texttt{void Fl\_Widget::labelcolor ( Fl\_Color c ) [inline]}

Sets the label color.

The default color is \texttt{FL\_FOREGROUND\_COLOR}.

Parameters

\texttt{in \ c \ the new label color}

31.146.4.69  labelfont() [1/2]

\texttt{Fl\_Font Fl\_Widget::labelfont () const [inline]}

Generated by Doxygen
Gets the font to use.

Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Returns

- current font used by the label

See also

Fl_Font

---

Sets the font to use.

Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Parameters

- **in f** the new font for the label

See also

Fl_Font

---

Gets the font size in pixels.

The default size is 14 pixels.

Returns

- the current font size

---

Sets the font size in pixels.
Parameters

| in   | pix  | the new font size |

See also

Fl_Fontsize labelsize()

---

31.146.4.73 labeltype() [1/2]

Fl_Labeltype Fl_Widget::labeltype ( ) const [inline]

Gets the label type.

Returns

the current label type.

See also

Fl_Labeltype

---

31.146.4.74 labeltype() [2/2]

void Fl_Widget::labeltype ( Fl_Labeltype a ) [inline]

Sets the label type.

The label type identifies the function that draws the label of the widget. This is generally used for special effects such as embossing or for using the label() pointer as another form of data such as an icon. The value FL_NORMAL_LABEL prints the label as plain text.

Parameters

| in   | a   | new label type |

See also

Fl_Labeltype
31.146.4.75  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)

31.146.4.76  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
    0 if the widget is used for input and output

See also
    set_output(), clear_output()

31.146.4.77  parent()[1/2]

Fl_Group* Fl_Widget::parent ( ) const [inline]

Returns a pointer to the parent widget.

Usually this is a Fl_Group or Fl_Window.

Return values
    NULL if the widget has no parent
See also

\[ \text{Fl\_Group::add(Fl\_Widget*)} \]

---

31.146.4.78 parent() [2/2]

void Fl_Widget::parent ( 
    Fl_Group \ast \ p \ ) [inline]

Internal use only - "for hacks only".

It is **STRONGLY recommended** not to use this method, because it short-circuits \texttt{Fl\_Group}'s normal widget adding and removing methods, if the widget is already a child widget of another \texttt{Fl\_Group}.

Use \texttt{Fl\_Group::add(Fl\_Widget*)} and/or \texttt{Fl\_Group::remove(Fl\_Widget*)} instead.

---

31.146.4.79 position()

void Fl_Widget::position ( 
    int \ X, 
    int \ Y \ ) [inline]

Repositions the window or widget.

\textit{position(X, Y) is a shortcut for resize(X, Y, w(), h()).}

**Parameters**

| in | X, Y | new position relative to the parent window |

See also

\[ \text{resize(int,int,int, int), size(int,int)} \]

---

31.146.4.80 redraw()

void Fl_Widget::redraw ( )

Schedules the drawing of the widget.

Marks the widget as needing its \texttt{draw()} routine called.

---

31.146.4.81 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.
31.146.4.82 resize()

```cpp
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**

<table>
<thead>
<tr>
<th>in</th>
<th>x, y</th>
<th>new position relative to the parent window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w, h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`


31.146.4.83 selection_color()[1/2]

```cpp
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)`

31.146.4.84 selection_color()[2/2]

```cpp
void Fl_Widget::selection_color ( Fl_Color a ) [inline]
```

Sets the selection color.

The selection color is defined for Forms compatibility and is usually used to color the widget when it is selected, although some widgets use this color for other purposes. You can set both colors at once with `color(Fl_Color bg, Fl_Color sel)`.
Parameters

- **in** `a` the new selection color

See also

- `selection_color()`, `color(Fl_Color, Fl_Color)`

31.146.4.85 set_active()

```cpp
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()`

31.146.4.86 set_changed()

```cpp
void Fl_Widget::set_changed ( ) [inline]
```

Marks the value of the widget as changed.

See also

- `changed()`, `clear_changed()`

31.146.4.87 set_output()

```cpp
void Fl_Widget::set_output ( ) [inline]
```

Sets a widget to output only.

See also

- `output()`, `clear_output()`
31.146.4.88  set_visible()

```c
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.

31.146.4.89  set_visible_focus()

```c
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)`

31.146.4.90  shortcut_label() [1/2]

```c
void Fl_Widget::shortcut_label (int value ) [inline]
```

Sets whether the widget's label uses '&' to indicate shortcuts.

By default, all objects of classes `Fl_Menu_` (and derivatives), `Fl_Button` (and derivatives), `Fl_Text_Display`, `Fl_Value_Input`, and `Fl_Input_` (and derivatives) use character '&' in their label, unless '&' is repeated, to indicate shortcuts: '&' does not appear in the drawn label, the next character after '&' in the label is drawn underlined, and typing this character triggers the corresponding menu window, button, or other widget. If the label contains 2 consecutive '&', only one is drawn and the next character is not underlined and not used as a shortcut. If `value` is set to 0, all these labels don't process character '&' as indicating a shortcut: '&' is drawn in the label, the next character is not underlined and does not define a shortcut.

31.146.4.91  shortcut_label() [2/2]

```c
int Fl_Widget::shortcut_label ( ) const [inline]
```

Returns whether the widget's label uses '&' to indicate shortcuts.

See also

- `void shortcut_label(int value)`

Generated by Doxygen
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()


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

<table>
<thead>
<tr>
<th>in</th>
<th>W/H</th>
<th>new size</th>
</tr>
</thead>
</table>

See also
  position(int,int), resize(int,int,int,int)

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.
31.146 Fl_Widget Class Reference

Returns

true if the widget accepted the focus.

31.146.4.95 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

31.146.4.96 testShortcut() [1/2]

`int Fl_Widget::testShortcut ( )` [ ]

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.

31.146.4.97 testShortcut() [2/2]

`int Fl_Widget::testShortcut ( 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.

---

#### 31.146.4.98 tooltip() [1/2]

```cpp
class Fl_Widget {
public:
    const char* 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*)`

---

#### 31.146.4.99 tooltip() [2/2]

```cpp
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()

31.146.4.100 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()

31.146.4.101 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)

31.146.4.102 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"?

31.146.4.103  type() [2/2]

void Fl_Widget::type (  
   uchar t ) [inline]

Sets the widget type.

This is used for Forms compatibility.

31.146.4.104  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

31.146.4.105  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

| in | v | new user data |

31.146.4.106  visible()

unsigned int Fl_Widget::visible ( ) const [inline]

Returns whether a widget is visible.
Return values

- **0** if the widget is not drawn and hence invisible.

See also

- `show()`, `hide()`, `visible_r()`

### 31.146.4.107 `visible_focus()` [1/2]

```cpp
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()`

### 31.146.4.108 `visible_focus()` [2/2]

```cpp
unsigned int Fl_Widget::visible_focus ( ) [inline]
```

Checks whether this widget has a visible focus.

Return values

- **0** if this widget has no visible focus.

See also

- `visible_focus(int)`, `set_visible_focus()`, `clear_visible_focus()`

### 31.146.4.109 `visible_r()`

```cpp
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()

31.146.4.110 w() [1/2]

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.

31.146.4.111 w() [2/2]

int Fl_Widget::w ( ) const [inline]

Gets the widget width.

Returns

the width of the widget in pixels.

31.146.4.112 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)
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 |

31.146.4.114  window()

Fl_Window * Fl_Widget::window () const

Returns a pointer to the nearest parent window up the widget hierarchy.

This will return sub-windows if there are any, or the parent window if there's no sub-windows. If this widget IS the top-level window, NULL is returned.

Return values

| NULL | if no window is associated with this widget. |
Note

for an Fl_Window widget, this returns its parent window (if any), not this window.

See also

top_window()

31.146.4.115 \texttt{x()} [1/2]

\begin{verbatim}
void Fl_Widget::x ( int v ) [inline], [protected]
\end{verbatim}

Internal use only.

Use \texttt{position(int,int)}, \texttt{size(int,int)} or \texttt{resize(int,int,int,int)} instead.

31.146.4.116 \texttt{x()} [2/2]

\begin{verbatim}
int Fl_Widget::x ( ) const [inline]
\end{verbatim}

Gets the widget position in its window.

Returns

the x position relative to the window

31.146.4.117 \texttt{y()} [1/2]

\begin{verbatim}
void Fl_Widget::y ( int v ) [inline], [protected]
\end{verbatim}

Internal use only.

Use \texttt{position(int,int)}, \texttt{size(int,int)} or \texttt{resize(int,int,int,int)} instead.

31.146.4.118 \texttt{y()} [2/2]

\begin{verbatim}
int Fl_Widget::y ( ) const [inline]
\end{verbatim}

Gets the widget position in its window.

Returns

the y position relative to the window

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
A surface on which any FLTK widget can be drawn.

#include <Fl_Widget_Surface.H>

Inheritance diagram for Fl_Widget_Surface:

- Fl_Surface_Device
- Fl_Widget_Surface
- Fl_Copy_Surface
- Fl_EPS_File_Surface
- Fl_Image_Surface
- Fl_Paged_Device
- Fl_SVG_File_Surface
- Fl_PostScript_File_Device
- Fl_Printer

### Public Member Functions

- **void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)**
  
  Draws the widget on the drawing surface.

- **virtual void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)**
  
  Draws a window with its title bar and frame if any.

- **virtual void origin (int x, int y)**
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- **virtual void origin (int *x, int *y)**
  
  Computes the coordinates of the current origin of graphics functions.

- **void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  
  Draws a rectangular part of an on-screen window.

- **virtual int printable_rect (int *w, int *h)**
  
  Computes the width and height of the drawable area of the drawing surface.

- **virtual void translate (int x, int y)**
  
  Translates the current graphics origin accounting for the current rotation.

- **virtual void untranslate ()**
  
  Undoes the effect of a previous translate() call.

### Protected Member Functions

- **Fl_Widget_Surface (Fl_Graphics_Driver *d)**
  
  The constructor.

### Protected Attributes

- **int x_offset**
  
  horizontal offset to the origin of graphics coordinates

- **int y_offset**
  
  vertical offset to the origin of graphics coordinates
Additional Inherited Members

31.147.1 Detailed Description

A surface on which any FLTK widget can be drawn.

31.147.2 Constructor & Destructor Documentation

31.147.2.1 Fl_Widget_Surface()

Fl_Widget_Surface::Fl_Widget_Surface (
        Fl_Graphics_Driver * d ) [protected]

The constructor.

Parameters

\( d \) can be null.

31.147.3 Member Function Documentation

31.147.3.1 draw()

void Fl_Widget_Surface::draw ( 
        Fl_Widget * widget,
        int delta_x = 0,
        int delta_y = 0 )

Draws the widget on the drawing surface.

The widget's position on the surface is determined by the last call to `origin()` and by the optional `delta_x` and `delta_y` arguments. Its dimensions are in points unless there was a previous call to `scale()`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in widget</code></td>
<td>Fl_Widget</td>
<td>Any FLTK widget (e.g., standard, custom, window).</td>
</tr>
<tr>
<td><code>in delta_x</code></td>
<td>int</td>
<td>Optional horizontal and vertical offsets for positioning the widget top left relatively to the current origin of graphics.</td>
</tr>
</tbody>
</table>
31.147.3.2 draw_decorated_window()

```cpp
void Fl_Widget_Surface::draw_decorated_window (  
    Fl_Window * win,  
    int win_offset_x = 0,  
    int win_offset_y = 0 ) [virtual]
```

Draws a window with its title bar and frame if any.

`win_offset_x` and `win_offset_y` are optional coordinates of where to position the window top left. Equivalent to `draw()` if `win` is a subwindow or has no border. Use `Fl_Window::decorated_w()` and `Fl_Window::decorated_h()` to get the size of the framed window.

Reimplemented in `Fl_Printer`, and `Fl_Copy_Surface`.

31.147.3.3 origin() [1/2]

```cpp
void Fl_Widget_Surface::origin (  
    int x,  
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.

Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don’t combine their effects. Origin() calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x, y</code></td>
<td>Horizontal and vertical positions in the drawing surface of the desired origin of graphics.</td>
</tr>
</tbody>
</table>


31.147.3.4 origin() [2/2]

```cpp
void Fl_Widget_Surface::origin (  
    int * x,  
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x, y</code></td>
<td>If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.</td>
</tr>
</tbody>
</table>

Generated by Doxygen

31.147.3.5  print_window_part()

```c++
void Fl_Widget_Surface::print_window_part (
    Fl_Window * win,
    int x,
    int y,
    int w,
    int h,
    int delta_x = 0,
    int delta_y = 0 )
```

Draws 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. Can be an Fl_Gl_Window. Sub-windows that intersect the rectangle are also captured.</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>, <code>delta_y</code></td>
<td>Optional horizontal and vertical offsets from current graphics origin where to draw the top left of the captured rectangle.</td>
</tr>
</tbody>
</table>

31.147.3.6  printable_rect()

```c++
int Fl_Widget_Surface::printable_rect (  
    int * w,
    int * h )  [virtual]
```

Computes the width and height of the drawable area of the drawing surface.

Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**

0 if OK, non-zero if any error

31.147.3.7 translate()

```cpp
void Fl_Widget_Surface::translate (  
    int x,  
    int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation.

Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.


The documentation for this class was generated from the following files:

- `Fl_Widget_Surface.H`
- `Fl_Widget_Surface.cxx`

31.148 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.

```cpp
#include <Fl.H>
```

Public Member Functions

- `int deleted ()`
  
  Returns 1, if the watched widget has been deleted.

- `int exists ()`
  
  Returns 1, if the watched widget exists (has not been deleted).

- `Fl_Widget_Tracker (Fl_Widget *wi)`
  
  The constructor adds a widget to the watch list.

- `Fl_Widget * widget ()`
  
  Returns a pointer to the watched widget.

- `~Fl_Widget_Tracker ()`
  
  The destructor removes a widget from the watch list.
31.148.1 Detailed Description

This class should be used to control safe widget deletion.

You can use an `Fl_Widget_Tracker` object to watch another widget, if you need to know whether this widget has been deleted during a callback.

This simplifies the use of the "safe widget deletion" methods `Fl::watch_widget_pointer()` and `Fl::release_widget_pointer()` and makes their use more reliable, because the destructor automatically releases the widget pointer from the widget watch list.

`Fl_Widget_Tracker` is intended to be used as an automatic (local/stack) variable, such that its destructor is called when the object's scope is left. This ensures that no stale widget pointers are left in the widget watch list (see example below).

You can also create `Fl_Widget_Tracker` objects with `new`, but then it is your responsibility to delete the object (and thus remove the widget pointer from the watch list) when it is no longer needed.

Example:

```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,
        // and it is safe to access the widget:
        box(FL_FLAT_BOX);
        color(FL_WHITE);
        redraw();
    }
}
```

31.148.2 Member Function Documentation

31.148.2.1 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.
31.148.2.2 exists()

int Fl_Widget_Tracker::exists ( ) [inline]

Returns 1, if the watched widget exists (has not been deleted).

This is a convenience method. You can also use something like

if (wp.widget() != 0) // ...

where \( wp \) is an Fl_Widget_Tracker object.

31.148.2.3 widget()

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

31.149 Fl_Window Class Reference

This widget produces an actual window.

#include <Fl_Window.H>

Inheritance diagram for Fl_Window:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Window
  ↓
Fl_Double_Window
  ↓
Fl_Cairo_Window
  ↓
Fl_Gl_Window
  ↓
Fl_Overlay_Window
  ↓
Fl_Glut_Window
  ↓
Fl_Menu_Window
```

Generated by Doxygen
Public Member Functions

- virtual class Fl_Double_Window * as_double_window ()
  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- virtual Fl_Window * as_window ()
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- void border (int b)
  Sets whether or not the window manager border is around the window.
- unsigned int border () const
  Returns whether the window possesses a border.
- 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_Cursor)
  Changes the cursor for this window.
- 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.
- int decorated_h () const
  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const
  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor)
  Sets the default window cursor.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
- Fl_Window (int w, int h, const char * title=0)
  Creates a window from the given width w, height h, and title.
- Fl_Window (int x, int y, int w, int h, const char * title=0)
  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()
  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int fullscreen_active () const
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void fullscreen_off ()
  Turns off any side effects of fullscreen()
- void fullscreen_off (int X, int Y, int W, int H)
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).
- void fullscreen_screens (int top, int bottom, int left, int right)
  Sets which screens should be used when this window is in fullscreen mode.
- virtual int handle (int)
  Handles the specified event.
virtual void hide ()

Removes the window from the screen.

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.

void hotspot (const Fl_Widget &p, int offscreen=0)

See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

void icon (const Fl_RGB_Image *)

Sets or resets a single window icon.

void icon (const Fl_Widget &p, int offscreen=0)

See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

void icon (const void *)

Sets the current icon window target dependent data.

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.

int screen_num ()

The number of the screen containing the mapped 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 *img)`
  Assigns a non-rectangular shape to the window.

• void `shape(const Fl_Image &b)`
  Set the window’s shape with an `Fl_Image`.

• `const Fl_Image * shape()`
  Returns the image controlling the window shape or NULL.

• 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_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.

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 void `default_xclass(const char *)`
  Sets the default window xclass.

• static const char * `default_xclass()`
  Returns the default xclass.

• static bool `is_a_rescale()`
  Returns true when a window is being rescaled.
Protected Member Functions

- void draw ()
  
  Draws the widget.
- virtual void flush ()
  
  Forces the window to be drawn, this window is also made current and calls draw().
- void force_position (int force)
  
  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- int force_position () const
  
  Returns the internal state of the window's FORCE_POSITION flag.
- void free_icons ()
  
  Deletes all icons previously attached to the window.

Static Protected Attributes

- static Fl_Window * current_
  
  Stores the last window that was made current.

Friends

- int Fl::arg (int argc, char **argv, int &i)
- class Fl_Window_Driver
- class Fl_X

Additional Inherited Members

31.149.1 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().

31.149.2 Constructor & Destructor Documentation

Generated by Doxygen
31.149.2.1 Fl_Window() [1/2]

Fl_Window::Fl_Window (  
    int w,  
    int h,  
    const char * title = 0 )

Creates a window from the given width w, height h, and title.

If Fl_Group::current() is not NULL, the window is created as a subwindow of the parent window.

The (w, h) form of the constructor creates a top-level window and asks the window manager to position the window. The (x, y, w, h) form of the constructor either creates a subwindow or a top-level window at the specified location (x, y), subject to window manager configuration. If you do not specify the position of the window, the window manager will pick a place to show the window or allow the user to pick a location. Use position(x, y) or hotspot() before calling show() to request a position on the screen. See Fl_Window::resize() for some more details on positioning windows.

Top-level windows initially have visible() set to 0 and parent() set to NULL. Subwindows initially have visible() set to 1 and parent() set to the parent window pointer.

Fl_Widget::box() defaults to FL_FLAT_BOX. If you plan to completely fill the window with children widgets you should change this to FL_NO_BOX. If you turn the window border off you may want to change this to FL_UP_BOX.

See also

    Fl_Window(int x, int y, int w, int h, const char *title)

31.149.2.2 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 (x, y), size (w, h) and title.

See also

    Fl_Window(int w, int h, const char *title)

31.149.2.3 ~Fl_Window()

Fl_Window::~Fl_Window ( ) [virtual]

The destructor also deletes all the children.

This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Window and all of its children can be automatic (local) variables, but you must declare the Fl_Window first so that it is destroyed last.
31.149.3 Member Function Documentation

31.149.3.1 as_window()

virtual Fl_Window* Fl_Window::as_window ( ) [inline], [virtual]

Returns an Fl_Window pointer if this widget is an Fl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer to access its children or other Fl_Window-specific methods.

Return values

NULL if this widget is not derived from Fl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented from Fl_Widget.

31.149.3.2 border()

void Fl_Window::border ( int b )

Sets whether or not the window manager border is around the window.

The default value is true. With some X window managers, this does not work after show() has been called.

31.149.3.3 clear_border()

void Fl_Window::clear_border ( ) [inline]

Fast inline function to turn the window manager border off.

It only works before show() is called.
31.149.3.4 clear_modal_states()

```cpp
class Fl_Window:
public:
    void clear_modal_states() [inline]
```

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

Note that there are three states for a window: modal, non-modal, and normal.

You can not change the "modality" of a window whilst it is shown, so it is necessary to first hide() the window, change its "modality" as required, then re-show the window for the new state to take effect.

This method can also be used to change a "modal" window into a "non-modal" one. On several supported platforms, the "modal" state over-rides the "non-modal" state, so the "modal" state must be cleared before the window can be set into the "non-modal" state. In general, the following sequence should work:

```cpp
win->hide();
win->clear_modal_states();
// Set win to new state as desired, or leave "normal", e.g...
win->set_non_modal();
win->show();
```

**Note**

Under some window managers, the sequence of hiding the window and changing its modality will often cause it to be re-displayed at a different position when it is subsequently shown. This is an irritating feature but appears to be unavoidable at present. As a result we would advise to use this method only when absolutely necessary.

See also

- void set_modal(), void set_non_modal()

31.149.3.5 current()

```cpp
class Fl_Window:
public:
    Fl_Window * current() [static]
```

Returns the last window that was made current.

See also

- Fl_Window::make_current()
31.149.3.6 cursor() [1/3]

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()

31.149.3.7 cursor() [2/3]

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()

31.149.3.8 cursor() [3/3]

void Fl_Window::cursor ( 
    Fl_Cursor c, 
    Fl_Color , 
    Fl_Color  = FL_WHITE )

For back compatibility only.

Same as Fl_Window::cursor(Fl_Cursor)
31.149.3.9 decorated_h()

```cpp
int Fl_Window::decorated_h ( ) const
```

Returns the window height including any window title bar and any frame added by the window manager.

Same as `h()` if applied to a subwindow.

31.149.3.10 decorated_w()

```cpp
int Fl_Window::decorated_w ( ) const
```

Returns the window width including any frame added by the window manager.

Same as `w()` if applied to a subwindow.

31.149.3.11 default_cursor()[1/2]

```cpp
void Fl_Window::default_cursor ( Fl_Cursor c )
```

Sets the default window cursor.

This is the cursor that will be used after the mouse pointer leaves a widget with a custom cursor set.

See also

```
cursor(const Fl_RGB_Image*, int, int), default_cursor()
```

31.149.3.12 default_cursor()[2/2]

```cpp
void Fl_Window::default_cursor ( Fl_Cursor c,
                                Fl_Color ,
                                Fl_Color = FL_WHITE )
```

For back compatibility only.

same as `Fl_Window::default_cursor(Fl_Cursor)`

31.149.3.13 default_icon()

```cpp
void Fl_Window::default_icon ( const Fl_RGB_Image * icon ) [static]
```

Sets a single default window icon.

If `icon` is NULL the current default icons are removed.
Parameters

| in  | icon | default icon for all windows subsequently created or NULL |

See also

Fl_Window::default_icons(const Fl_RGB_Image *[], int)
Fl_Window::icon(const Fl_RGB_Image *)
Fl_Window::icons(const Fl_RGB_Image *[], int)

31.149.3.14 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

| in  | icons | default icons for all windows subsequently created |
| in  | count | number of images in icons. Set to 0 to remove the current default icons |

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)

31.149.3.15 default_xclass() [1/2]

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.

Generated by Doxygen
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 *)`

31.149.3.16  default_xclass() [2/2]

```cpp
const char * Fl_Window::default_xclass ( ) [static]
```

Returns the default xclass.

See also

- `Fl_Window::default_xclass(const char *)`

31.149.3.17  draw()

```cpp
void Fl_Window::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

Reimplemented in `Fl_Glut_Window`. 

Generated by Doxygen
### flush()

```cpp
void Fl_Window::flush ( ) [protected], [virtual]
```

Forces the window to be drawn, this window is also made current and calls `draw()`.

Reimplemented in `Fi_Gl_Window`, `Fi_Overlay_Window`, `Fi_Double_Window`, and `Fi_Menu_Window`.

### force_position() [1/2]

```cpp
void Fl_Window::force_position ( int force ) [inline], [protected]
```

Sets an internal flag that tells FLTK and the window manager to honor position requests. This is used internally and should not be needed by user code.

**Parameters**

| in  | force | 1 to set the FORCE_POSITION flag, 0 to clear it |

### force_position() [2/2]

```cpp
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)`

### free_icons()

```cpp
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)`
31.149.3.22 free_position()

```cpp
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

31.149.3.23 fullscreen()

```cpp
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**

```cpp
void Fl_Window::fullscreen_screens()
```

31.149.3.24 fullscreen_screens()

```cpp
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**

```cpp
void Fl_Window::fullscreen()
```
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

FI_Event

Reimplemented from Fl_Group.

Reimplemented in Fl_Gl_Window, and Fl_Glut_Window.

31.149.3.26 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_Gl_Window, Fl_Overlay_Window, Fl_Double_Window, and Fl_Menu_Window.

31.149.3.27 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()
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)`

###(icon) [2/3]

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

Gets the current icon window target dependent data.

**Deprecated** in 1.3.3

###(icon) [3/3]

```cpp
void Fl_Window::icon ( const void * ic )
```

Sets the current icon window target dependent data.

**Deprecated** in 1.3.3

###(iconize)

```cpp
void Fl_Window::iconize ( )
```

Iconifies the window.

If you call this when `shown()` is false it will `show()` it as an icon. If the window is already iconified this does nothing.

Call `show()` to restore the window.

When a window is iconified/restored (either by these calls or by the user) the `handle()` method is called with `FL_HIDE` and `FL_SHOW` events and `visible()` is turned on and off.

There is no way to control what is drawn in the icon except with the string passed to `Fl_Window::xclass()`. You should not rely on window managers displaying the icons.
31.149.3.32  iconlabel()

void Fl_Window::iconlabel (  
    const char * iname )

Sets the icon label.

31.149.3.33  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 *)

31.149.3.34  label()[1/2]

void Fl_Window::label (  
    const char * name )

Sets the window title bar label.

31.149.3.35  label()[2/2]

void Fl_Window::label (  
    const char * label,  
    const char * iconlabel )

Sets the icon label.
31.149.3.36  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.

31.149.3.37  menu_window()

unsigned int Fl_Window::menu_window ( ) const [inline]

Returns true if this window is a menu window.

31.149.3.38  modal()

unsigned int Fl_Window::modal ( ) const [inline]

Returns true if this window is modal.

31.149.3.39  non_modal()

unsigned int Fl_Window::non_modal ( ) const [inline]

Returns true if this window is modal or non-modal.

31.149.3.40  override()

unsigned int Fl_Window::override ( ) const [inline]

Returns non zero if FL_OVERRIDE flag is set, 0 otherwise.
31.149 Fl_Window Class Reference

31.149.3.41 resize()

```cpp
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_Gl_Window, Fl_Overlay_Window, and Fl_Double_Window.

31.149.3.42 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.

31.149.3.43 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().
31.149.3.44 set_non_modal()

void Fl_Window::set_non_modal ( ) [inline]

A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

There are three states for a window: modal, non-modal, and normal.

31.149.3.45 set_tooltip_window()

void Fl_Window::set_tooltip_window ( ) [inline]

Marks the window as a tooltip window.

This is intended for internal use, but it can also be used if you write your own tooltip handling. However, this is not recommended.

This flag is used for correct "parenting" of windows in communication with the windowing system. Modern X window managers can use different flags to distinguish menu and tooltip windows from normal windows.

This must be called before the window is shown and cannot be changed later.

Note

Since Fl_Tooltip_Window is derived from Fl_Menu_Window, this also clears the menu_window() state.

31.149.3.46 shape() [1/2]

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_Image::scale().

Platform details:

• On the unix/linux platform, the SHAPE extension of the X server is required. This function does control the shape of Fl_Gl_Window instances.

• On the Windows platform, this function does nothing with class Fl_Gl_Window.

• On the Mac platform, OS version 10.4 or above is required. An 8-bit shape-mask is used when img is an Fl_RGB_Image: with depths 2 or 4, the image alpha channel becomes the shape mask such that areas with alpha = 0 are out of the shaped window; with depths 1 or 3, white and black are in and out of the shaped window, respectively, and other colors give intermediate masking scores. This function does nothing with class Fl_Gl_Window.

The window borders and caption created by the window system are turned off by default. They can be re-enabled by calling Fl_Window::border(1).

A usage example is found at example/shapedwindow.cxx.

Version

1.3.3

31.149.3.47  shape() [2/2]

void Fl_Window::shape (  
    const Fl_Image & img   )

Set the window’s shape with an Fl_Image.

See also

void shape(const Fl_Image* img)

31.149.3.48  show() [1/2]

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(0) in other toolkits.

Fl_Window::show(int argc, char **argv) is used for top-level windows and allows standard arguments to be parsed from the command-line.
Note

For some obscure reasons `Fl_Window::show()` resets the current group by calling `Fl_Group::current(0)`. The comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have unwanted side effects if you `show()` an unrelated window (maybe for an error message or warning) while building a window or any other group widget.

Todo  Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

See also

`Fl_Window::show(int argc, char **argv)`

Reimplemented from `Fl_Widget`.


31.149.3.49 show() [2/2]

```cpp
void Fl_Window::show ( 
  int argc,
  char ** argv )
```

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 `argc` and `argv` from `main(int argc, char **argv)` for this call.

The first call also sets up some system-specific internal variables like the system colors.

Todo  explain which system parameters are set up.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>argc</code></td>
<td>command-line argument count, usually from <code>main()</code></td>
</tr>
<tr>
<td><code>argv</code></td>
<td>command-line argument vector, usually from <code>main()</code></td>
</tr>
</tbody>
</table>

See also

`virtual void Fl_Window::show()`
shown()

int Fl_Window::shown() [inline]

Returns non-zero if show() has been called (but not hide()).

You can tell if a window is iconified with (w->shown() && !w->visible()).

size_range()

void Fl_Window::size_range(
    int minWidth,
    int minHeight,
    int maxWidth = 0,
    int maxHeight = 0,
    int deltaX = 0,
    int deltaY = 0,
    int aspectRatio = 0)

Sets the allowable range the user can resize this window to.

This only works for top-level windows.

If this function is not called, FLTK tries to figure out the range from the setting of resizable():

- If resizable() is NULL (this is the default) then the window cannot be resized and the resize border and max-size control will not be displayed for the window.
- If either dimension of resizable() is less than 100, then that is considered the minimum size. Otherwise the resizable() has a minimum size of 100.
- If either dimension of 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 size_range().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in minLength, minHeight</td>
<td>The smallest the window can be. Either value must be greater than 0.</td>
</tr>
<tr>
<td>in maxWidth, maxHeight</td>
<td>The largest the window can be. If either is equal to the minimum then you cannot resize in that direction. If either is zero then FLTK picks a maximum size in that direction such that the window will fill the screen.</td>
</tr>
<tr>
<td>in deltaX, deltaY</td>
<td>These are size increments. The window will be constrained to widths of minLength + N * deltaX, where N is any non-negative integer. If these are less or equal to 1 they are ignored. (this is ignored on Windows)</td>
</tr>
<tr>
<td>in aspectRatio</td>
<td>A flag that indicates that the window should preserve its aspect ratio. This only works if both the maximum and minimum have the same aspect ratio (ignored on Windows and by many X window managers).</td>
</tr>
</tbody>
</table>
### tooltip_window()

```cpp
unsigned int Fl_Window::tooltip_window ( ) const [inline]
```

Returns true if this window is a tooltip window.

### wait_for_expose()

```cpp
void Fl_Window::wait_for_expose ( )
```

Waits for the window to be displayed after calling `show()`.

**FL_Window::show()** is not guaranteed to show and draw the window on all platforms immediately. Instead this is done in the background; particularly on X11 it will take a few messages (client server roundtrips) to display the window. Usually this small delay doesn't matter, but in some cases you may want to have the window instantiated and displayed synchronously.

Currently (as of FLTK 1.3.4) this method has an effect on X11 and Mac OS. On Windows, **show()** is always synchronous. The effect of **show()** varies with versions of Mac OS X: early versions have the window appear on the screen when **show()** returns, later versions don't. If you want to write portable code and need this synchronous **show()** feature, add **win->wait_for_expose()** on all platforms, and FLTK will just do the right thing.

This method can be used for displaying splash screens before calling **Fl::run()** or for having exact control over which window has the focus after calling **show()**.

If the window is not **shown()**, this method does nothing.

**Note**

Depending on the platform and window manager **wait_for_expose()** may not guarantee that the window is fully drawn when it is called. Under X11 it may only make sure that the window is **mapped**, i.e. the internal (OS dependent) window object was created (and maybe shown on the desktop as an empty frame or something like that). You may need to call **Fl::flush()** after **wait_for_expose()** to make sure the window and all its widgets are drawn and thus visible.

FLTK does the best it can do to make sure that all widgets get drawn if you call **wait_for_expose()** and **Fl::flush()**. However, dependent on the window manager it can not be guaranteed that this does always happen synchronously. The only guaranteed behavior that all widgets are eventually drawn is if the FLTK event loop is run continuously, for instance with **Fl::run()**.

**See also**

virtual void Fl_Window::show()

**Example code for displaying a window before calling 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 **Fl::run()**.
31.149.3.54 xclass() [1/2]

const char * Fl_Window::xclass ( ) const

Returns the xclass for this window, or a default.

See also

Fl_Window::default_xclass(const char *)
Fl_Window::xclass(const char *)

31.149.3.55 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 *)
31.149.4.1 current_

Fl_Window * Fl_Window::current_ [static], [protected]

Stores the last window that was made current.

See current() const

The documentation for this class was generated from the following files:

- Fl_Window.H
- Fl_arg.cxx
- Fl_cursor.cxx
- Fl_Window.cxx
- Fl_Window_fullscreen.cxx
- Fl_Window_hotspot.cxx
- Fl_Window_iconize.cxx

31.150 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
  ↓
Fl_Group
  ↓
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.
Protected Member Functions

- virtual void draw ()
  
  Draws the wizard border and visible child.

Additional Inherited Members

31.150.1 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.

31.150.2 Constructor & Destructor Documentation

31.150.2.1 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.

31.150.3 Member Function Documentation

31.150.3.1 draw()

void Fl_Wizard::draw ( 
    void ) [protected], [virtual]

Draws the wizard border and visible child.

Reimplemented from Fl_Group.
31.150.3.2 next()

```cpp
void Fl_Wizard::next ( )
```

This method shows the next child of the wizard.

If the last child is already visible, this function does nothing.

31.150.3.3 prev()

```cpp
void Fl_Wizard::prev ( )
```

Shows the previous child.

31.150.3.4 value() [1/2]

```cpp
Fl_Widget * Fl_Wizard::value ( )
```

Gets the current visible child widget.

31.150.3.5 value() [2/2]

```cpp
void Fl_Wizard::value ( Fl_Widget * kid )
```

Sets the child widget that is visible.

The documentation for this class was generated from the following files:

- Fl_Wizard.H
- Fl_Wizard.cxx

31.151 Fl_XBM_Image Class Reference

The **Fl_XBM_Image** class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

```cpp
#include <Fl_XBM_Image.H>
```

Inheritance diagram for **Fl_XBM_Image**:

```plaintext
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.*

Additional Inherited Members

31.151.1 Detailed Description

The **Fl_XBM_Image** class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

31.151.2 Constructor & Destructor Documentation

31.151.2.1 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**

31.152 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**

---

Generated by Doxygen
31.153  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.

Additional Inherited Members

31.153.1  Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

31.153.2  Constructor & Destructor Documentation

31.153.2.1  Fl_XPM_Image()

```
Fl_XPM_Image::Fl_XPM_Image (const char * name )
```

The constructor loads the XPM image from the name filename.

The destructor frees all memory and server resources that are used by the image.

The documentation for this class was generated from the following files:

- Fl_XPM_Image.H
- Fl_XPM_Image.cxx
31.154 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

31.154.1 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

31.155 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 (unsigned int n)**
  - Creates a group name or entry name on the fly.
- **Name (const char *format,...)**
  - Creates a group name or entry name on the fly.
- **operator const char * ()**
  - Return the Name as a "C" string.
31.155.1 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.

31.155.2 Constructor & Destructor Documentation

31.155.2.1 Name() [1/2]

Fl_Preferences::Name::Name ( 
    unsigned int n )

Creates a group name or entry name on the fly.

This version creates a simple unsigned integer as an entry name.

```c
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], "" );
```

31.155.2.2 Name() [2/2]

Fl_Preferences::Name::Name ( 
    const char * format, 
    ... )

Creates a group name or entry name on the fly.

This version creates entry names as in ‘printf’.

```c
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
Public Member Functions

- void add (const char *line)
- Node *addChild (const char *path)
- const char *child (int ix)
- Node *childNode (int ix)
- void clearDirtyFlags ()
- void deleteAllChildren ()
- void deleteAllEntries ()
- char deleteEntry (const char *name)
- char dirty ()
- Entry & entry (int i)
- Node *find (const char *path)
- RootNode *findRoot ()
- const char *get (const char *name)
- int getEntry (const char *name)
- const char *name ()
- int nChildren ()
- int nEntry ()
- Node (const char *path)
- Node *parent ()
- const char *path ()
- char remove ()
- Node *search (const char *path, int offset=0)
- void set (const char *name, const char *value)
- void set (const char *line)
- void setParent (Node *parent)
- void setRoot (RootNode *r)
- int write (FILE *f)

Static Public Attributes

- static int lastEntrySet = -1

The documentation for this class was generated from the following files:

- FL_Preferences.H
- FL_Preferences.cxx

31.157 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

31.157.1 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

31.158 Fl_Preferences::RootNode Class Reference

Public Member Functions

- char getPath (char *path, int pathlen)
- int read ()
- 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 *)
- int write ()

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

31.159 Fl_Scroll::ScrollInfo Struct Reference

Structure to manage scrollbar and widget interior sizes.

#include <Fl_Scroll.H>

Classes

- struct Fl_Region_LRTB
  A local struct to manage a region defined by left/right/top/bottom.
- struct Fl_Region_XYWH
  A local struct to manage a region defined by xywh.
- struct Fl_Scrollbar_Data
  A local struct to manage a scrollbar's xywh region and tab values.
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

### 31.159.1 Detailed Description

Structure to manage scrollbar and widget interior sizes.

This is filled out by `recalc_scrollbars()` for use in calculations that need to know the visible scroll area size, etc.

#### Version

1.3.3

The documentation for this struct was generated from the following file:

- Fl_Scroll.H

### 31.160 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.

```cpp
#include <Fl_Text_Display.H>
```

Public Attributes

- **unsigned attr**
  
  currently unused (this may be changed in the future)
- **Fl_Color color**
  
  text color
- **Fl_Font font**
  
  text font
- **Fl_Fontsize size**
  
  text font size
31.160.1 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 32

File Documentation

32.1 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"
#include <FL/platform_types.h>
```

**Macros**

- `#define FL_IMAGE_WITH_ALPHA 0x40000000`

**Version Numbers**

*FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.*

The following constants are defined:

- `#define FL_MAJOR_VERSION 1`
  *The major release version of this FLTK library.*
- `#define FL_MINOR_VERSION 4`
  *The minor release version for this library.*
- `#define FL_PATCH_VERSION 0`
  *The patch version for this library.*
- `#define FL_VERSION` (FL_MAJOR_VERSION ∗ 10000 + FL_MINOR_VERSION ∗ 100 + FL_PATCH_VERSION)
  *The FLTK version number as a double.*
- `#define FL_ABI_VERSION FL_API_VERSION`
  *The FLTK ABI (Application Binary Interface) version number as an int.*

**Mouse and Keyboard Events**

*This and the following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events.*
See also

`Fl::event_key()` and `Fl::get_key(int)` (use ASCII letters for all other keys):

**Todo** FL_Button and FL_key... constants could be structured better (use an enum or some doxygen grouping?)

• #define FL_Button 0xfee8
  A mouse button; use FL_Button + n for mouse button n.
• #define FL_BackSpace 0xff08
  The backspace key.
• #define FL_Tab 0xff09
  The tab key.
• #define FL_iso_Key 0xff0c
  The additional key of ISO keyboards.
• #define FL_Enter 0xff0d
  The enter key.
• #define FL_Pause 0xff13
  The pause key.
• #define FL_Scroll_Lock 0xff14
  The scroll lock key.
• #define FL_Escape 0xff1b
  The escape key.
• #define FL_Kana 0xff2e
  The Kana key of JIS keyboards.
• #define FL_Eisu 0xff2f
  The Eisu key of JIS keyboards.
• #define FL_Yen 0xff30
  The Yen key of JIS keyboards.
• #define FL_JIS_Underscore 0xff31
  The underscore key of JIS keyboards.
• #define FL_Home 0xff50
  The home key.
• #define FL_Left 0xff51
  The left arrow key.
• #define FL_Up 0xff52
  The up arrow key.
• #define FL_Right 0xff53
  The right arrow key.
• #define FL_Down 0xff54
  The down arrow key.
• #define FL_Page_Up 0xff55
  The page-up key.
• #define FL_Page_Down 0xff56
  The page-down key.
• #define FL_End 0xff57
  The end key.
• #define FL_Print 0xff61
  The print (or print-screen) key.
• #define FL_Insert 0xff63
  The insert key.
• #define FL_Menu 0xff67
  The menu key.
• #define FL_Help 0xff68
  The 'help' key on Mac 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_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_Shift_L 0xffe1
  The lefthand shift key.

- #define FL_Shift_R 0xffe2
  The righthand shift key.

- #define FL_Control_L 0xffe3
  The lefthand control key.

- #define FL_Control_R 0xffe4
  The righthand control key.

- #define FL_Caps_Lock 0xffe5
  The caps lock key.

- #define FL_Meta_L 0xffe7
  The left meta/Windows key.

- #define FL_Meta_R 0xffe8
  The right meta/Windows key.

- #define FL_Alt_L 0xffe9
  The left alt key.

- #define FL_Alt_R 0xffea
  The right alt key.

- #define FL_Delete 0xffff
  The delete 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_Media_Play 0xEF14 /* Start playing of audio */
  - #define FL_Media_Stop 0xEF15 /* Stop playing audio */
  - #define FL_Media_Prev 0xEF16 /* Previous track */
  - #define FL_Media_Next 0xEF17 /* Next track */
  - #define FL_Home_Page 0xEF18 /* Display user's home page */
  - #define FL_Mail 0xEF19 /* Invoke user's mail program */
  - #define FL_Search 0xEF1B /* Search */
  - #define FL_Back 0xEF26 /* Like back on a browser */
  - #define FL_Forward 0xEF27 /* Like forward on a browser */
  - #define FL_Stop 0xEF28 /* Stop current operation */
  - #define FL_Refresh 0xEF29 /* Refresh the page */
  - #define FL_Sleep 0xEF2F /* Put system to sleep */
  - #define FL_Favorites 0xEF30 /* Show favorite locations */

**Mouse Buttons**

These constants define the button numbers for FL_PUSH and FL_RELEASE events.

See also

"Fl::event_button()"

- #define FL_LEFT_MOUSE 1
  The left mouse button.

- #define FL_MIDDLE_MOUSE 2
  The middle mouse button.

- #define FL_RIGHT_MOUSE 3
  The right mouse button.

**Event States**
The following constants define bits in the Fl::event_state() value.

- `#define FL_SHIFT 0x00010000`  
  One of the shift keys is down.
- `#define FL_CAPS_LOCK 0x00020000`  
  The caps lock is on.
- `#define FL_CTRL 0x00040000`  
  One of the ctrl keys is down.
- `#define FL_ALT 0x00080000`  
  One of the alt keys is down.
- `#define FL_NUM_LOCK 0x00100000`  
  The num lock is on.
- `#define FL_META 0x00400000`  
  One of the meta/Windows keys is down.
- `#define FL_SCROLL_LOCK 0x00800000`  
  The scroll lock is on.
- `#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_KEY_MASK 0x0000ffff`  
  All keys are 16 bit for now.

**Typedefs**

- `typedef int Fl_Fontsize`  
  Size of a font in pixels.

**Enumerations**

- `enum { FL_READ = 1, FL_WRITE = 4, FL_EXCEPT = 8 }`  
  FD "when" conditions.
- `enum Fl_Damage {`  
  FL_DAMAGE_CHILD = 0x01, FL_DAMAGE_EXPOSE = 0x02, FL_DAMAGE_SCROLL = 0x04, FL_DAMAGE_OVERLAY = 0x08, FL_DAMAGE_USER1 = 0x10, FL_DAMAGE_USER2 = 0x20, FL_DAMAGE_ALL = 0x80 }`  
  Damage masks.
- `enum Fl_Event {`  
  FL_NO_EVENT = 0, FL_PUSH = 1, FL_RELEASE = 2, FL_ENTER = 3, FL_LEAVE = 4, FL_DRAG = 5, FL_FOCUS = 6, FL_UNFOCUS = 7, FL_KEYDOWN = 8, FL_KEYBOARD = 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, FL_ZOOM_EVENT = 27 }`  
  Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.
• enum Fl_Labeltype {
    FL_NORMAL_LABEL = 0, FL_NO_LABEL, _FL_SHADOW_LABEL, _FL_ENGRAVED_LABEL,
    _FL_EMBOSSED_LABEL, _FL_MULTI_LABEL, _FL_ICON_LABEL, _FL_IMAGE_LABEL,
    FL_FREE_LABELTYPE }

    The labeltype() method sets the type of the label.

• enum Fl_Mode {
    FL_RGB = 0, FL_INDEX = 1, FL_SINGLE = 0, FL_DOUBLE = 2,
    FL_ACCUM = 4, FL_ALPHA = 8, FL_DEPTH = 16, FL_STENCIL = 32,
    FL_RGB8 = 64, FL_MULTISAMPLE = 128, FL_STEREO = 256, FL_FAKE_SINGLE = 512,
    FL_OPENGL3 = 1024 }

    visual types and Fl_Gl_Window::mode() (values match Glut)

When Conditions

• enum Fl_When {
    FL_WHEN_NEVER = 0, FL_WHEN_CHANGED = 1, FL_WHEN_NOT_CHANGED = 2, FL_WHEN_RELASE = 4,
    FL_WHEN_RELEASE_ALWASY = 6, FL_WHEN_ENTER_KEY = 8, FL_WHEN_ENTER_KEY_ALWASY = 10,
    FL_WHEN_ENTER_KEY_CHANGED = 11 }

    These constants determine when a callback is performed.

Cursors

• enum Fl_Cursor {
    FL_CURSOR_DEFAULT = 0, FL_CURSOR_ARROW = 35, FL_CURSOR_CROSS = 66, FL_CURSOR_WAIT = 76,
    FL_CURSOR_INSERT = 77, FL_CURSOR_HAND = 31, FL_CURSOR_HELP = 47, FL_CURSOR_MOVE = 27,
    FL_CURSOR_NS = 78, FL_CURSOR_WE = 79, FL_CURSOR_NWSE = 80, FL_CURSOR_NESW = 81,
    FL_CURSOR_N = 70, FL_CURSOR_NE = 69, FL_CURSOR_E = 49, FL_CURSOR_SE = 8,
    FL_CURSOR_S = 9, FL_CURSOR_SW = 7, FL_CURSOR_W = 36, FL_CURSOR_NW = 68,
    FL_CURSOR_NONE = 255 }

    The following constants define the mouse cursors that are available in FLTK.

Variables

• FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE

    normal font size

Box Types

FLTK standard box types

This enum defines the standard box types included with FLTK.
The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. _FL_SHADOW_BOX. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

FL_NO_BOX means nothing is drawn at all, so whatever is already on the screen remains. The FL_..._FRAME types only draw their edges, leaving the interior unchanged. The blue color in Figure 1 is the area that is not drawn by the frame types.

![FLTK standard box types](image)

Figure 32.1 FLTK standard box types

Todo Description of boxtypes is incomplete. See below for the defined enum Fl_Boxtype.

See also

src/Fl_get_system_colors.cxx

- #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)
- #define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()
- #define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)
- #define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()
- #define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()
- #define FL_RSHADOW_BOX fl_define_FL_RSHADOW_BOX()
- #define FL_DIAMOND_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)
- #define FL_RDIAMOND_BOX (Fl_Boxtype)(fl_define_FL_RDIAMOND_BOX()+1)
- #define FL_OVAL_BOX fl_define_FL_OVAL_BOX()
- #define FL_OSHADOW_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_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_UP_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+2)
• #define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+3)
• #define FL_PLASTIC_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+4)
• #define FL_PLASTIC_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+5)
• #define FL_PLASTIC_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
• #define FL_PLASTIC_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)
• #define FL_GTK_UP_BOX (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_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+2)
• #define FL_GTK_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+3)
• #define FL_GTK_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+4)
• #define FL_GTK_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+5)
• #define FL_GTK_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+6)
• #define FL_GTK_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+7)
• #define FL_GTK_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+8)
• #define FL_GTK_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)
• #define FL_GLEAM_UP_BOX (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_UP_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
• #define FL_GLEAM_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
• #define FL_GLEAM_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+4)
• #define FL_GLEAM_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
• #define FL_GLEAM_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+6)
• #define FL_GLEAM_THIN_DOWN_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)
• #define FL_FRAME Boxtype 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

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_ENGRAVED_BOX, FL_ENGRAVED_FRAME, FL_THIN_DOWN_FRAME,
    FL_BORDER_BOX, FL_SHADOW_BOX, FL_BORDER_FRAME, FL_SHADOW_FRAME,
    FL_RSHADOW_BOX, _FL_ROUNDED_BOX, _FL_ROUNDED_FRAME,
    FL_ROUND_UP_BOX, FL_ROUND_DOWN_BOX, FL_OVAL_BOX, FL_OSHADOW_BOX,
    FL_OVAL_FRAME, FL_OFLAT_BOX, _FL_PLASTIC_UP_BOX, _FL_RFLAT_BOX,
    _FL_RSHADOW_BOX, _FL_DIAMOND_UP_BOX, _FL_DIAMOND_DOWN_BOX,
    _FL_GTK_UP_BOX, _FL_GTK_DOWN_BOX, _FL_GTK_UP_FRAME, _FL_GTK_DOWN_FRAME,
    _FL_GTK_THIN_UP_BOX, _FL_GTK_THIN_DOWN_BOX, _FL_GTK_ROUND_UP_BOX, _FL_GTK_ROUND_DOWN_BOX,
    _FL_GLEAM_UP_BOX, _FL_GLEAM_DOWN_BOX, _FL_GLEAM_UP_FRAME, _FL_GLEAM_DOWN_FRAME,
    _FL_GLEAM_THIN_UP_BOX, _FL_GLEAM_THIN_DOWN_BOX, _FL_GLEAM_ROUND_UP_BOX, _FL_GLEAM_ROUND_DOWN_BOX,
    _FL_FREE_BOXTYPE
}

FLTK standard box types.
• FL_EXPORT Fl_Boxtype fl_define_FL_OVAL_BOX()
• FL_EXPORT Fl_Boxtype fl_define_FL_PLASTIC_UP_BOX()
• FL_EXPORT Fl_Boxtype fl_define_FL_GTK_UP_BOX()
• FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX()

Get the filled version of a frame.

• Fl_Boxtype fl_box (Fl_Boxtype b)

Get the "pressed" or "down" version of a box.

• Fl_Boxtype fl_down (Fl_Boxtype b)

Get the unfilled, frame only version of a box.

• Fl_Boxtype fl_frame (Fl_Boxtype b)

• #define FL_SYMBOL_LABEL FL_NORMAL_LABEL

Sets the current label type and returns its corresponding Fl_Labeltype value.

• #define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()

Draws a label with shadows behind the text.

• #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()

Draws a label with engraved text.

• #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()

Draws a label with embossed text.

• #define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()

Draws a label that can comprise several parts like text and images.

• #define FL_ICON_LABEL fl_define_FL_ICON_LABEL()

Draws an icon as the label.

• #define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

Draws an image (Fl_Image) as the label.

• Fl_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL()

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.

• Fl_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL()

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.

• Fl_Labeltype FL_EXPORT fl_define_FL_EMBOSSED_LABEL()

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.

• Fl_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL()

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.

• Fl_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL()

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.

• Fl_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL()

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.

Alignment Flags

Flags to control the label alignment.

This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER (0) for most widgets, which centers the label inside the widget.

All alignment flags use the common prefix “FL_ALIGN_”. In the following descriptions this prefix is sometimes omitted for brevity.

Flags can be or’d to achieve a combination of alignments, but there are some "magic values" (e.g. combinations of TOP and BOTTOM and of LEFT and RIGHT) that have special meanings (see below). For instance: FL_ALIGN_TOP_LEFT == (FL_ALIGN_TOP | FL_ALIGN_LEFT) != FL_ALIGN_LEFT_TOP.
Outside alignments (FL_ALIGN_INSIDE is not set):

```
+---------------------------------+
| LEFT_TOP | TOP | TOP_RIGHT |
| LEFT | CENTER | TOP | TOP_RIGHT |
| LEFT_BOTTOM | TOP | TOP_RIGHT |
+---------------------------------+
```

Inside alignments (FL_ALIGN_INSIDE is set):

```
+---------------------------------+
| TOP_LEFT | TOP | TOP_RIGHT |
| LEFT | CENTER | RIGHT |
| BOTTOM_LEFT | BOTTOM | BOTTOM_RIGHT |
+---------------------------------+
```

See also

Fl_Align, FL_ALIGN_CENTER, etc.

Note

1. Bit positions not defined in the following constants of type Fl_Align are reserved for future extensions. Do not use.
2. The "magic values" (FL_ALIGN_ )LEFT_TOP, RIGHT_TOP, LEFT_BOTTOM, and RIGHT_BOTTOM must not be used together with FL_ALIGN_INSIDE. Use TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, or BOTTOM_RIGHT instead.
3. Although bits can be or’d together there are some unused/illegal combinations, for instance:
   - setting both FL_ALIGN_TOP and FL_ALIGN_BOTTOM in combinations other than those given in the Fl_Align constants below (magic values)
   - setting both FL_ALIGN_LEFT and FL_ALIGN_RIGHT in combinations other than those given in the Fl_Align constants below (magic values)
   - using one of the "magic values" (2) together with FL_ALIGN_INSIDE

Using illegal bit combinations or undefined bits may yield unexpected behavior, and this behavior may be changed without notice in future FLTK versions.

- typedef unsigned Fl_Align

  FLTK type for alignment control.
- const Fl_Align FL_ALIGN_CENTER = 0x0000
  Align the label horizontally in the middle.
- const Fl_Align FL_ALIGN_TOP = 0x0001
  Align the label at the top of the widget.
- const Fl_Align FL_ALIGN_BOTTOM = 0x0002
  Align the label at the bottom of the widget.
- const Fl_Align FL_ALIGN_LEFT = 0x0004
  Align the label at the left of the widget.
- const Fl_Align FL_ALIGN_RIGHT = 0x0008
  Align the label at the right of the widget.
- const Fl_Align FL_ALIGN_INSIDE = 0x0010
  Draw the label inside of the widget.
- const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
  If the label contains an image, draw the text on top of the image.
- const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000

Generated by Doxygen
If the label contains an image, draw the text below the image.

- `const Fl_Align FL_ALIGN_CLIP = 0x0040`
  All parts of the label that are larger than the widget will not be drawn.

- `const Fl_Align FL_ALIGN_WRAP = 0x0080`
  Wrap text that does not fit the width of the widget.

- `const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100`
  If the label contains an image, draw the text to the right of the image.

- `const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120`
  If the label contains an image, draw the text to the left of the image.

- `const Fl_Align FL_ALIGN_IMAGE_BACKDROP = 0x0200`
  If the label contains an image, draw the image or deimage in the background.

- `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_NOWRAP = 0x0000`
  Nothing, same as FL_ALIGN_CENTER, for back compatibility.

- `const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f`
  Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.

- `const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320`
  Mask value to test for image alignment flags.

Font Numbers

The following constants define the standard FLTK fonts:

- `typedef int Fl_Font`
  A font number is an index into the internal font table.

- `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_ITALIC = 2`
  Helvetica (or Arial) oblique.

- `const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3`
  Helvetica (or Arial) bold-italic.

- `const Fl_Font FL_COURIER = 4`
  Courier normal.

- `const Fl_Font FL_COURIER_BOLD = 5`
  Courier bold.

- `const Fl_Font FL_COURIER_ITALIC = 6`
  Courier oblique.
Courier italic.
• const Fl_Font FL_COURIER_BOLD_ITALIC = 7

Courier bold-italic.
• 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 = 10

Times roman italic.
• const Fl_Font FL_TIMES_BOLD_ITALIC = 11

Times roman bold-italic.
• const Fl_Font FL_SYMBOL = 12

Standard symbol font.
• 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_ZAPF_DINGBATS = 15

Zapf-dingbats font.
• const Fl_Font FL_FREE_FONT = 16

first one to allocate
• const Fl_Font FL_BOLD = 1

add this to helvetica, courier, or times
• const Fl_Font FL_ITALIC = 2

add this to helvetica, courier, or times
• const Fl_Font FL_BOLD_ITALIC = 3

add this to helvetica, courier, or times

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 => 0xrrggbii
    | | | |
    | | +--- 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.

• #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

typedef unsigned int Fl_Color  
An FLTK color value; see also Colors.

• const Fl_Color **FL_FOREGROUND_COLOR** = 0  
the default foreground color (0) used for labels and text

• const Fl_Color **FL_BACKGROUND2_COLOR** = 7  
the default background color for text, list, and valuator widgets

• const Fl_Color **FL_INACTIVE_COLOR** = 8  
the inactive foreground color

• const Fl_Color **FL_SELECTION_COLOR** = 15  
the default selection/highlight color

• const Fl_Color **FL_GRAY0** = 32  
• const Fl_Color **FL_DARK3** = 39  
• const Fl_Color **FL_DARK2** = 45  
• const Fl_Color **FL_DARK1** = 47  
• const Fl_Color **FL_BACKGROUND_COLOR** = 49  
• const Fl_Color **FL_LIGHT1** = 50  
• const Fl_Color **FL_LIGHT2** = 52  
• const Fl_Color **FL_LIGHT3** = 54  
• 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

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

• **FL_EXPORT** Fl_Color **fl_contrast** (Fl_Color fg, Fl_Color bg)  
Returns a color that contrasts with the background color.

• **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.

• Fl_Color **fl_lighter** (Fl_Color c)  
Returns a lighter version of the specified color.

• Fl_Color **fl_darker** (Fl_Color c)  
Returns a darker version of the specified color.

• Fl_Color **fl_rgb_color** (uchar r, uchar g, uchar b)  
Returns the 24-bit color value closest to r, g, b.

• Fl_Color **fl_rgb_color** (uchar g)  
Returns the 24-bit color value closest to g (grayscale).

• **FL_COLOR** **fl_gray_ramp** (int i)  
Returns a gray color value from black (i = 0) to white (i == FL_NUM_GRAY - 1).

• Fl_Color **fl_color_cube** (int r, int g, int b)  
Returns a color out of the color cube.
32.1 Enumerations.H File Reference

32.1.1 Detailed Description

This file contains type definitions and general enumerations.

32.1.2 Macro Definition Documentation

32.1.2.1 FL_ABI_VERSION

#define FL_ABI_VERSION FL_API_VERSION

The FLTK ABI (Application Binary Interface) version number as an int.

FL_ABI_VERSION is an int that describes the major, minor, and patch ABI version numbers in the same format as FL_API_VERSION.

The ABI version number FL_ABI_VERSION is usually the same as the API version FL_API_VERSION with the last two digits set to '00'.

FLTK retains the ABI (Application Binary Interface) during patch releases of the same major and minor versions. Examples:

<table>
<thead>
<tr>
<th>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>

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

32.1.2.2 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()
32.1.2.3 FL_IMAGE_LABEL

#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

Draws an image (Fl_Image) as the label.

This is useful for one particular part of an Fl_Multi_Label. Use Fl_Widget::image() and/or Fl_Widget::deimage() for normal widgets with images as labels.

32.1.2.4 FL_MAJOR_VERSION

#define FL_MAJOR_VERSION 1

The major release version of this FLTK library.

See also

   FL_VERSION

32.1.2.5 FL_MINOR_VERSION

#define FL_MINOR_VERSION 4

The minor release version for this library.

FLTK remains mostly source-code compatible between minor version changes.

32.1.2.6 FL_MULTI_LABEL

#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()

Draws a label that can comprise several parts like text and images.

See also

   Fl_Multi_Label

32.1.2.7 FL_PATCH_VERSION

#define FL_PATCH_VERSION 0

The patch version for this library.

FLTK remains binary compatible between patches.
32.1.2.8 FL_SYMBOL_LABEL

#define FL_SYMBOL_LABEL FL_NORMAL_LABEL

Sets the current label type and returns its corresponding Fl_Labeltype value.

FL_SYMBOL_LABEL is an alias for FL_NORMAL_LABEL.

'@' symbols can be drawn with normal labels as well.
This definition is for historical reasons only (forms compatibility). You should use FL_NORMAL_LABEL instead.

32.1.2.9 FL_VERSION

#define FL_VERSION

Value:

( (double)FL_MAJOR_VERSION +
  (double)FL_MINOR_VERSION * 0.01 +
  (double)FL_PATCH_VERSION * 0.0001 )

The FLTK version number as a double.

FL_VERSION is a double that describes the major, minor, and patch version numbers.

Version 1.2.3 is actually stored as 1.0203 to allow for more than 9 minor and patch releases.

Deprecated This double version number is retained for compatibility with existing program code. New code should use int FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

FL_VERSION is equivalent to (double)FL_API_VERSION / 10000.

See also

Fl::version() (deprecated as well)
FL_API_VERSION
Fl::api_version()

32.1.3 Typedef Documentation

32.1.3.1 Fl_Align

typedef unsigned Fl_Align

FLTK type for alignment control.
32.1.3.2 Fl_Font

typedef int Fl_Font

A font number is an index into the internal font table.

32.1.3.3 Fl_Fontsize

typedef int Fl_Fontsize

Size of a font in pixels.
This is the approximate height of a font in pixels.

32.1.4 Enumeration Type Documentation

32.1.4.1 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>

32.1.4.2 Fl_Boxtype

enum Fl_Boxtype

FLTK standard box types.
This enum defines the standard box types included with FLTK.

Note

The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. _FL← _SHADOW_BOX. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!
## Enumerations

<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_EMBOSSED_BOX</td>
<td>see figure 1</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</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_SHADOW_BOX</td>
<td>see figure 1, use FL_SHADOW_BOX</td>
</tr>
<tr>
<td>FL_SHADOW_FRAME</td>
<td>see figure 1, use FL_SHADOW_FRAME</td>
</tr>
<tr>
<td>FL_ROUNDED_BOX</td>
<td>see figure 1, use FL_ROUNDED_BOX</td>
</tr>
<tr>
<td>FL_RSHADOW_BOX</td>
<td>see figure 1, use FL_RSHADOW_BOX</td>
</tr>
<tr>
<td>FL_ROUNDED_FRAME</td>
<td>see figure 1, use FL_ROUNDED_FRAME</td>
</tr>
<tr>
<td>FL_RFLAT_BOX</td>
<td>see figure 1, use FL_RFLAT_BOX</td>
</tr>
<tr>
<td>FL_ROUND_UP_BOX</td>
<td>see figure 1, use FL_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_ROUND_DOWN_BOX</td>
<td>see figure 1, use FL_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_UP_BOX</td>
<td>see figure 1, use FL_DIAMOND_UP_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_DOWN_BOX</td>
<td>see figure 1, use FL_DOWN_BOX</td>
</tr>
<tr>
<td>FL_OVAL_BOX</td>
<td>see figure 1, use FL_OVAL_BOX</td>
</tr>
<tr>
<td>_FL_OSHADOW_BOX</td>
<td>see figure 1, use FL_OSHADOW_BOX</td>
</tr>
<tr>
<td>_FL_OVAL_FRAME</td>
<td>see figure 1, use FL_OVAL_FRAME</td>
</tr>
<tr>
<td>_FL_OFLAT_BOX</td>
<td>see figure 1, use FL_OFLAT_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_BOX</td>
<td>plastic version of FL_UP_BOX, use FL_PLASTIC_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_BOX</td>
<td>plastic version of FL_DOWN_BOX, use FL_PLASTIC_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_FRAME</td>
<td>plastic version of FL_UP_FRAME, use FL_PLASTIC_UP_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_FRAME</td>
<td>plastic version of FL_DOWN_FRAME, use FL_PLASTIC_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_UP_BOX</td>
<td>gtk+ version of FL_UP_BOX, use FL_GTK_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_BOX</td>
<td>gtk+ version of FL_DOWN_BOX, use FL_GTK_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_FRAME</td>
<td>gtk+ version of FL_UP_FRAME, use FL_GTK_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_FRAME</td>
<td>gtk+ version of FL_DOWN_FRAME, use FL_GTK_DOWN_FRAME</td>
</tr>
</tbody>
</table>
32.1.4.3 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.

Enumerator

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CURSOR_DEFAULT</td>
<td>the default cursor, usually an arrow.</td>
</tr>
<tr>
<td>FL_CURSOR_ARROW</td>
<td>an arrow pointer.</td>
</tr>
<tr>
<td>FL_CURSOR_CROSS</td>
<td>crosshair.</td>
</tr>
<tr>
<td>FL_CURSOR_WAIT</td>
<td>busy indicator (for instance hourglass).</td>
</tr>
<tr>
<td>FL_CURSOR_INSERT</td>
<td>I-beam.</td>
</tr>
<tr>
<td>FL_CURSOR_HAND</td>
<td>pointing hand.</td>
</tr>
<tr>
<td>FL_CURSOR_HELP</td>
<td>question mark pointer.</td>
</tr>
<tr>
<td>FL_CURSOR_MOVE</td>
<td>4-pointed arrow or hand.</td>
</tr>
</tbody>
</table>
Enumerations

```
<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CURSOR_NS</td>
<td>up/down resize.</td>
</tr>
<tr>
<td>FL_CURSOR_WE</td>
<td>left/right resize.</td>
</tr>
<tr>
<td>FL_CURSOR_NWSE</td>
<td>diagonal resize.</td>
</tr>
<tr>
<td>FL_CURSOR_NESW</td>
<td>diagonal resize.</td>
</tr>
<tr>
<td>FL_CURSOR_N</td>
<td>upwards resize.</td>
</tr>
<tr>
<td>FL_CURSOR_NE</td>
<td>upwards, right resize.</td>
</tr>
<tr>
<td>FL_CURSOR_E</td>
<td>rightwards resize.</td>
</tr>
<tr>
<td>FL_CURSOR_SE</td>
<td>downwards, right resize.</td>
</tr>
<tr>
<td>FL_CURSOR_S</td>
<td>downwards resize.</td>
</tr>
<tr>
<td>FL_CURSOR_SW</td>
<td>downwards, left resize.</td>
</tr>
<tr>
<td>FL_CURSOR_W</td>
<td>leftwards resize.</td>
</tr>
<tr>
<td>FL_CURSOR_NW</td>
<td>upwards, left resize.</td>
</tr>
<tr>
<td>FL_CURSOR_NONE</td>
<td>invisible.</td>
</tr>
</tbody>
</table>
```

32.1.4.4 Fl_Damage

```
enum Fl_Damage
{
    FL_DAMAGE_CHILD = 1, // A child needs to be redrawn.
    FL_DAMAGE_EXPOSE = 2, // The window was exposed.
    FL_DAMAGE_SCROLL = 3, // The Fl_Scroll widget was scrolled.
    FL_DAMAGE_OVERLAY = 4, // The overlay planes need to be redrawn.
    FL_DAMAGE_USER1 = 5,  // First user-defined damage bit.
    FL_DAMAGE_USER2 = 6,  // Second user-defined damage bit.
    FL_DAMAGE_ALL = 7,    // Everything needs to be redrawn.
};
```

32.1.4.5 Fl_Event

```
enum Fl_Event
{
    FL_EVENT_MOUSEMOVE = 1,  // User moved the cursor.
    FL_EVENT_MOUSEDOWN = 2,  // User pressed a mouse button.
    FL_EVENT_MOUSEUP = 3,    // User released a mouse button.
    FL_EVENT_KEYDOWN = 4,    // User pressed a key.
    FL_EVENT_KEYUP = 5,      // User released a key.
    FL_EVENT_RESIZE = 6,     // The window was resized.
    FL_EVENT_EXPOSE = 7,     // The window was exposed.
    FL_EVENT_SCROLL = 8,     // The Fl_Scroll widget was scrolled.
    FL_EVENT_OVERLAY = 9,    // The overlay planes need to be redrawn.
};
```

Generated by Doxygen
See also

Fl::event_text(), Fl::event_key(), class Fl::

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NO_EVENT</td>
<td>No event.</td>
</tr>
<tr>
<td>FL_PUSH</td>
<td>A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling Fl::event_button(). You find out the mouse position by calling Fl::event_x() and Fl::event_y(). A widget indicates that it &quot;wants&quot; the mouse click by returning non-zero from its Fl_Widget::handle() method. It will then become the Fl::pushed() widget and will get FL_DRAG and the matching FL_RELEASE events. If Fl_Widget::handle() returns zero then FLTK will try sending the FL_PUSH to another widget.</td>
</tr>
<tr>
<td>FL_RELEASE</td>
<td>A mouse button has been released. You can find out what button by calling Fl::event_button(). In order to receive the FL_RELEASE event, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_ENTER</td>
<td>The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.</td>
</tr>
<tr>
<td>FL_LEAVE</td>
<td>The mouse has moved out of the widget. In order to receive the FL_LEAVE event, the widget must return non-zero when handling FL_ENTER.</td>
</tr>
<tr>
<td>FL_DRAG</td>
<td>The mouse has moved with a button held down. The current button state is in Fl::event_state(). The mouse position is in Fl::event_x() and Fl::event_y(). In order to receive FL_DRAG events, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_FOCUS</td>
<td>This indicates an attempt to give a widget the keyboard focus. If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its handle() method. It then becomes the Fl::focus() widget and gets FL_KEYDOWN, FL_KEYUP, and FL_UNFOCUS events. The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check Fl::event_key() to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.</td>
</tr>
<tr>
<td>FL_UNFOCUS</td>
<td>This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.</td>
</tr>
</tbody>
</table>
### Enumerations

<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.</td>
</tr>
<tr>
<td>FL_KEYUP</td>
<td>Key release event.</td>
</tr>
<tr>
<td>FL_CLOSE</td>
<td>The user clicked the close button of a window. This event is used internally only to trigger the callback of Fl_Window derived classes. The default callback closes the window calling Fl_Window::hide().</td>
</tr>
<tr>
<td>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>
<tr>
<td>Enumerator</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FL_HIDE</td>
<td>This widget is no longer visible, due to <code>Fl_Widget::hide()</code> being called on it or one of its parents, or due to a parent window being minimized. <code>Fl_Widget::visible()</code> may still be true after this, but the widget is visible only if <code>visible()</code> is true for it and all its parents (use <code>Fl_Widget::visible_r()</code> to check this).</td>
</tr>
<tr>
<td>FL_SHOW</td>
<td>This widget is visible again, due to <code>Fl_Widget::show()</code> being called on it or one of its parents, or due to a parent window being restored. Child <code>Fl_Windows</code> respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass <code>FL_SHOW</code> to the base class <code>Fl_Widget::handle()</code> method!</td>
</tr>
<tr>
<td>FL_PASTE</td>
<td>You should get this event some time after you call <code>Fl::paste()</code>. The contents of <code>Fl::event_text()</code> is the text to insert and the number of characters is in <code>Fl::event_length()</code>.</td>
</tr>
<tr>
<td>FL_SELECTIONCLEAR</td>
<td>The <code>Fl::selection_owner()</code> will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.</td>
</tr>
<tr>
<td>FL_MOUSEWHEEL</td>
<td>The user has moved the mouse wheel. The <code>Fl::event_dx()</code> and <code>Fl::event_dy()</code> methods can be used to find the amount to scroll horizontally and vertically.</td>
</tr>
<tr>
<td>FL_DND_ENTER</td>
<td>The mouse has been moved to point at this widget. A widget that is interested in receiving drag’n’drop data must return 1 to receive <code>FL_DND_DRAG</code>, <code>FL_DND_LEAVE</code> and <code>FL_DND_RELEASE</code> events.</td>
</tr>
<tr>
<td>FL_DND_DRAG</td>
<td>The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag’n’drop data should indicate the possible drop position.</td>
</tr>
<tr>
<td>FL_DND_LEAVE</td>
<td>The mouse has moved out of the widget.</td>
</tr>
<tr>
<td>FL_DND_RELEASE</td>
<td>The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following <code>FL_PASTE</code> event.</td>
</tr>
<tr>
<td>FL_SCREEN_CONFIGURATION_CHANGED</td>
<td>The screen configuration (number, positions) was changed. Use <code>Fl::add_handler()</code> to be notified of this event.</td>
</tr>
<tr>
<td>FL_FULLSCREEN</td>
<td>The fullscreen state of the window has changed. This event is sent to the window's handle method.</td>
</tr>
<tr>
<td>FL_ZOOM_GESTURE</td>
<td>The user has made a zoom/pinch/magnification gesture (Mac OS platform only). The <code>Fl::event_dy()</code> method can be used to find magnification amount, <code>Fl::event_x()</code> and <code>Fl::event_y()</code> are set as well. This event is sent to the window's handle method.</td>
</tr>
<tr>
<td>FL_ZOOM_EVENT</td>
<td>A zoom event (ctrl/+/-/0/ or cmd/+/-/0/) was processed. Use <code>Fl::add_handler()</code> to be notified of this event.</td>
</tr>
</tbody>
</table>

32.1.4.6  Fl_Labeltype

```c
enum Fl_Labeltype
```

The `labeltype()` method sets the type of the label.
Note

The documented enum Fl_Labeltype contains some values (names) with leading underscores, e.g. _FL_IMAGE_LABEL. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

The following standard label types are included:

<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 labeltype to use for creating own labeltypes</td>
</tr>
</tbody>
</table>

32.1.4.7 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

<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>
32.1.5 Function Documentation

32.1.5.1 fl_box()

```c
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.

32.1.5.2 fl_color_cube()

```c
Fl_Color fl_color_cube (int r, int g, int b) [inline]
```

Returns a color out of the color cube.

r must be in the range 0 to FL_NUM_RED (5) minus 1, g must be in the range 0 to FL_NUM_GREEN (8) minus 1, b must be in the range 0 to FL_NUM_BLUE (5) minus 1.

To get the closest color to a 8-bit set of R,G,B values use:

```c
fl_color_cube(R * (FL_NUM_RED - 1) / 255,
G * (FL_NUM_GREEN - 1) / 255,
B * (FL_NUM_BLUE - 1) / 255);
```

32.1.5.3 fl_darker()

```c
Fl_Color fl_darker (Fl_Color c) [inline]
```

Returns a darker version of the specified color.

32.1.5.4 fl_define_FL_EMBOSSED_LABEL()

```c
Fl_Labeltype FL_EXPORT fl_define_FL_EMBOSSED_LABEL ()
```

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.

Internal use only.
32.1.5.5 fl_define_FL_ENGRAVED_LABEL()

**FL_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL ( )**

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.

Internal use only.

32.1.5.6 fl_define_FL_ICON_LABEL()

**FL_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL ( )**

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.

Internal use only.

32.1.5.7 fl_define_FL_IMAGE_LABEL()

**FL_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL ( )**

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.

Internal use only.

32.1.5.8 fl_define_FL_MULTI_LABEL()

**FL_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL ( )**

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.

Internal use only.

32.1.5.9 fl_define_FL_SHADOW_LABEL()

**FL_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL ( )**

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.

Internal use only.

32.1.5.10 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.
32.1.5.11  \texttt{fl\_frame()}

\begin{verbatim}
Fl_Boxtype fl_frame ( Fl_Boxtype b ) \[inline\]
\end{verbatim}

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.

32.1.5.12  \texttt{fl\_gray\_ramp()}

\begin{verbatim}
Fl_Color fl_gray_ramp ( int i ) \[inline\]
\end{verbatim}

Returns a gray color value from black \((i = 0)\) to white \((i = \text{FL\textunderscore NUM\textunderscore GRAY - 1})\).

\texttt{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:

\[ \text{fl\_gray\_ramp}(I \cdot (\text{FL\textunderscore NUM\textunderscore GRAY - 1}) / 255) \]

32.1.5.13  \texttt{fl\_lighter()}

\begin{verbatim}
Fl_Color fl_lighter ( Fl_Color c ) \[inline\]
\end{verbatim}

Returns a lighter version of the specified color.

32.1.5.14  \texttt{fl\_rgb\_color() \[1/2\]}

\begin{verbatim}
Fl_Color fl_rgb_color ( uchar r, uchar g, uchar b ) \[inline\]
\end{verbatim}

Returns the 24-bit color value closest to \(r, g, b\).

32.1.5.15  \texttt{fl\_rgb\_color() \[2/2\]}

\begin{verbatim}
Fl_Color fl_rgb_color ( uchar g ) \[inline\]
\end{verbatim}

Returns the 24-bit color value closest to \(g\) (grayscale).
32.1.6 Variable Documentation

32.1.6.1 FL_ALIGN_BOTTOM

const Fl_Align FL_ALIGN_BOTTOM = 0x0002

Align the label at the bottom of the widget.

32.1.6.2 FL_ALIGN_CENTER

const Fl_Align FL_ALIGN_CENTER = 0x0000

Align the label horizontally in the middle.

32.1.6.3 FL_ALIGN_CLIP

const Fl_Align FL_ALIGN_CLIP = 0x0040

All parts of the label that are larger than the widget will not be drawn.

32.1.6.4 FL_ALIGN_IMAGE_BACKDROP

const Fl_Align FL_ALIGN_IMAGE_BACKDROP = 0x0200

If the label contains an image, draw the image or deimage in the background.

32.1.6.5 FL_ALIGN_IMAGE_MASK

const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320

Mask value to test for image alignment flags.
32.1.6.6  FL_ALIGN_IMAGE_NEXT_TO_TEXT

const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100

If the label contains an image, draw the text to the right of the image.

32.1.6.7  FL_ALIGN_IMAGE_OVER_TEXT

const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000

If the label contains an image, draw the text below the image.

32.1.6.8  FL_ALIGN_INSIDE

const Fl_Align FL_ALIGN_INSIDE = 0x0010

Draw the label inside of the widget.

32.1.6.9  FL_ALIGN_LEFT

const Fl_Align FL_ALIGN_LEFT = 0x0004

Align the label at the left of the widget.

Inside labels appear left-justified starting at the left side of the widget, outside labels are right-justified and drawn to the left of the widget.

32.1.6.10  FL_ALIGN_LEFT_BOTTOM

const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d

Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.

32.1.6.11  FL_ALIGN_LEFT_TOP

const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007

Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.
32.1.6.12  FL_ALIGN_NOWRAP

const Fl_Align FL_ALIGN_NOWRAP = 0x0000

Nothing, same as FL_ALIGN_CENTER, for back compatibility.

32.1.6.13  FL_ALIGN_POSITION_MASK

const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f

Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.

32.1.6.14  FL_ALIGN_RIGHT

const Fl_Align FL_ALIGN_RIGHT = 0x0008

Align the label to the right of the widget.

32.1.6.15  FL_ALIGN_RIGHT_BOTTOM

const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e

Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.

32.1.6.16  FL_ALIGN_RIGHT_TOP

const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b

Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.

32.1.6.17  FL_ALIGN_TEXT_NEXT_TO_IMAGE

const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120

If the label contains an image, draw the text to the left of the image.
32.1.6.18  **FL_ALIGN_TEXT_OVER_IMAGE**

```c
const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
```

If the label contains an image, draw the text on top of the image.

32.1.6.19  **FL_ALIGN_TOP**

```c
const Fl_Align FL_ALIGN_TOP = 0x0001
```

Align the label at the top of the widget.

Inside labels appear below the top, outside labels are drawn on top of the widget.

32.1.6.20  **FL_ALIGN_WRAP**

```c
const Fl_Align FL_ALIGN_WRAP = 0x0080
```

Wrap text that does not fit the width of the widget.

32.1.6.21  **FL_NORMAL_SIZE**

```c
FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE
```

normal font size

**32.2  filename.H File Reference**

File names and URI utility functions.

```c
#include "Fl_Export.H"
#include <FL/platform_types.h>
```

**Macros**

- `#define FL_PATH_MAX 2048`
  
  *all path buffers should use this length*
32.3 Fl.cxx File Reference

**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 extension of a filename._

- FL_EXPORT void fl_filename_free_list (struct dirent ***l, 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 ***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)._

32.2.1 Detailed Description

File names and URI utility functions.

32.3 Fl.cxx File Reference

Implementation of the member functions of class Fl.

```c
#include "config_lib.h"
#include <FL/Fl.H>
#include <FL/platform.H>
#include "Fl_Screen_Driver.H"
#include "Fl_Window_Driver.H"
#include "Fl_System_Driver.H"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <ctype.h>
#include <stdlib.h>
#include "flstring.h"
```

Generated by Doxygen
Macros

- `#define FOREVER 1e20`

Functions

- `bool fl_clipboard_notify_empty (void)`
- `void fl_close_display ()`
- `const char * fl_filename_name (const char *name)`
  
  Gets the file name from a path.
- `FL_Window * fl_find (Window xid)`
- `void fl_fix_focus ()`
- `void fl_open_callback (void(*)(const char *))`
  
  Register a function called for each file dropped onto an application icon.
- `void fl_open_display ()`
  
  Opens the display.
- `int fl_send_system_handlers (void *)`
- `void fl_throw_focus (Fl_Widget *)`
- `void fl_trigger_clipboard_notify (int source)`
- `FL_EXPORT Window fl_xid_(const Fl_Window *)`

Variables

- `FL_EXPORT const char * fl_local_alt = Fl::system_driver()->alt_name()`
  
  String pointer used in shortcuts, you can change it to another language
- `FL_EXPORT const char * fl_local_ctrl = Fl::system_driver()->control_name()`
  
  String pointer used in shortcuts, you can change it to another language
- `int(* fl_local_grab)(int)`
- `FL_EXPORT const char * fl_local_meta = Fl::system_driver()->meta_name()`
  
  String pointer used in shortcuts, you can change it to another language
- `FL_EXPORT const char * fl_local_shift = Fl::system_driver()->shift_name()`
  
  String pointer used in shortcuts, you can change it to another language
- `Fl_Widget * fl_oldfocus`
- `Fl_Widget * fl_selection_requestor`

32.3.1 Detailed Description

Implementation of the member functions of class Fl.

32.3.2 Function Documentation

32.3.2.1 fl_open_display()

`void fl_open_display ( )`

Opens the display.

Automatically called by the library when the first window is `show()`'n. Does nothing if the display is already open.
32.4 Fl.H File Reference

Fl static class.

```c
#include <FL/Fl_Export.H>
#include <FL/platform_types.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!

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.
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

32.4.1 Detailed Description

Fl static class.

32.5 fl_arc.cxx File Reference

Utility functions for drawing arcs and circles.

```c
#include <FL/fl_draw.H>
#include <FL/math.h>
```

32.5.1 Detailed Description

Utility functions for drawing arcs and circles.

32.6 fl_ask.cxx File Reference

Utility functions for common dialogs.

```c
#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/platform.H>
#include "FL_Screen_Driver.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.

- **void 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_position (const int x, const int y, const int center)**
  Sets the preferred position for the common message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password()

- **void fl_message_position (Fl_Widget ∗widget)**
  Sets the preferred position for the common message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password()

- **int fl_message_position (int ∗x, int ∗y)**
  Gets the preferred position for the common message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password()

- **void fl_message_title (const char ∗title)**
  Sets the title of the dialog window used in many common dialogs.

- **void fl_message_title_default (const char ∗title)**
  Sets the default title of the dialog window used in many common dialogs.

- **const char ∗fl_password (const char ∗fmt, const char ∗defstr,...)**
  Shows an input dialog displaying the fmt message.

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
32.6.1 Detailed Description

Utility functions for common dialogs.

32.7 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_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 Fl_Widget

  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 int fl_message_hotspot (void)

• FL_EXPORT int FL_EXPORT const char FL_EXPORT const char 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 int FL_EXPORT const char FL_EXPORT const char FL_EXPORT Fl_Widget * fl_message_position (int x, int y, int center=0)

  Sets the preferred position for the common message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password()).

  • FL_EXPORT void fl_message_position (Fl_Widget *widget)
Sets the preferred position for the common message box used in many common dialogs like \texttt{fl_message()}, \texttt{fl_alert()}, \texttt{fl_ask()}, \texttt{fl_choice()}, \texttt{fl_input()}, \texttt{fl_password()}.

- \texttt{FL_EXPORT int fl_message_position (int *x=0, int *y=0)}
  
  Gets the preferred position for the common message box used in many common dialogs like \texttt{fl_message()}, \texttt{fl_alert()}, \texttt{fl_ask()}, \texttt{fl_choice()}, \texttt{fl_input()}, \texttt{fl_password()}.

- \texttt{void fl_message_position (Fl_Widget &widget)}

- \texttt{FL_EXPORT void fl_message_title (const char *title)}
  
  Sets the title of the dialog window used in many common dialogs.

- \texttt{FL_EXPORT void fl_message_title_default (const char *title)}
  
  Sets the default title of the dialog window used in many common dialogs.

- \texttt{FL_EXPORT int FL_EXPORT const char FL_EXPORT const char fl_password(const char *label, const char *deflt=0,...) __fl_attr(__format__(__printf__}}

\section*{Variables}

- \texttt{FL_EXPORT void FL_EXPORT void FL_EXPORT int __deprecated__}
- \texttt{FL_EXPORT const char * fl_cancel}
  
  string pointer used in common dialogs, you can change it to another language

- \texttt{FL_EXPORT const char * fl_close}
  
  string pointer used in common dialogs, you can change it to another language

- \texttt{FL_EXPORT Fl_Font fl_message_font_}
- \texttt{FL_EXPORT Fl_Fontsize fl_message_size_}

- \texttt{FL_EXPORT const char * fl_no}
  
  string pointer used in common dialogs, you can change it to another language

- \texttt{FL_EXPORT const char * fl_ok}
  
  string pointer used in common dialogs, you can change it to another language

- \texttt{FL_EXPORT const char * fl_yes}
  
  string pointer used in common dialogs, you can change it to another language

\subsection*{32.7.1 Detailed Description}

API for common dialogs.

\subsection*{32.7.2 Enumeration Type Documentation}

\subsubsection*{32.7.2.1 Fl_Beep}

\begin{verbatim}
enum Fl_Beep
\end{verbatim}

Different system beeps available.

\textbf{See also}

\begin{verbatim}
fl_beep(int)
\end{verbatim}
### Enumerator

<table>
<thead>
<tr>
<th></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>

### 32.7.3 Function Documentation

#### 32.7.3.1 fl_message_position()

```c
void fl_message_position (  
   Fl_Widget & widget ) [inline]
```

See also

`fl_message_position(Fl_Widget *widget)`.

### 32.8 fl_boxtype.cxx File Reference

drawing code for common box types.

```c
#include <FL/Fl.H>  
#include <FL/Fl_Widget.H>  
#include <FL/fl_draw.H>  
#include <config.h>
```

### Macros

- `#define D1 BORDER_WIDTH`
- `#define D2 (BORDER_WIDTH+BORDER_WIDTH)`
- `#define fl_border_box fl_rectbound`

  allow consistent naming
Functions

- `void fl_border_frame (int x, int y, int w, int h, Fl_Color c)`
  Draws a frame of type `FL_BORDER_FRAME`.

- `void fl_down_box (int x, int y, int w, int h, Fl_Color c)`
  Draws a box of type `FL_DOWN_BOX`.

- `void fl_down_frame (int x, int y, int w, int h, Fl_Color)`
  Draws a frame of type `FL_DOWN_FRAME`.

- `void fl_draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c)`
  Draws a box using given type, position, size and color.

- `void fl_embossed_box (int x, int y, int w, int h, Fl_Color c)`
  Draws a box of type `FL_EMBOSSED_BOX`.

- `void fl_embossed_frame (int x, int y, int w, int h, Fl_Color)`
  Draws a frame of type `FL_EMBOSSED_FRAME`.

- `void fl_engraved_box (int x, int y, int w, int h, Fl_Color c)`
  Draws a box of type `FL_ENGRAVED_BOX`.

- `void fl_engraved_frame (int x, int y, int w, int h, Fl_Color)`
  Draws a frame of type `FL_ENGRAVED_FRAME`.

- `void fl_flat_box (int x, int y, int w, int h, Fl_Color c)`
  Draws a box of type `FL_FLAT_BOX`.

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

- `void fl_frame2 (const char ∗s, int x, int y, int w, int h)`
  Draws a series of line segments around the given box.

- `const uchar ∗fl_gray_ramp ()`

- `void fl_internal_boxtype (Fl_Boxtype t, Fl_Box_Draw_F ∗f)`
  Sets the drawing function for a given box type.

- `void fl_no_box (int, int, int, 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`.

### 32.8.1 Detailed Description

drawing code for common box types.
32.8.2 Function Documentation

32.8.2.1 fl_internal_boxtype()

```c
void fl_internal_boxtype ( Fl_Boxtype t,
                          Fl_Box_Draw_F * f )
```

Sets the drawing function for a given box type.

Parameters

<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>

32.8.2.2 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.

32.9 fl_color.cxx File Reference

Color handling.

```c
#include <FL/Fl.H>
#include <FL/Fl_Device.H>
#include <FL/Fl_Graphics_Driver.H>
#include "fl_cmap.h"
```

Functions

- **FL_Color fl_color_average (FL_Color color1, FL_Color color2, float weight)**
  
  Returns the weighted average color between the two given colors.

- **FL_Color fl_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.
Variables

- unsigned fl_cmap [256]

32.9.1 Detailed Description

Color handling.

32.10 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.

32.10.1 Detailed Description

Fl_Color_Chooser widget .

32.11 Fl_compose.cxx File Reference

Utility functions to support text input.

```
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
```

32.11.1 Detailed Description

Utility functions to support text input.
32.12  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>
```

32.12.1 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!

32.13  Fl_Device.H File Reference

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

```c
#include <FL/Fl_Plugin.H>
#include <FL/platform_types.h>
#include <stdlib.h>
```

Classes

- class Fl_Device_Plugin
  
  This plugin socket allows the integration of new device drivers for special window or screen types.

- class Fl_Display_Device
  
  A display to which the computer can draw.

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

32.13.1 Detailed Description

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

32.14  Fl_Double_Window.cxx File Reference

Fl_Double_Window implementation.

```c
#include <FL/Fl.H>
#include <FL/platform.H>
#include <FL/Fl_Double_Window.H>
#include <FL/fl_draw.H>
#include "Fl_Window_Driver.H"
```
32.14.1 Detailed Description

`Fl_Double_Window` implementation.

32.15 `fl_draw.H` File Reference

Utility header to pull drawing functions together

```c
#include <FL/Enumerations.H>
#include <FL/Fl_Graphics_Driver.H>
```

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

```c
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 (int x, int y, int w, int h, double a1, double a2)`
  Draws ellipse sections using integer coordinates.
- `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_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.
- `FL_EXPORT void fl_begin_offscreen (Fl_Offscreen ctx)`
  Sends 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.
- `char fl_can_do_alpha_blending ()`
  Checks whether platform supports true alpha blending for RGBA images.
- `FL_EXPORT Fl_RGB_Image * fl_capture_window_part (Fl_Window *win, int x, int y, int w, int h)`
  Captures the content of a rectangular zone of a mapped window.
- 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(x,y,r) 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 a rectangle with the current clip region and returns the bounding box of the result.*

- void fl_clip_region (Fl_Region r)
  *Replaces the top of the clipping stack with a clipping region of any shape.*

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

- 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_Color fl_color ()
  *Returns the last fl_color() that was set.*

- 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_EXPORT 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.*

- FL_EXPORT void fl_delete_offscreen (Fl_Offscreen ctx)
  *Deletion of an offscreen graphics buffer.*

- 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_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.*

- FL_EXPORT void fl_draw (int angle, const char *str, 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 (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, 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, Fl_Align, Fl_Image *, int) to do the real work.*

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

- int fl_draw_pixmap (char **const data, int x, int y, Fl_Color bg=FL_GRAY)
  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.

- FL_EXPORT void fl_end_offscreen ()
  Quit 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.

- void fl_focus_rect (int x, int y, int w, int h)
  Draws a dotted rectangle, used to indicate keyboard focus on a widget.

- FL_EXPORT void fl_font (Fl_Font face, Fl_Fontsize fsize)
  Sets the current font, which is then used in various drawing routines.

- Fl_Font fl_font ()
  Returns the face set by the most recent call to fl_font().

- 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 WindowX/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_point (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_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.
• FLEXPORT void fl_rescale_offscreen (Fl_Offscreen &ctx)
  Adapts an offscreen buffer to a changed value of the scale factor.
• FL_EXPORT void fl_reset_spot (void)
• 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, double y)
  Concatenates scaling transformation onto the current one.
• void fl_scale (double x)
  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 *fl_shortcut_label (unsigned int shortcut)
  Get a human-readable string from a shortcut value.
• FL_EXPORT const char *fl_shortcut_label (unsigned int shortcut, const char **eom)
  Get a human-readable string from a shortcut value.
• Fl_Fontsize fl_size ()
  Returns the size set by the most recent call to fl_font().
• FLEXPORT void fl_text_extents (const char *, int &, int &, int &, int &, int &, int &h)
  Determines the minimum pixel dimensions of a nul-terminated string using the current fl_font().
• void fl_text_extents (const char *, int n, int &, int &, int &, int &, int &h)
  Determines the minimum pixel dimensions of a sequence of n characters using the current fl_font().
• 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 nul-terminated string using the current font face and size.
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_draw_shortcut

32.15.1 Detailed Description

utility header to pull drawing functions together

32.16 Fl_Image.H File Reference

**Fl_Image, Fl_RGB_Image** classes.

```
#include "Enumerations.H"
#include <stdlib.h>
#include "Fl_Widget.H"
```

Classes

- class Fl_Image
  
  Base class for image caching, scaling 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.
32.16.1 Detailed Description

*Fl_Image*, *Fl_RGB_Image* classes.

32.16.2 Enumeration Type Documentation

32.16.2.1 *Fl_RGB_Scaling*

```c
enum Fl_RGB_Scaling
```

The scaling algorithm to use for RGB images.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_RGB_SCALING_NEAREST</td>
<td>default RGB image scaling algorithm</td>
</tr>
<tr>
<td>FL_RGB_SCALING_BILINEAR</td>
<td>more accurate, but slower RGB image scaling algorithm</td>
</tr>
</tbody>
</table>

32.17 *Fl_Menu_Item.H* File Reference

```c
#include "Fl_Widget.H"
#include "Fl_Image.H"
#include <FL/platform_types.h>
```

Classes

- `struct Fl_Menu_Item`

  The *Fl_Menu_Item* structure defines a single menu item that is used by the *Fl_Menu_* class.

Typedefs

- typedef *Fl_Menu_Item* *Fl_Menu*

Enumerations

```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_RADIO = FL_MENU_RADIO,
  FL_PUP_INVISIBLE = FL_MENU_INVISIBLE, FL_PUP_SUBMENU = FL_SUBMENU_POINTER
}
```
Functions

- FL_EXPORT Fl_Shortcut fl_old_shortcut (const char *)

  Emulation of XForms named shortcuts.

32.17.1 Enumeration Type Documentation

32.17.1.1 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>

32.18 Fl_Native_File_Chooser.H File Reference

Fl_Native_File_Chooser widget.

#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>

Classes

- class Fl_Native_File_Chooser

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

32.18.1 Detailed Description

Fl_Native_File_Chooser widget.
32.19  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>
```

32.19.1  Detailed Description

implementation of class Fl_Paged_Device.

32.20  Fl_Paged_Device.H File Reference

declaration of class Fl_Paged_Device.

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

Classes

- class Fl_Paged_Device
  
  Represents page-structured drawing surfaces.

- struct Fl_Paged_Device::page_format
  
  width, height and name of a page format

Macros

- #define NO_PAGE_FORMATS 30 /* MSVC6 compilation fix */
  Number of elements in enum Page_Format.

32.20.1  Detailed Description

declaration of class Fl_Paged_Device.

32.21  Fl_PostScript.H File Reference


```
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>
```

Generated by Doxygen
Classes

- class Fl_EPS_File_Surface
  Encapsulated PostScript drawing surface.
- class Fl_PostScript_File_Device
  To send graphical output to a PostScript file.

Typedefs

- typedef int() Fl_PostScript_Close_Command(FILE *)

32.21.1 Detailed Description


32.22 Fl_Printer.H File Reference

declaration of class Fl_Printer.

#include <FL/Fl_Paged_Device.H>

Classes

- class Fl_Printer
  OS-independent print support.

32.22.1 Detailed Description

declaration of class Fl_Printer.

32.23 fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.

#include <FL/platform.H>
#include <FL/Fl_Graphics_Driver.H>

32.23.1 Detailed Description

Drawing and clipping routines for rectangles.
32.24 Fl_Shared_Image.H File Reference

Fl_Shared_Image class.

#include "Fl_Image.H"

Classes

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

Typedefs

• typedef Fl_Image *( Fl_Shared_Handler)(const char *, uchar *, header, int headerlen)

Functions

• FL_EXPORT void fl_register_images()
  
  Register the image formats.

32.24.1 Detailed Description

Fl_Shared_Image class.

32.24.2 Function Documentation

32.24.2.1 fl_register_images()

FL_EXPORT void fl_register_images()

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.

32.25 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.

32.25.1 Detailed Description

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

32.26 Fl_Sys_Menu_Bar.H File Reference

Definition of class Fl_Sys_Menu_Bar.

#include <FL/Fl_Menu_Bar.H>

Classes

- class Fl_Sys_Menu_Bar
  A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

Variables

- Fl_Sys_Menu_Bar * fl_sys_menu_bar
  The system menu bar.

32.26.1 Detailed Description

Definition of class Fl_Sys_Menu_Bar.

32.27 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.

32.27.1 Detailed Description

This file contains the definitions of the Fl_Tree class.

32.27.2 Enumeration Type Documentation

32.27.2.1 Fl_Tree_Reason

enum Fl_Tree_Reason

The reason the callback was invoked.

Enumerator

<table>
<thead>
<tr>
<th>Fl_Tree_Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_REASON_NONE</td>
<td>unknown reason</td>
</tr>
<tr>
<td>FL_TREE_REASON_SELECTED</td>
<td>an item was selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_DESELECTED</td>
<td>an item was de-selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_RESELECTED</td>
<td>an item was re-selected (double-clicked). See Fl_Tree_Item_Reselect_Mode to enable this.</td>
</tr>
<tr>
<td>FL_TREE_REASON_OPENED</td>
<td>an item was opened</td>
</tr>
<tr>
<td>FL_TREE_REASON_CLOSED</td>
<td>an item was closed</td>
</tr>
<tr>
<td>FL_TREE_REASON_DRAGGED</td>
<td>an item was dragged into a new place</td>
</tr>
</tbody>
</table>

32.28 Fl_Tree_Item.H File Reference

This file contains the definitions for Fl_Tree_Item.

#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>

Generated by Doxygen
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item_Array.H>
#include <FL/Fl_Tree_Prefs.H>

Classes

• class Fl_Tree_Item
  Tree widget item.

32.28.1 Detailed Description

This file contains the definitions for Fl_Tree_Item.

32.29 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.

Variables

• class FL_EXPORT Fl_Tree_Item

32.29.1 Detailed Description

This file defines a class that manages an array of Fl_Tree_Item pointers.

32.30 Fl_Tree_Prefs.H File Reference

This file contains the definitions for Fl_Tree's preferences.

#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_Select { FL_TREE_SELECT_NONE =0, FL_TREE_SELECT_SINGLE =1, FL_TREE_SELECT_MULTI =2, FL_TREE_SELECT_SINGLE_DRAGGABLE =3 }
  
  Tree selection style.

- enum Fl_Tree_Sort { FL_TREE_SORT_NONE =0, FL_TREE_SORT_ASCENDING =1, FL_TREE_SORT_DESCENDING =2 }
  
  Sort order options for items added to the tree.

32.30.1 Detailed Description

This file contains the definitions for Fl_Tree's preferences.

Fl_Tree_Prefs
  
  . . . .
  
  Fl_Tree
  
  |____ Fl_Tree_Item

32.30.2 Enumeration Type Documentation

32.30.2.1 Fl_Tree_Connector

enum Fl_Tree_Connector

Defines the style of connection lines between items.
### 32.30.2.2 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>FL_TREE_ITEM_DRAW_DEFAULT</th>
<th>If widget() defined, draw in place of label, and widget() tracks item height (default)</th>
</tr>
</thead>
<tbody>
<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>

### 32.30.2.3 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>FL_TREE_SELECTABLE_ONCE</th>
<th>Item can only be selected once (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECTABLE_ALWAYS</td>
<td>Enables FL_TREE_REASON_RESELECTED events for callbacks.</td>
</tr>
</tbody>
</table>

### 32.30.2.4 Fl_Tree_Select

**enum Fl_Tree_Select**

Tree selection style.

**Enumerator**

<table>
<thead>
<tr>
<th>FL_TREE_SELECT_NONE</th>
<th>Nothing selected when items are clicked.</th>
</tr>
</thead>
<tbody>
<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>
32.30.5 Fl_Tree_Sort

enum Fl_Tree_Sort

Sort order options for items added to the tree.

<table>
<thead>
<tr>
<th>Enumerator</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>

32.31 fl_types.h File Reference

This file contains simple "C"-style type definitions.

Typedefs

**Miscellaneous**

- typedef unsigned char uchar
  
  
  
  unsigned char
- typedef unsigned long ulong
  
  
  
  unsigned long
- typedef char * Fl_String
  
  Flexible length UTF-8 Unicode text.
- typedef const char * Fl_CString
  
  Flexible length UTF-8 Unicode read-only string.
- typedef unsigned int Fl_Shortcut
  
  16-bit Unicode character + 8-bit indicator for keyboard flags.
- typedef unsigned int Fl_Char
  
  24-bit Unicode character - upper 8 bits are unused

32.31.1 Detailed Description

This file contains simple "C"-style type definitions.

32.31.2 Typedef Documentation
32.31.2.1  Fl_CString

typedef const char* Fl_CString

Flexible length UTF-8 Unicode read-only string.

See also
    Fl_String

32.31.2.2  Fl_Shortcut

typedef unsigned int Fl_Shortcut

16-bit Unicode character + 8-bit indicator for keyboard flags.

Note
    This should be 24-bit Unicode character + 8-bit indicator for keyboard flags. The upper 8 bits are currently
    unused but reserved.

Due to compatibility issues this type and all FLTK shortcuts can only be used with 16-bit Unicode characters
(U+0000 .. U+FFFF) and not with the full range of unicode characters (U+0000 .. U+10FFFF).

This is caused by the bit flags FL_SHIFT, FL_CTRL, FL_ALT, and FL_META being all in the range 0x010000 .. 0x400000.

Todo
    Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-
    bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally
    intended. This would be possible if we could rely on all programs being coded with symbolic names and not
    hard coded bit values.

32.31.2.3  Fl_String

typedef char* Fl_String

Flexible length UTF-8 Unicode text.

Todo
    FIXME: temporary (?) typedef to mark UTF-8 and Unicode conversions
header for Unicode and UTF-8 character handling

#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

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_chdir (const char *path)
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

- 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 int fl_fopen (const char *f, const char *mode)
  Cross-platform function to open files with a UTF-8 encoded name.

- FL_EXPORT char *fl_getcwd (char *buf, int len)
  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 *fname, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.

- FL_EXPORT int fl_open_ext (const char *fname, int binary, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.

- FL_EXPORT int fl_putenv (const char *var)
  Cross-platform function to write environment variables with a UTF-8 encoded name or value.

- 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.
Returns the Unicode upper case value of $\text{ucs}$.

- **FL_EXPORT unsigned fl_ucs_toUtf16** (const unsigned ucs, unsigned short *dst, const unsigned dstlen)
  
  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.

- **FL_EXPORT int fl_unlink** (const char *fname)
  
  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)
  
  Move $p$ backward until it points to the start of a UTF-8 character.

- **FL_EXPORT int fl_utf8bytes** (unsigned ucs)
  
  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)
  
  Decode a single UTF-8 encoded character starting at $p$.

- **FL_EXPORT int fl_utf8encode** (unsigned ucs, char *buf)
  
  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.

- **FL_EXPORT 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.

- **FL_EXPORT unsigned fl_utf8from wchar_t** (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.

- **FL_EXPORT 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.

- **FL_EXPORT int fl_utf8locale** ()
  
  Return true if the "locale" seems to indicate that UTF-8 encoding is used.

- **FL_EXPORT 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.

- **FL_EXPORT unsigned fl_utf8from_mb** (char *dst, unsigned dstlen, const char *src, unsigned srclen)
  
  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

- **FL_EXPORT unsigned fl_utf8to_mb** (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  
  Convert a UTF-8 sequence into an array of 1-byte characters.

- **FL_EXPORT unsigned fl_utf8toUtf16** (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  
  Convert a UTF-8 sequence into an array of 16-bit characters.

- **FL_EXPORT unsigned fl_utf8towc** (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
  
  Converts a UTF-8 string into a wide character string.

- **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.
32.32.1 Detailed Description

header for Unicode and UTF-8 character handling

32.33 fl_vertex.cxx File Reference

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

```c
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>
```

32.33.1 Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

32.34 Fl_Widget.H File Reference

Fl_Widget, Fl_Label classes.

```c
#include "Enumerations.H"
#include "Fl.H"
```

Classes

- struct Fl_Label
  
  This struct stores all information for a text or mixed graphics label.

- class Fl_Widget
  
  Fl_Widget is the base class for all widgets in FLTK.

Macros

- #define FL_RESERVED_TYPE 100
  
  Reserved type numbers (necessary for my cheapo RTTI) start here.

Typedefs

- typedef void() Fl_Callback(Fl_Widget *, void *)
  
  Default callback type definition for all fltk widgets (by far the most used)

- typedef void() Fl_Callback0(Fl_Widget *)
  
  One parameter callback type definition passing only the widget.

- typedef void() Fl_Callback1(Fl_Widget *, long)
  
  Callback type definition passing the widget and a long data value.

- typedef Fl_Callback * Fl_Callback_p
  
  Default callback type pointer definition for all fltk widgets.
32.34.1 Detailed Description

`Fl_Widget`, `Fl_Label` classes.

32.34.2 Macro Definition Documentation

32.34.2.1 `FL_RESERVED_TYPE`

```c
#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.

32.35 `Fl_Window.H` File Reference

`Fl_Window` widget.

```c
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Bitmap.H>
#include <stdlib.h>
```

Classes

- class `Fl_Window`
  
  This widget produces an actual window.

Macros

- `#define FL_DOUBLE_WINDOW 0xF1`
  
  double window type id

- `#define FL_WINDOW 0xF0`
  
  window type id: all subclasses have type() >= this

32.35.1 Detailed Description

`Fl_Window` widget.
This file defines wrapper functions for OpenGL in FLTK.

```c
#include "Enumerations.H"
#include <GL/gl.h>
```

### Functions

- **FL_EXPORT void gl_color (Fl_Color i)**
  Sets the curent 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 ∗, 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 x, int y)**
  Draws a null-terminated string in the current font at the given position.

- **FL_EXPORT void gl_draw (const char ∗, float x, float y)**
  Draws a null-terminated 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 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 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, Fl_Align)**
  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)**
  Measure 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 void gl_texture_pile_height (int max)**
  Changes the maximum height of the pile of pre-computed string textures.

- **FL_EXPORT int gl_texture_pile_height ()**
  Returns the current maximum height of the pile of pre-computed string textures.

- **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.
32.36.1 Detailed Description

This file defines wrapper functions for OpenGL in FLTK.

To use OpenGL from within an FLTK application you MUST use gl_visual() to select the default visual before doing show() on any windows. Mesa will crash if you try to use a visual not returned by glxChooseVisual.

Historically, this did not always work well with Fl_Double_Window's! It can try to draw into the front buffer. Depending on the system this might either crash or do nothing (when pixmaps are being used as back buffer and GL is being done by hardware), work correctly (when GL is done with software, such as Mesa), or draw into the front buffer and be erased when the buffers are swapped (when double buffer hardware is being used).

32.36.2 Function Documentation

32.36.2.1 gl_color()

FL_EXPORT void gl_color ( Fl_Color i )

Sets the curent 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!

32.36.2.2 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

   gl_texture_pile_height(int)

32.36.2.3 gl_draw() [2/7]

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

   gl_texture_pile_height(int)
32.36.2.4  gl_draw() [3/7]

FL_EXPORT void gl_draw {
    const char * str,
    int x,
    int y )

Draws a nul-terminated string in the current font at the given position.

See also

    gl_texture_pile_height(int)

32.36.2.5  gl_draw() [4/7]

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

    gl_texture_pile_height(int)

32.36.2.6  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

    gl_texture_pile_height(int)
32.36.7  gl_draw() [6/7]

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  
    gl_texture_pile_height(int)

32.36.8  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().

32.36.9  gl_font()

FL_EXPORT void gl_font {
    int fontid,
    int size
}

Sets the current OpenGL font to the same font as calling fl_font().

See also  
    Fl::draw_GL_text_with_textures(int val)

32.36.10  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.
32.36.2.11 gl_rectf()

```c
void gl_rectf ( int x, int y, int w, int h ) [inline]
```

Fills the given rectangle with the current color.

See also

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

32.36.2.12 gl_texture_pile_height() [1/2]

```c
FL_EXPORT void gl_texture_pile_height ( int max )
```

Changes the maximum height of the pile of pre-computed string textures.

Strings that are often re-displayed can be processed much faster if this pile is set high enough to hold all of them.

Parameters

| max | Maximum height of the texture pile |

See also

```c
Fl::draw_GL_text_with_textures(int)
```

32.36.2.13 gl_texture_pile_height() [2/2]

```c
FL_EXPORT int gl_texture_pile_height ( void )
```

Returns the current maximum height of the pile of pre-computed string textures.

The default value is 100

See also

```c
Fl::draw_GL_text_with_textures(int)
```

32.37 mac.H File Reference

Mac OS X-specific symbols.

---

Generated by Doxygen
Classes

• class Fl_Mac_App_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.

Variables

• int fl_mac_os_version
  
  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13).

32.37.1 Detailed Description

Mac OS X-specific symbols.

32.38 numericsort.c File Reference

#include <ctype.h>
#include <stdlib.h>
#include <FL/platform_types.h>
#include <FL/filename.H>

Functions

• int fl_casenumericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-insensitive).

• int fl_numericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-sensitive).

32.38.1 Function Documentation
32.38.1.1  fl_casenumericsort()

```c
int fl_casenumericsort (  
    struct dirent ** A,  
    struct dirent ** B )
```

Compares directory entries alphanumerically (case-insensitive).

**Note**

This comparison is not (yet) UTF-8 aware.

**Todo**  Make comparison UTF-8 aware.

**See also**

fl_numericsort()

32.38.1.2  fl_numericsort()

```c
int fl_numericsort (  
    struct dirent ** A,  
    struct dirent ** B )
```

Compares directory entries alphanumerically (case-sensitive).

Numbers are compared without sign, i.e. "-" is not taken as a sign of following numerical values. The following list of files would be in ascending order (examples are ASCII and numbers only for simplicity):

1. 1zzz.txt  
2. 2xxx.txt  
3. 19uuu.txt  
4. 100aaa.txt  
5. file1z.txt  
6. file5a.txt  
7. file5z.txt  
8. file30z.txt  
9. file200a.txt  
10. temp+5.txt ("+" is lexically lower than ")  
11. temp-5.txt ("-" is not a sign)  
12. temp-100.txt (100 is bigger than 5, no sign)
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>first directory entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>B</td>
<td>second directory entry</td>
</tr>
</tbody>
</table>

Returns

comparison result (-1, 0, or +1)

Return values

<table>
<thead>
<tr>
<th>-1</th>
<th>A &lt; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A == B</td>
</tr>
<tr>
<td>+1</td>
<td>A &gt; B</td>
</tr>
</tbody>
</table>

Note

This comparison is not (yet) UTF-8 aware:

- UTF-8 characters are compared according to their binary values.
- Locale settings may influence the result in unexpected ways.
- The latter is particularly true for `fl_casenumericsort()`. This may be changed in a future release.

Todo

Make comparison UTF-8 aware.

See also

`fl_casenumericsort()`

32.39 platform_types.h File Reference

Definitions of platform-dependent types.

Macros

- `#define FL_COMMAND opaque`
  
  An alias for `FL_CTRL` on Windows and X11, or `FL_META` on MacOS X.

- `#define FL_CONTROL opaque`
  
  An alias for `FL_META` on Windows and X11, or `FL_CTRL` on MacOS X.
32.39.1 Detailed Description

Definitions of platform-dependent types.

The exact nature of these types varies with the platform. Therefore, portable FLTK applications should not assume these types have a specific size, or that they are pointers.

32.39.2 Typedef Documentation

32.39.2.1 fl_intptr_t

typedef opaque fl_intptr_t

An integral type large enough to store a pointer or a long value.

A pointer value can be safely cast to fl_intptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl_intptr_t can also store a long int value.

32.39.2.2 fl_uintptr_t

typedef opaque fl_uintptr_t

An unsigned integral type large enough to store a pointer or an unsigned long value.

A pointer value can be safely cast to fl_uintptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl_uintptr_t can also store an unsigned long int value.
Index

(remove

Fl_Browser, 415
～Fl_Check_Browser
Fl_Check_Browser, 477
～Fl_Double_Window
Fl_Double_Window, 519
～Fl_EPS_File_Surface
Fl_EPS_File_Surface, 523
～Fl_File_Chooser
Fl_File_Chooser, 533
～Fl_Group
Fl_Group, 583
～Fl_Help_View
Fl_Help_View, 605
～Fl_Image_Surface
Fl_Image_Surface, 630
～Fl_Input_
Fl_Input_, 641
～Fl_Menu_Window
Fl_Menu_Window, 718
～Fl_Native_File_Chooser
Fl_Native_File_Chooser, 731
～Fl_Plugin_Manager
Fl_Plugin_Manager, 755
～Fl_Preferences
Fl_Preferences, 779
～Fl_SVG_File_Surface
Fl_SVG_File_Surface, 882
～Fl_SVG_Image
Fl_SVG_Image, 887
～Fl_Scrollbar
Fl_Scrollbar, 833
～Fl_Shared_Image
Fl_Shared_Image, 842
～Fl_Surface_Device
Fl_Surface_Device, 878
～Fl_Table
Fl_Table, 908
～Fl_Table_Row
Fl_Table_Row, 924
～Fl_Text_Display
Fl_Text_Display, 967
～Fl_Widget
Fl_Widget, 1163
～Fl_Window
Fl_Window, 1214

abi_check
Fl, 377

abi_version

about

Fl_Sys_Menu_Bar, 893

absolute_top_line_number

Fl_Text_Display, 968

activate

Fl_Menu_Item, 704
Fl_Tree_Item, 1103
Fl_Widget, 1163

active

Fl_Menu_Item, 704
Fl_Widget, 1163

active_r

Fl_Widget, 1164

activevisible

Fl_Menu_Item, 705

add

Fl_Browser, 416
Fl_Chart, 471
Fl_Check_Browser, 478
Fl_File_Icon, 542
Fl_Input_Choice, 662
Fl_Menu_, 678, 680
Fl_Menu_Item, 705
Fl_Shared_Image, 842
Fl_Sys_Menu_Bar, 893, 895
Fl_Tree, 1054, 1055
Fl_Tree_Item, 1103, 1104
Fl_Tree_Item_Array, 1126

add_awake_handler_
Fl, 378

add_check
Fl, 378

add_clipboard_notify
Selection & Clipboard functions, 256

add_color
Fl_File_Icon, 543

add_default_key_bindings
Fl_Text_Editor, 1009

add_extra
Fl_File_Chooser, 534

add_fd
Fl, 379

add_handler
Events handling functions, 240

add_idle
Fl, 380

add_key_binding
Fl_Text_Editor, 1010
add_modify_callback
    Fl_Text_Buffer, 941
add_system_handler
    Events handling functions, 240
add_timeout
    Fl, 380
add_vertex
    Fl_File_Icon, 543
addPlugin
    Fl_Plugin_Manager, 755
address
    Fl_Text_Buffer, 941, 942
align
    Fl_Widget, 1164
angle1
    Fl.Dial, 517
ansi
    Fl_Simple_Terminal, 852, 853
api_version
    Fl, 380
append
    Fl_Input_, 641
    Fl_Simple_Terminal, 853
    Fl_Text_Buffer, 942
appendfile
    Fl_Text_Buffer, 942
arg
    Fl, 381
args
    Fl, 381, 382
argument
    Fl_Menu_Item, 705
    Fl_Widget, 1165
array
    Fl_Group, 584
    Fl_RGB_Image, 818
    Fl_Table, 909
as_gl_window
    Fl_Gl_Window, 568
    Fl_Widget, 1165
as_group
    Fl_Group, 584
    Fl_Widget, 1166
as_shared_image
    Fl_Image, 620
    Fl_Shared_Image, 842
as_svg_image
    Fl_RGB_Image, 815
    Fl_SVG_Image, 887
as_window
    Fl_Widget, 1166
    Fl_Window, 1215
atclose
    Windows handling functions, 237
autosize
    Fl_Chart, 471
awake
    Multithreading support functions, 313
    Fl_Color Chooser, 501
    Fl_Rect, 806
background
    Fl, 382
background2
    Fl, 383
bbox
    Fl_Browser_, 444
    Fl_Scroll, 827
begin
    Fl_Group, 585
begin_job
    Fl_Paged_Device, 746
    Fl_PostScript_File_Device, 766, 767
    Fl_Printer, 795
begin_page
    Fl_Paged_Device, 746
    Fl_PostScript_File_Device, 767
    Fl_Printer, 796
belowmouse
    Events handling functions, 241
bitmap
    Fl_FormsBitmap, 555
border
    Fl_Window, 1215
bottomline
    Fl_Browser, 416
bound_key_function
    Fl_Text_Editor, 1010
bounds
    Fl_Chart, 472
    Fl_Group, 585
    Fl_Slider, 864
    Fl_Valuator, 1138
box
    Fl_Widget, 1167
box_color
    Fl, 383
box_dh
    Fl, 383
box_dw
    Fl, 383
box_dx
    Fl, 384
box_dy
    Fl, 384
buffer
    Fl_Text_Display, 968, 969
buffer_modified_cb
    Fl_Text_Display, 969
buffer_predelete_cb
    Fl_Text_Display, 970
byte_at
    Fl_Text_Buffer, 942
CORE_READ_OK
    Fl_Preferences, 791
CORE_WRITE_OK
INDEX

FL_Preferences, 792
Cairo Support Functions and Classes, 319
cairo_autolink_context, 319
cairo_cc, 320
cairo_make_current, 320
cairo_autolink_context
    Cairo Support Functions and Classes, 319
cairo_cc
    Cairo Support Functions and Classes, 320
cairo_make_current
    Cairo Support Functions and Classes, 320
calc_dimensions
    FL_Tree, 1056
calc_item_height
    FL_Tree_Item, 1104
calc_last_char
    FL_Text_Display, 970
calc_line_starts
    FL_Text_Display, 970
calc_tree
    FL_Tree, 1056
callback
    FL_Menu_Item, 706, 707
    FL_Table, 909
    FL_Widget, 1168, 1169
Callback function typedefs, 229
    FL_Event_Dispatch, 230
callback_col
    FL_Table, 910
callback_context
    FL_Table, 910
callback_item
    FL_Tree, 1056, 1057
callback_reason
    FL_Tree, 1057
callback_row
    FL_Table, 910
can_do
    FL_Gl_Window, 568, 569
can_do_overlay
    FL_Gl_Window, 569
c
    FL_Cairo_State, 467
changed
    FL_Input_Choice, 663
    FL_Widget, 1169
char_at
    FL_Text_Buffer, 943
check
    FL, 384
    FL_Menu_Item, 707
check_all
    FL_Check_Browser, 478
check_none
    FL_Check_Browser, 478
checkbox
    FL_Menu_Item, 707
checked

FL_Check_Browser, 478
FL_Menu_Item, 707
child
    FL_Group, 586
    FL_Table, 910
    FL_Tree_Item, 1104
children
    FL_Table, 911
clamp
    FL_Wrapper, 1138
clear
    FL_Browser, 417
    FL_Button, 463
    FL_Check_Browser, 479
    FL_File_Icon, 544
    FL_Group, 586
    FL_Input_Choice, 663
    FL_Menu, 681
    FL_Menu_Item, 708
    FL_Menu_Item, 708
    FL_Scroll, 827
    FL_Simple_Terminal, 854
    FL_Sys_Menu_Bar, 895
    FL_Table, 911
    FL_Table_Row, 925
    FL_Tree, 1057
    FL_Tree_Item_Array, 1126
clear_active
    FL_Widget, 1170
clear_border
    FL_Wrapper, 1215
clear_changed
    FL_Input_Choice, 663
    FL_Widget, 1170
clear_children
    FL_Tree, 1058
clear_damage
    FL_Widget, 1170
clear_modal_states
    FL_Wrapper, 1215
clear_output
    FL_Wrapper, 1171
clear_overlay
    FL_Menu_Window, 719
clear_rect
    FL_Text_Display, 972
clear_selection
    FL_Help_View, 606
clear_submenu
    FL_Menu, 681
    FL_Sys_Menu_Bar, 896
clear_visible
    FL_Wrapper, 1171
clear_visible_focus
    FL_Wrapper, 1171
clear_widget_pointer
    Safe widget deletion support functions, 316
client_area
    FL_Tabs, 931

Generated by Doxygen
INDEX 1325

Fl_Widget, 1174

copy_tooltip
Fl_Widget, 1174

count
Fl_File_Chooser, 535
Fl_Help_Font_Stack, 600
Fl_Image, 621
Fl_Native_File_Chooser, 731
count_displayed_characters
Fl_Text_Buffer, 944
count_lines
Fl_Text_Buffer, 944
Fl_Text_Display, 972
create_window_menu
Fl_Sys_Menu_Bar, 896
current
Fl_Group, 587
Fl_Tooltip, 1039
Fl_Window, 1216
current_
Fl_Window, 1235
current_style_index
Fl_Simple_Terminal, 854
cursor
Fl_Window, 1216, 1217
cursor2rowcol
Fl_Table, 912
cursor_color
Fl_Input_, 642
Fl_Text_Display, 973
Fl_Value_Input, 1143, 1144
cursor_style
Fl_Text_Display, 973
custom_application_menu_items
Fl_Mac_App_Menu, 674
cut
Fl_Input_, 643
d
Fl_Image, 621
damage
Fl, 385
Fl_Widget, 1174, 1175
damage_resize
Fl_Widget, 1176
damage_zone
Fl_Table, 913
data
Fl_Browser, 418, 419
Fl_Image, 621
deactivate
Fl_Menu_Item, 708
Fl_Tree_Item, 1105
Fl_Widget, 1176
decorated_h
Fl_Window, 1217
decorated_w
Fl_Window, 1218
default_atclose

Windows handling functions, 235
default_callback
Fl_Widget, 1176
default_cursor
Fl_Window, 1218
default_icon
Fl_Window, 1218
default_icons
Fl_Window, 1219
default_key_function
Fl_Text_Editor, 1010
default_xclass
Fl_Window, 1219, 1220
dejimage
Fl_Widget, 1177, 1178
delay
Fl_Tooltip, 1039
delete_widget
Safe widget deletion support functions, 316
deleteEntry
Fl_Preferences, 779
deleteGroup
Fl_Preferences, 780
detected
Fl_Widget_Tracker, 1208
deleting
Fl_Browser_, 444
deparent
Fl_Tree_Item, 1105
Fl_Tree_Item_Array, 1126
depth
Fl_Tree_Item, 1105
desaturate
Fl_Image, 621
Fl_Pixmap, 751
Fl_RGB_Image, 816
Fl_SVG_Image, 888
Fl_Shared_Image, 844
Fl_Tiled_Image, 1031
deselect
Fl_Browser_, 444
Fl_Tree, 1060
deselect_all
Fl_Tree, 1061
Fl_Tree_Item, 1105
direction
Fl_Timer, 1033
directory
Fl_Browser, 535
Fl_Help_View, 606
Fl_Native_File_Chooser, 731
disable
Fl_Tooltip, 1039
disable_im
Events handling functions, 242
display
Fl, 385
Fl_Browser, 419

Generated by Doxygen
FL_Browser_, 445
FL_Tree, 1062
display_insert
  FL_Text_Display, 974
displayed
  FL_Browser, 419
  FL_Browser_, 445
  FL_Tree, 1062
dnd
  Selection & Clipboard functions, 257
dnd_text_ops
  FL, 385
do_callback
  FL_Menu_Item, 708
  FL_Table, 913
  FL_Widget, 1178, 1179
do_widget_deletion
  Safe widget deletion support functions, 316
down_box
  FL_Button, 463
  FL_File_Input, 549, 550
  FL_Menu_, 682
draw
  FL_Adjuster, 402
  FL_Bitmap, 405
  FL_Box, 410
  FL_Button, 464
  FL_Chart, 472
  FL_Choice, 488
  FL_Clock_Output, 496
  FL_Counter, 511
  FL_Dial, 517
  FL_File_Icon, 544
  FL_FormsBitmap, 556
  FL_FormsPixmap, 558
  FL_FormsText, 559
  FL_Free, 561
  FL_Gl_Window, 570
  FL_Glut_Window, 579
  FL_Group, 587
  FL_Help_View, 606
  FL_Image, 621, 622
  FL_Input, 636
  FL_Label, 669
  FL_Light_Button, 672
  FL_Menu_Bar, 695
  FL_Menu_Button, 698
  FL_Menu_Item, 709
  FL_Pack, 742
  FL_Pixmap, 752
  FL_Positioner, 761
  FL_Progress, 801
  FL_RGB_Image, 816
  FL_Return_Button, 811
  FL_Roller, 820
  FL_SVG_Image, 888
  FL_Scroll, 827
  FL_Scrollbar, 833
  FL_Shared_Image, 844
  FL_Simple_Terminal, 855
  FL_Slider, 864
  FL_Sys_Menu_Bar, 896
  FL_Table, 913
  FL_Tabs, 932
  FL_Text_Display, 974
  FL_Tiled_Image, 1031
  FL_Timer, 1034
  FL_Tree_Item, 1105
  FL_Value_Input, 1144
  FL_Value_Output, 1149
  FL_Value_Slider, 1153
  FL_Widget, 1179
  FL_Widget_Surface, 1204
  FL_Window, 1220
  FL_Wizard, 1237
draw_GL_text_with_textures
  FL, 386
draw_box
  FL_Widget, 1180
draw_box_active
  FL, 385
draw_cell
  FL_Table, 913
draw_child
  FL_Group, 587
draw_children
  FL_Group, 588
draw_cursor
  FL_Text_Display, 974
draw_decorated_window
  FL_Copy_Surface, 507
  FL_Printer, 796
  FL_Widget_Surface, 1204
draw_empty
  FL_Image, 622
draw_focus
  FL_Widget, 1180
draw_horizontal_connector
  FL_Tree_Item, 1106
draw_item_content
  FL_Tree_Item, 1106
draw_label
  FL_Widget, 1180, 1181
draw_line_numbers
  FL_Text_Display, 975
draw_outside_label
  FL_Group, 588
draw_overlay
  FL_Glut_Window, 580
  FL_Overlay_Window, 739
draw_range
  FL_Text_Display, 975
draw_scaled
  FL_Image, 622
draw_string
  FL_Text_Display, 975
INDEX

Drawing functions, 278
fl_add_symbol, 283
fl_arc, 283, 284
fl_begin_complex_polygon, 285
fl_begin_offscreen, 285
fl_begin_points, 286
fl_can_do_alpha_blending, 286
fl_capture_window_part, 286
fl_circle, 287
fl_clip, 282
fl_clip_box, 287
fl_clip_region, 288, 289
fl_copy_offscreen, 289
fl_create_offscreen, 289
fl_cursor, 290
fl_curve, 290
fl_delete_offscreen, 291
fl_draw, 291, 292
fl_draw_box, 292
fl_draw_image, 293
fl_draw_image_mono, 294
fl_draw_pixmap, 295
fl_draw_symbol, 296
fl_expand_text, 296
fl_frame, 296
fl_frame2, 297
fl_gap, 297
fl_line_style, 297
fl_measure, 298
fl_measure_pixmap, 299
fl_mult_matrix, 299
fl_not_clipped, 300
fl_oldShortcut, 300
fl_pie, 301
fl_polygon, 302
fl_pop_clip, 303
fl_push_clip, 303
fl_push_matrix, 303
fl_read_image, 303
fl_rect, 304
fl_rect2, 304
fl_rescale_offscreen, 305
fl_reset_spot, 305
fl_rotate, 305
fl_scale, 306
fl_scroll, 306
fl_set_spot, 307
fl_set_status, 307
fl_shortcut_label, 307, 309
fl_transform_dx, 309
fl_transform_dy, 309
fl_transform_x, 311
fl_transform_y, 311
fl_transformed_vertex, 311
fl_translate, 311
fl_vertex, 312
drawtext
Fl_Input, 644
driver
Fl_EPS_File_Surface, 524
Fl_Surface_Device, 878
ERRORS_TO_CP1252
Unicode and UTF-8 functions, 323
ERRORS_TO_ISO8859_1
Unicode and UTF-8 functions, 323
empty_vlines
Fl_Text_Display, 977
enable
Fl_Tooltip, 1039
enable_im
Events handling functions, 242
enabled
Fl_Tooltip, 1040
draw_text
Fl_Text_Display, 976
draw_vertical_connector
Fl_Tree_Item, 1108
draw_vline
Fl_Text_Display, 976
drawbgcolor
Fl_Tree_Item, 1108
drawfgcolor
Fl_Tree_Item, 1108
Enumeration.H, 1247
FL_ABI_VERSION, 1259
FL_ALIGN_BOTTOM, 1273
FL_ALIGN_CENTER, 1273
FL_ALIGN_CLIP, 1273
FL_ALIGN_IMAGE_BACKDROP, 1273
FL_ALIGN_IMAGE_MASK, 1273
FL_ALIGN_IMAGE_NEXT_TO_TEXT, 1273
FL_ALIGN_IMAGE_OVER_TEXT, 1274
FL_ALIGN_INSIDE, 1274
FL_ALIGN_LEFT_BOTTOM, 1274
FL_ALIGN_LEFT_TOP, 1274
FL_ALIGN_LEFT, 1274
FL_ALIGN_NOWRAP, 1274

Generated by Doxygen
FL_ALIGN_POSITION_MASK, 1275
FL_ALIGN_RIGHT_BOTTOM, 1275
FL_ALIGN_RIGHT_TOP, 1275
FL_ALIGN_RIGHT, 1275
FL_ALIGN_TEXT_NEXT_TO_IMAGE, 1275
FL_ALIGN_TEXT_OVER_IMAGE, 1275
FL_ALIGN_TOP, 1276
FL_ALIGN_WRAP, 1276
FL_API_VERSION, 1259
FL_IMAGE_LABEL, 1259
FL_MAJOR_VERSION, 1260
FL_MINOR_VERSION, 1260
FL_MULTI_LABEL, 1260
FL_NORMAL_SIZE, 1276
FL_PATCH_VERSION, 1260
FL_SYMBOL_LABEL, 1260
FL_VERSION, 1261
Fl_Align, 1261
Fl_Boxtype, 1262
Fl_Cursor, 1264
Fl_Damage, 1265
Fl_Event, 1265
Fl_Font, 1261
Fl_Fontsize, 1262
Fl_Labeltype, 1268
Fl_When, 1269
fl_box, 1270
fl_color_cube, 1270
fl_darker, 1270
fl_define_FL_EMBOSSED_LABEL, 1270
fl_define_FL_ENGRAVED_LABEL, 1270
fl_define_FL_ICON_LABEL, 1271
fl_define_FL_IMAGE_LABEL, 1271
fl_define_FL_MULTI_LABEL, 1271
fl_define_FL_SHADOW_LABEL, 1271
fl_down, 1271
fl_frame, 1271
fl_gray_ramp, 1272
fl_lighter, 1272
fl_RGB_color, 1272
errmsg
Fl_Native_File_Chooser, 732
error
Common Dialogs classes and functions, 358
errorcolor
Fl_File_Input, 550
event
Events handling functions, 243
event_alt
Events handling functions, 243
event_button
Events handling functions, 243
event_button1
Events handling functions, 243
event_button2
Events handling functions, 244
event_button3
Events handling functions, 244
event_buttons
Events handling functions, 244
event_clicks
Events handling functions, 244, 245
event_clipboard
Events handling functions, 245
event_clipboard_type
Events handling functions, 245
event_command
Events handling functions, 245
event_ctrl
Events handling functions, 245
event_dispatch
Events handling functions, 246
event_dx
Events handling functions, 246
event_dy
Events handling functions, 246
event_inside
Events handling functions, 247
event_is_click
Events handling functions, 248
event_key
Events handling functions, 248
event_length
Events handling functions, 249
event_original_key
Events handling functions, 249
event_shift
Events handling functions, 249
event_state
Events handling functions, 250
event_text
Events handling functions, 250
event_x_root
Events handling functions, 251
event_y_root
Events handling functions, 251
Events handling functions, 238
add_handler, 240
add_system_handler, 240
belowmouse, 241
compose, 241
compose_reset, 242
disable_im, 242
enable_im, 242
event, 243
event_alt, 243
event_button, 243
event_button1, 243
event_button2, 244
event_button3, 244
event_buttons, 244
event_clicks, 244, 245
event_clipboard, 245
event_clipboard_type, 245
event_command, 245
event_ctrl, 245
file_access
   Fl_Preferences, 781, 782
file_encoding_warning_message
   Fl_Text_Buffer, 957
filename
   Fl_Help_View, 606
   Fl_Native_File_Chooser, 732
filename.H, 1276
filetype
   Fl_File_Browser, 527
   Fl_File_Chooser, 535
   Fl_Native_File_Chooser, 732
filter_value
   Fl_File_Chooser, 536
   Fl_Native_File_Chooser, 733
find
   Fl_File_Icon, 544
   Fl_Group, 588
   Fl_Help_View, 606
   Fl_Shared_Image, 845
find_cell
   Fl_Table, 915
find_child
   Fl_Tree_Item, 1109
find_child_item
   Fl_Tree_Item, 1109, 1110
find_clicked
   Fl_Tree, 1064
   Fl_Tree_Item, 1110
find_index
   Fl_Menu_, 682, 683
find_item
   Fl_Browser_, 445
   Fl_Menu_, 684
   Fl_Tree, 1064
   Fl_Tree_Item, 1111
find_line
   Fl_Browser, 420
find_line_end
   Fl_Text_Display, 977
find_shortcut
   Fl_Menu_Item, 709
find_wrap_range
   Fl_Text_Display, 978
find_x
   Fl_Text_Display, 978
findchar_backward
   Fl_Text_Buffer, 944
findchar_forward
   Fl_Text_Buffer, 945
first
   Fl_File_Icon, 544
   Fl_Menu_Item, 709, 710
   Fl_Tree, 1065
first_selected_item
   Fl_Tree, 1065
first_visible
   Fl_Tree, 1065
first_visible_item
   Fl_Tree, 1066
first_window
   Windows handling functions, 235, 236
Fl, 367
   abi_check, 377
   abi_version, 378
   add_awake_handler_, 378
   add_check, 378
   add_fd, 379
   add_idle, 380
   add_timeout, 380
   api_version, 380
   arg, 381
   args, 381, 382
   background, 382
   background2, 383
   box_color, 383
   box_dh, 383
   box_dw, 383
   box_dx, 384
   box_dy, 384
   check, 384
   damage, 385
   display, 385
   dnd_text_ops, 385
   draw_GL_text_with_textures, 386
   draw_box_active, 385
   Fl_Option, 377
   flush, 386
   foreground, 387
   get_awake_handler_, 387
   get_boxtype, 387
   get_system_colors, 387
   gl_visual, 387
   help, 400
   idle, 400
   insertion_point_location, 388
   is_scheme, 388
   menu_linespacing, 389
   option, 389, 390
   own_colormap, 391
   program_should_quit, 391
   readqueue, 391
   ready, 392
   release, 392
   reload_scheme, 392
   remove_check, 393
   remove_fd, 393
   remove_timeout, 393
   repeat_timeout, 393
   reset_marked_text, 394
   run, 394
   scheme, 394
   scrollbar_size, 395
   set_box_color, 396
set_boxtype, 396
set_idle, 396
set_labeltype, 397
use_high_res_GL, 397
version, 398
visible_focus, 398
visual, 398
wait, 399
Fl.cxx, 1277
   fl_open_display, 1278
Fl.H, 1279
Fl_Adjuster, 401
draw, 402
   Fl_Adjuster, 402
   handle, 402
   soft, 403
Fl_Align
  Enumerations.H, 1261
Fl_BMP_Image, 407
   Fl_BMP_Image, 407, 408
Fl_Beep
   fl_ask.H, 1283
Fl_Bitmap, 403
copy, 405
draw, 405
   Fl_Bitmap, 404, 405
   label, 406
   uncache, 406
Fl_Box, 408
draw, 410
   Fl_Box, 409
   handle, 410
Fl_Boxtype
   Enumerations.H, 1262
Fl_Browser, 411
   _remove, 415
   add, 416
   bottomline, 416
   clear, 417
   column_char, 417
   column_widths, 417, 418
   data, 418, 419
   display, 419
   displayed, 419
   find_line, 420
   Fl_Browser, 415
   format_char, 420, 421
   full_height, 421
   hide, 422
   icon, 422, 423
   incr_height, 423
   insert, 423, 424
   item_at, 424
   item_draw, 425
   item_first, 425
   item_height, 426
   item_last, 426
   item_next, 427
   item_prev, 427
   item_select, 428
   item_selected, 428
   item_swap, 428
   item_text, 429
   item_width, 429
   lineno, 430
   lineposition, 430
   load, 431
   make_visible, 431
   middleline, 432
   move, 432
   remove, 432
   remove_icon, 433
   replace, 433
   select, 433
   selected, 434
   show, 434
   size, 435
   swap, 435
   text, 436
   textsize, 436
   topline, 437
   value, 437, 438
   visible, 438
Fl_Browser_, 439
   bbox, 444
   deleting, 444
   deselect, 444
   display, 445
   displayed, 445
   find_item, 445
   Fl_Browser_, 443
   full_height, 446
   full_width, 446
   handle, 446
   has_scrollbar, 447
   hposition, 447
   hscrollbar, 460
   incr_height, 448
   inserting, 448
   item_at, 448
   item_first, 450
   item_height, 450
   item_last, 450
   item_next, 451
   item_prev, 451
   item_quick_height, 451
   item_select, 452
   item_selected, 452
   item_swap, 452
   item_text, 453
   item_width, 453
   leftedge, 454
   new_list, 454
   position, 454
   redraw_line, 455
   redraw_lines, 455
replacing, 455
resize, 456
scrollbar, 460
scrollbar_left, 456
scrollbar_right, 456
scrollbar_size, 456
scrollbar_width, 457
select, 457
select_only, 458
selection, 458
sort, 458
swapping, 459
textfont, 459
Fl_Button, 460
clear, 463
down_box, 463
draw, 464
Fl_Button, 462
handle, 464
set, 465
shortcut, 465
value, 466
Fl_CString
fl_types.h, 1305
Fl_Cairo_State, 466
cc, 467
Fl_Cairo_Window, 467
set_draw_cb, 468
Fl_Chart, 469
add, 471
autosize, 471
bounds, 472
draw, 472
Fl_Chart, 470
insert, 472
maxsize, 473
replace, 473
textcolor, 474
textfont, 474
textsize, 474
Fl_Check_Browser, 475
~Fl_Check_Browser, 477
add, 478
check_all, 478
check_none, 478
checked, 478
clear, 479
Fl_Check_Browser, 477
handle, 479
item_at, 479
item_first, 480
item_height, 480
item_next, 480
item_prev, 481
item_select, 481
item_selected, 481
item_swap, 482
item_text, 482
item_width, 483
nchecked, 483
nitems, 483
remove, 483
set_checked, 483
text, 484
value, 484
Fl_Check_Button, 484
Fl_Check_Button
Fl_Check_Browser, 485
Fl_Choice, 486
draw, 488
Fl_Choice, 488
handle, 488
value, 489, 490
Fl_Clock, 490
Fl_Clock, 492
handle, 493
Fl_Clock_Output, 493
draw, 496
Fl_Clock_Output, 495
hour, 496
minute, 496
second, 496
shadow, 497
value, 497, 498
Fl_Color_Chooser, 499
b, 501
Fl_Color_Chooser, 501
g, 501
handle, 501
hsv, 502
hsv2rgb, 503
hue, 503
mode, 503
r, 504
rgb, 504
rgb2hsv, 504
saturation, 505
value, 505
Fl_Color_Chooser.H, 1287
Fl_Copy_Surface, 505
draw_decorated_window, 507
Fl_Copy_Surface, 507
origin, 507, 508
printable_rect, 508
set_current, 508
translate, 509
Fl_Counter, 509
draw, 511
Fl_Counter, 511
handle, 512
lstep, 512
step, 513
Fl_Cursor
Enumerations.H, 1264
Fl_Damage
Enumerations.H, 1265
Fl_Device.H, 1288
INDEX

Fl_Device_Plugin, 513
   rectangle_capture, 514
Fl_Dial, 515
   angle1, 517
   draw, 517
   Fl_Dial, 516
   handle, 517
Fl_Display_Device, 518
Fl_Double_Window, 518
   ~Fl_Double_Window, 519
   flush, 520
   hide, 520
   resize, 520
   show, 520
Fl_Double_Window.cxx, 1288
Fl_EPS_File_Surface, 522
   ~Fl_EPS_File_Surface, 523
   close, 524
   driver, 524
   Fl_EPS_File_Surface, 523
   origin, 524
   printable_rect, 525
   translate, 525
Fl_End, 521
Fl_Event
   Enumerations.H, 1265
Fl_Event_Dispatch
   Callback function typedefs, 230
Fl_File_Browser, 526
   filetype, 527
   filter, 527
   Fl_File_Browser, 527
   iconsize, 528
   load, 528
Fl_File_Chooser, 528
   ~Fl_File_Chooser, 533
   add_extra, 534
   color, 534
   count, 535
   directory, 535
   filter, 535
   filter_value, 536
   Fl_File_Chooser, 533
   hide, 536
   iconsize, 536
   label, 536, 537
   preview, 537
   rescan, 537
   show, 537
   showHiddenButton, 540
   shown, 537
   textcolor, 538
   textfont, 538
   textsize, 538, 539
   type, 539
   user_data, 539
   value, 539
   visible, 540
Fl_File_ICON, 540
   add, 542
   add_color, 543
   add_vertex, 543
   clear, 544
   draw, 544
   find, 544
   first, 544
   Fl_File_ICON, 542
   label, 545
   labeltype, 545
   load, 545
   load_fti, 546
   load_image, 546
   load_system_icons, 546
   next, 546
   pattern, 547
   size, 547
   type, 547
   value, 547
Fl_File_Input, 548
   down_box, 549, 550
   errorcolor, 550
   Fl_File_Input, 549
   handle, 550
   value, 551
Fl_File_Sort_F
   File names and URI utility functions, 360
Fl_Fill_Dial, 551
   Fl_Fill_Dial, 552
Fl_Fill_Slider, 553
   Fl_Fill_Slider, 553
Fl_Float_Input, 554
   Fl_Float_Input, 554
Fl_Font
   Enumerations.H, 1261
Fl_Fontsize
   Enumerations.H, 1262
Fl_Fill_Bitmap, 555
   bitmap, 555
   draw, 556
   set, 556
Fl_Fill_Pixmap, 556
   draw, 558
   Fl_Fill_Pixmap, 557
   Pixmap, 558
   set, 558
Fl_Fill_Text, 559
   draw, 559
Fl_Free, 560
   draw, 561
   Fl_Free, 561
   handle, 562
Fl_GIF_Image, 563
   Fl_GIF_Image, 563, 564
Fl_Gl_Choice, 564
Fl_Gl_Window, 565
   as_gl_window, 568
can_do, 568, 569
can_do_overlay, 569
context, 569
context_valid, 570
draw, 570
Fl_Gl_Window, 567, 568
flush, 571
hide_overlay, 571
make_current, 571
make_overlay_current, 571
mode, 571, 572
ortho, 573
pixel_h, 573
pixel_w, 573
pixels_per_unit, 573
resize, 574
show, 574
swap_buffers, 575
valid, 575
Fl_Glut_Bitmap_Font, 576
Fl_Glut_StrokeChar, 576
Fl_Glut_StrokeFont, 576
Fl_Glut_StrokeStrip, 577
Fl_Glut_StrokeVertex, 577
Fl_Glut_Window, 577
draw, 579
draw_overlay, 580
Fl_Glut_Window, 578, 579
Fl_Group, 580
~Fl_Group, 583
array, 584
as_group, 584
begin, 585
bounds, 585
child, 586
clear, 586
clip_children, 586
current, 587
draw, 587
draw_child, 587
draw_children, 588
draw_outside_label, 588
end, 588
find, 588
Fl_Group, 583
focus, 588
handle, 589
init_sizes, 589
insert, 590
remove, 590, 591
resizable, 591
resize, 592
sizes, 593
update_child, 593
Fl_Help_Block, 594
Fl_Help_Dialog, 594
Fl_Help_Dialog, 596
h, 596
hide, 596
load, 596
position, 597
resize, 597
show, 597
textsize, 597
value, 598
visible, 598
w, 598
x, 598
y, 598
Fl_Help_Font_Stack, 599
count, 600
Fl_Help_Font_Stack, 599
top, 600
Fl_Help_Font_Style, 600
Fl_Help_Link, 601
Fl_Help_Target, 601
Fl_Help_View, 602
~Fl_Help_View, 605
clear_selection, 606
directory, 606
draw, 606
filename, 606
find, 606
handle, 607
leftline, 607
link, 607
load, 608
resize, 608
scrollbar_size, 609
select_all, 610
size, 610
textcolor, 610
textfont, 610
textsize, 611
title, 611
topline, 611, 612
value, 612
Fl_Hold_Browser, 613
Fl_Hold_Browser, 613
Fl_Hor_Fill_Slider, 614
Fl_Hor_Nice_Slider, 615
Fl_Hor_Slider, 615
Fl_Hor_Value_Slider, 616
Fl_Image, 617
as_shared_image, 620
color_average, 620
copy, 620, 621
count, 621
d, 621
data, 621
desaturate, 621
draw, 621, 622
draw_empty, 622
draw_scaled, 622
fail, 623
INDEX

Fl_Image, 619
  h, 623, 624
inactive, 624
  label, 624
id, 624, 625
RGB_scaling, 625, 626
release, 625
  scale, 626
  scaling_algorithm, 627
uncache, 627
w, 627
Fl_Image.H, 1294
  Fl_RGB_Scaling, 1295
Fl_Image_Reader, 628
Fl_Image_Surface, 628
  ~Fl_Image_Surface, 630
  Fl_Image_Surface, 630
highres_image, 631
image, 631
offscreen, 631
origin, 631, 632
printable_rect, 632
rescale, 632
set_current, 633
translate, 633
Fl_Input, 634
draw, 636
  Fl_Input, 636
handle, 636
Fl_Input_, 637
  ~Fl_Input_, 641
append, 641
copy, 641
copy_cuts, 642
cursor_color, 642
cut, 643
drawtext, 644
  Fl_Input_, 640
handle_mouse, 644
handletext, 644
index, 645
input_type, 645
insert, 646
line_end, 646
line_start, 646
mark, 647
maximum_size, 648
position, 648, 649
readonly, 649
replace, 650
resize, 651
shortcut, 651
size, 652
static_value, 652, 653
tab_nav, 653, 654
textcolor, 654, 655
textfont, 655
textsize, 656
undo, 656
up_down_position, 656
value, 657, 658
word_end, 658
word_start, 659
wrap, 659
Fl_Input_Choice, 660
  add, 662
changed, 663
clear, 663
clear_changed, 663
Fl_Input_Choice, 662
input, 663
menu, 663, 664
menubutton, 664
update_menubutton, 664
value, 665
Fl_Int_Input, 666
  Fl_Int_Input, 666
Fl_JPEG_Image, 667
  Fl_JPEG_Image, 667, 668
Fl_Label, 668
draw, 669
measure, 669
type, 670
Fl_Labeltype
  Enumerations.H, 1268
Fl_Light_Button, 670
draw, 672
  Fl_Light_Button, 671
handle, 672
Fl_Line_Dial, 673
Fl_Mac_App_Menu, 673
  custom_application_menu_items, 674
  print, 674
Fl_Menu_, 675
  add, 678, 680
  clear, 681
clear_submenu, 681
copy, 682
down_box, 682
find_index, 682, 683
find_item, 684
Fl_Menu_, 678
global, 685
insert, 685
item_pathname, 686
menu, 686, 687
menu_end, 687
mode, 688
mvalue, 688
picked, 689
remove, 689
replace, 689
setonly, 690
shortcut, 690
size, 690
test_shortcut, 690

Generated by Doxygen
text, 690, 691
textcolor, 691
textfont, 691
textsize, 691, 692
value, 692
Fl_Menu_Bar, 693
draw, 695
Fl_Menu_Bar, 694
handle, 695
update, 696
Fl_Menu_Button, 696
draw, 698
Fl_Menu_Button, 698
handle, 699
popup, 699
popup_buttons, 698
Fl_Menu_Item, 700
activate, 704
active, 704
activevisible, 705
add, 705
argument, 705
callback, 706, 707
check, 707
checkbox, 707
checked, 707
clear, 708
deactivate, 708
do_callback, 708
draw, 709
find_shortcut, 709
first, 709, 710
hide, 710
image, 710
insert, 710
label, 711
labelcolor, 711
labelfont, 711, 712
labelsize, 712
labeltype, 712
measure, 713
next, 713
popup, 713
pulldown, 714
radio, 714
set, 714
setonly, 714
shortcut, 715
show, 715
size, 715
submenu, 716
test_shortcut, 716
uncheck, 716
value, 716
visible, 717
Fl_Menu_Item.H, 1295
Fl_Menu_Window, 717
Fl_Menu_Window, 718
clear_overlay, 719
Fl_Menu_Window, 718, 719
flush, 719
hide, 719
set_overlay, 720
show, 720
Fl_Multi_Browser, 721
Fl_Multi_Browser, 721
Fl_Multi_Label, 722
label, 723, 724
labela, 724
labelb, 724
typea, 724
typeb, 724
Fl_Multiline_Input, 725
Fl_Multiline_Output, 726
Fl_Multiline_Output, 727
Fl_Native_File_Chooser, 727
~Fl_Native_File_Chooser, 731
count, 731
directory, 731
ermsg, 732
filename, 732
filter, 732
filter_value, 733
Fl_Native_File_Chooser, 731
Option, 730
options, 733
preset_file, 733
show, 734
title, 734
Type, 730
Fl_Native_File_Chooser.H, 1296
Fl_Nice_Slider, 735
Fl_Option
Fl, 377
Fl_Output, 735
Fl_Output, 736
Fl_Overlay_Window, 737
draw_overlay, 739
Fl_Overlay_Window, 738
flush, 739
hide, 739
redraw_overlay, 739
resize, 739
show, 740
Fl_PNG_Image, 756
Fl_PNG_Image, 757
Fl_PNM_Image, 758
Fl_PNM_Image, 758
Fl_Pack, 741
draw, 742
Fl_Pack, 742
horizontal, 743
Fl_Paged_Device, 743
begin_job, 746
begin_page, 746
Generated by Doxygen
b, 806
FL_Rect, 805, 806
r, 806

Fl_Repeate Button, 808
  Fl_Repeate Button, 809
  handle, 809

Fl_Return Button, 810
draw, 811
  Fl_Return Button, 811
  handle, 811

Fl_Roller, 819
draw, 820
  Fl_Roller, 820
  handle, 820

Fl_Round Button, 821
  Fl_Round Button, 822

Fl_Round Clock, 823
  Fl_Round Clock, 823

Fl_SVG File Surface, 879
  ~Fl_SVG File Surface, 882
close, 882
  Fl_SVG File Surface, 880
  origin, 882
  printable_rect, 883
  translate, 883

Fl_SVG Image, 884
  ~Fl_SVG Image, 887
  as_svg_image, 887
  color_average, 887
copy, 887
desaturate, 888
draw, 888
  Fl_SVG Image, 886
  normalize, 888
  proportional, 889
  resize, 888

Fl_Scroll, 824
  bbox, 827
clear, 827
draw, 827
  Fl_Scroll, 827
  handle, 828
  recalc_scrollbars, 828
  resize, 829
  scroll_to, 829
  scrollbar_size, 830
  xposition, 831
  yposition, 831

Fl_Scroll::ScrollInfo, 1244
Fl_Scroll::ScrollInfo::Fl Region LRTB, 807
Fl_Scroll::ScrollInfo::Fl Region XYWH, 807
Fl_Scroll::ScrollInfo::Fl_Scrollbar Data, 835

Fl_Scrollbar, 831
  ~Fl_Scrollbar, 833
draw, 833
  Fl_Scrollbar, 833
  handle, 833
  linesize, 834
  value, 834

Fl_Secret Input, 836
  Fl_Secret Input, 836
  handle, 837

Fl_Select Browser, 838
  Fl_Select Browser, 838

Fl_Shared Image, 839
  ~Fl_Shared Image, 842
  add, 842
  as_shared_image, 842
  color_average, 842
  compare, 843
copy, 844
desaturate, 844
  draw, 844
  find, 845
  Fl_Shared Image, 841
  get, 845, 846
  num_images, 846
  original, 846
  refcount, 847
  release, 847
  reload, 847
  remove_handler, 847
  uncache, 847

Fl_Shared Image.H, 1299
  fl_register_images, 1299

Fl_Shortcut
  fl_types.h, 1306

Fl_Simple Counter, 848

Fl_Simple Terminal, 849
  ansi, 852, 853
  append, 853
clear, 854
current_style_index, 854
draw, 855
enforce_history_lines, 855
enforce_stay_at_bottom, 855
draw, 855
history_lines, 855, 856
  normal_style_index, 856
  printf, 856
  remove_lines, 857
  stay_at_bottom, 857, 858
  style_table, 858, 859
  style_table_size, 859
text, 860
  vprintf, 860

Fl_Single Window, 861
  show, 862

Fl_Slider, 862
  bounds, 864
draw, 864
  Fl_Slider, 864
  handle, 865
  scrollbar, 866
  slider, 866
  slider_size, 866

Fl_Spinner, 867

.Generated by Doxygen
countDisplayedCharacters, 944
countLines, 944
cfileEncodingWarningMessage, 957
findchar_backward, 944
findchar_forward, 945
FlTextBuffer, 941
highlight_text, 945
insert, 945
insert_, 946
insertfile, 946
isWordSeparator, 946
length, 947
line_end, 947
line_start, 947
line_text, 948
loadfile, 948
mPredeleteProcs, 957
mTabDist, 958
next_char, 948
outputfile, 948
prev_char, 949
printf, 949
remove, 950
remove_, 950
replace, 950
rewind_lines, 952
savefile, 952
search_backward, 952
search_forward, 953
secondary_selection_text, 953
selection_text, 953
skipDisplayedCharacters, 954
tab_distance, 954
text, 954
text_range, 956
transcodingWarningAction, 958
vprintf, 956
word_end, 957
word_start, 957
FlTextDisplay, 958
~FlTextDisplay, 967
absolute_top_line_number, 968
buffer, 968, 969
buffer_modified_cb, 969
buffer_predelete_cb, 970
calc_last_char, 970
calc_line_starts, 970
clear_rect, 972
col_to_x, 972
countLines, 972
cursor_color, 973
cursor_style, 973
display_insert, 974
draw, 974
draw_cursor, 974
draw_line_numbers, 975
draw_range, 975
draw_string, 975
draw_text, 976
draw_vline, 976
empty_vlines, 977
extend_range_for_styles, 977
find_line_end, 977
find_wrap_range, 978
find_x, 978
FlTextDisplay, 967
get_absolute_top_line_number, 979
handle_vline, 979
highlight_data, 980
in_selection, 981
insert, 981
insert_position, 982
line_end, 982
line_start, 983
linenumber_align, 983
linenumberbgcolor, 983
linenumber_fbgcolor, 984
linenumber_font, 984
linenumber_format, 984
linenumber_size, 984
linenumber_width, 985
longest_vline, 985
maintain_absolute_top_line_number, 985
maintaining_absolute_top_line_number, 986
measure_deleted_lines, 986
measure_proportional_character, 986
measure_vline, 987
move_down, 987
move_left, 987
move_right, 987
move_up, 988
offset_line_starts, 988
overstrike, 988
position_style, 989
position_to_line, 989
position_to_linecol, 991
position_to_xy, 991
redisplay_range, 992
reset_absolute_top_line_number, 992
resize, 992
rewind_lines, 993
scroll, 993
scroll_, 994
scroll_timer_cb, 994
scrollbar_align, 994
scrollbar_size, 995
scrollbar_width, 996
shortcut, 996
show_cursor, 997
show_insert_position, 997
skip_lines, 997
string_width, 998
textcolor, 998
textfont, 999
textsize, 999
update_h_scrollbar, 1000
Generated by Doxygen
show_widgets, 1121
swap_children, 1121, 1122
tree, 1122
update_prev_next, 1122
userdeicon, 1123
usericon, 1123
visible_r, 1124
w, 1124
Fl_Tree_Item.H, 1301
Fl_Tree_Item_Array, 1124
add, 1126
clear, 1126
deparent, 1126
Fl_Tree_Item_Array, 1125
insert, 1126
manage_item_destroy, 1127
move, 1127
remove, 1127, 1128
reparent, 1128
replace, 1128
Fl_Tree_Item_Array.H, 1302
Fl_Tree_Item_Draw_Mode
Fl_Tree_Prefs.H, 1304
Fl_Tree_Item_Reselect_Mode
Fl_Tree_Prefs.H, 1304
Fl_Tree_Prefs, 1128
closedeicon, 1131
closeicon, 1132
item_draw_mode, 1132
item_labelbgcolor, 1132
marginbottom, 1132, 1133
opendeicon, 1133
openicon, 1133
selectmode, 1133
showcollapse, 1134
showroot, 1134
sortorder, 1134
userdeicon, 1134
Fl_Tree_Prefs.H, 1302
Fl_Tree_Connector, 1303
Fl_Tree_Item_Draw_Mode, 1304
Fl_Tree_Item_Reselect_Mode, 1304
Fl_Tree_Select, 1304
Fl_Tree_Sort, 1305
Fl_Tree_Reason
Fl_Tree.H, 1301
Fl_Tree_Select
Fl_Tree_Prefs.H, 1304
Fl_Tree_Sort
Fl_Tree_Prefs.H, 1305
Fl_Valuator, 1135
bounds, 1138
clamp, 1138
Fl_Valuator, 1137
format, 1138
handle_drag, 1138
handle_release, 1138
increment, 1139
maximum, 1139
minimum, 1139
precision, 1139
range, 1140
round, 1140
set_value, 1140
step, 1140
value, 1141
Fl_Value_Input, 1141
cursor_color, 1143, 1144
draw, 1144
Fl_Value_Input, 1143
handle, 1144
resize, 1145
shortcut, 1145, 1146
soft, 1146
textcolor, 1146
textfont, 1146, 1147
textsize, 1147
Fl_Value_Output, 1147
draw, 1149
Fl_Value_Output, 1148
handle, 1149
soft, 1150
textcolor, 1150
textfont, 1150, 1151
textsize, 1151
Fl_Value_Slider, 1151
draw, 1153
Fl_Value_Slider, 1152
handle, 1153
textcolor, 1154
textfont, 1154
textsize, 1154, 1155
Fl_When
Enumerations.H, 1269
Fl_Widget, 1155
~Fl_Widget, 1163
activate, 1163
active, 1163
active_r, 1164
align, 1164
argument, 1165
as_gl_window, 1165
as_group, 1166
as_window, 1166
box, 1167
callback, 1168, 1169
changed, 1169
clear_active, 1170
clear_changed, 1170
clear_damage, 1170
clear_output, 1171
clear_visible, 1171
clear_visible_focus, 1171
color, 1171, 1172
color2, 1173
contains, 1173
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy_label</td>
<td>1174</td>
</tr>
<tr>
<td>copy_tooltip</td>
<td>1174</td>
</tr>
<tr>
<td>damage</td>
<td>1174, 1175</td>
</tr>
<tr>
<td>damage_resize</td>
<td>1176</td>
</tr>
<tr>
<td>deactivate</td>
<td>1176</td>
</tr>
<tr>
<td>default_callback</td>
<td>1176</td>
</tr>
<tr>
<td>deimage</td>
<td>1177, 1178</td>
</tr>
<tr>
<td>do_callback</td>
<td>1178, 1179</td>
</tr>
<tr>
<td>draw</td>
<td>1179</td>
</tr>
<tr>
<td>draw_box</td>
<td>1180</td>
</tr>
<tr>
<td>draw_focus</td>
<td>1180</td>
</tr>
<tr>
<td>draw_label</td>
<td>1180, 1181</td>
</tr>
<tr>
<td>Fl_Widget</td>
<td>1182</td>
</tr>
<tr>
<td>h</td>
<td>1181</td>
</tr>
<tr>
<td>handle</td>
<td>1181</td>
</tr>
<tr>
<td>hide</td>
<td>1182</td>
</tr>
<tr>
<td>image</td>
<td>1182, 1183</td>
</tr>
<tr>
<td>inside</td>
<td>1184</td>
</tr>
<tr>
<td>is_label_copied</td>
<td>1184</td>
</tr>
<tr>
<td>label</td>
<td>1184, 1185</td>
</tr>
<tr>
<td>label_shortcut</td>
<td>1185</td>
</tr>
<tr>
<td>labelcolor</td>
<td>1186</td>
</tr>
<tr>
<td>labelfont</td>
<td>1186, 1187</td>
</tr>
<tr>
<td>labelsize</td>
<td>1187</td>
</tr>
<tr>
<td>labeltype</td>
<td>1188</td>
</tr>
<tr>
<td>measure_label</td>
<td>1188</td>
</tr>
<tr>
<td>output</td>
<td>1189</td>
</tr>
<tr>
<td>parent</td>
<td>1189, 1190</td>
</tr>
<tr>
<td>position</td>
<td>1190</td>
</tr>
<tr>
<td>redraw</td>
<td>1190</td>
</tr>
<tr>
<td>redraw_label</td>
<td>1190</td>
</tr>
<tr>
<td>resize</td>
<td>1190</td>
</tr>
<tr>
<td>selection_color</td>
<td>1191</td>
</tr>
<tr>
<td>set_active</td>
<td>1192</td>
</tr>
<tr>
<td>set_changed</td>
<td>1192</td>
</tr>
<tr>
<td>set_output</td>
<td>1192</td>
</tr>
<tr>
<td>set_visible</td>
<td>1192</td>
</tr>
<tr>
<td>set_visible_focus</td>
<td>1193</td>
</tr>
<tr>
<td>shortcut_label</td>
<td>1193</td>
</tr>
<tr>
<td>show</td>
<td>1193</td>
</tr>
<tr>
<td>size</td>
<td>1194</td>
</tr>
<tr>
<td>take_focus</td>
<td>1194</td>
</tr>
<tr>
<td>takeevents</td>
<td>1195</td>
</tr>
<tr>
<td>testShortcut</td>
<td>1195</td>
</tr>
<tr>
<td>tooltip</td>
<td>1196</td>
</tr>
<tr>
<td>top_window</td>
<td>1197</td>
</tr>
<tr>
<td>top_window_offset</td>
<td>1197</td>
</tr>
<tr>
<td>type</td>
<td>1197, 1198</td>
</tr>
<tr>
<td>user_data</td>
<td>1198</td>
</tr>
<tr>
<td>visible</td>
<td>1198</td>
</tr>
<tr>
<td>visible_focus</td>
<td>1199</td>
</tr>
<tr>
<td>visible_r</td>
<td>1199</td>
</tr>
<tr>
<td>w,</td>
<td>1200</td>
</tr>
<tr>
<td>when</td>
<td>1200</td>
</tr>
<tr>
<td>window</td>
<td>1201</td>
</tr>
<tr>
<td>x,</td>
<td>1202</td>
</tr>
<tr>
<td>y,</td>
<td>1202</td>
</tr>
<tr>
<td>FL_Widget.H</td>
<td>1309</td>
</tr>
<tr>
<td>FL_RESERVED_TYPE</td>
<td>1310</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1203</td>
</tr>
<tr>
<td>draw</td>
<td>1204</td>
</tr>
<tr>
<td>draw_decorated_window</td>
<td>1204</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1204</td>
</tr>
<tr>
<td>origin</td>
<td>1205</td>
</tr>
<tr>
<td>print_window_part</td>
<td>1206</td>
</tr>
<tr>
<td>printable_rect</td>
<td>1206</td>
</tr>
<tr>
<td>translate</td>
<td>1206</td>
</tr>
<tr>
<td>Fl_Widget_Tracker</td>
<td>1207</td>
</tr>
<tr>
<td>deleted</td>
<td>1208</td>
</tr>
<tr>
<td>exists</td>
<td>1208</td>
</tr>
<tr>
<td>widget</td>
<td>1209</td>
</tr>
<tr>
<td>Fl_Window</td>
<td>1209</td>
</tr>
<tr>
<td>~Fl_Window</td>
<td>1214</td>
</tr>
<tr>
<td>as_window</td>
<td>1215</td>
</tr>
<tr>
<td>border</td>
<td>1215</td>
</tr>
<tr>
<td>clear_border</td>
<td>1215</td>
</tr>
<tr>
<td>clear_modal_states</td>
<td>1215</td>
</tr>
<tr>
<td>current</td>
<td>1216</td>
</tr>
<tr>
<td>current_</td>
<td>1235</td>
</tr>
<tr>
<td>cursor</td>
<td>1216, 1217</td>
</tr>
<tr>
<td>decorated_h</td>
<td>1217</td>
</tr>
<tr>
<td>decorated_w</td>
<td>1218</td>
</tr>
<tr>
<td>default_cursor</td>
<td>1218</td>
</tr>
<tr>
<td>default_icon</td>
<td>1218</td>
</tr>
<tr>
<td>default_icons</td>
<td>1219</td>
</tr>
<tr>
<td>default_xclass</td>
<td>1219, 1220</td>
</tr>
<tr>
<td>draw</td>
<td>1220</td>
</tr>
<tr>
<td>Fl_Window</td>
<td>1213, 1214</td>
</tr>
<tr>
<td>flush</td>
<td>1220</td>
</tr>
<tr>
<td>force_position</td>
<td>1221</td>
</tr>
<tr>
<td>free_icons</td>
<td>1221</td>
</tr>
<tr>
<td>fullscreen</td>
<td>1222</td>
</tr>
<tr>
<td>fullscreen_screens</td>
<td>1222</td>
</tr>
<tr>
<td>handle</td>
<td>1222</td>
</tr>
<tr>
<td>hide</td>
<td>1224</td>
</tr>
<tr>
<td>hotspot</td>
<td>1224</td>
</tr>
<tr>
<td>icon</td>
<td>1224, 1226</td>
</tr>
<tr>
<td>iconize</td>
<td>1226</td>
</tr>
<tr>
<td>iconlabel</td>
<td>1226</td>
</tr>
<tr>
<td>icons</td>
<td>1227</td>
</tr>
<tr>
<td>label</td>
<td>1227</td>
</tr>
<tr>
<td>make_current</td>
<td>1227</td>
</tr>
<tr>
<td>menu_window</td>
<td>1228</td>
</tr>
<tr>
<td>modal</td>
<td>1228</td>
</tr>
<tr>
<td>non_modal</td>
<td>1228</td>
</tr>
<tr>
<td>override</td>
<td>1228</td>
</tr>
<tr>
<td>resize</td>
<td>1228</td>
</tr>
<tr>
<td>set_menu_window</td>
<td>1229</td>
</tr>
<tr>
<td>set_modal</td>
<td>1229</td>
</tr>
<tr>
<td>set_non_modal</td>
<td>1229</td>
</tr>
<tr>
<td>set_tooltip_window</td>
<td>1230</td>
</tr>
<tr>
<td>shape</td>
<td>1230, 1231</td>
</tr>
<tr>
<td>show</td>
<td>1231, 1232</td>
</tr>
<tr>
<td>shown</td>
<td>1232</td>
</tr>
<tr>
<td>size_range</td>
<td>1233</td>
</tr>
</tbody>
</table>
tooltip_window, 1233
wait_for_expose, 1234
xclass, 1234, 1235
FI_Win_H, 1310
FI_Wizard, 1236
draw, 1237
FI_Wizard, 1237
next, 1237
prev, 1238
value, 1238
FI_XBM_Im, 1238
FI_XBM_Im, 1239
FI_XColor, 1239
FI_XPM_Im, 1240
FI_XPM_Im, 1240
/current
access
Unicode and UTF-8 functions, 324
draw
Drawing functions, 283
next
Drawing functions, 287
value
Drawing functions, 288, 289
prev
Drawing functions, 289
value
Drawing functions, 288, 289
next
Drawing functions, 288, 289
prev
Drawing functions, 288, 289
value
Drawing functions, 288, 289
fl_clip_box
Drawing functions, 282
fl_clip_region
Drawing functions, 287
fl_color
Color & Font functions, 268, 269
fl_copy_offscreen
Drawing functions, 270
fl_create_offscreen
Drawing functions, 270
fl_decode_uri
File names and URI utility functions, 360
fl_decode_uri
File names and URI utility functions, 360
fl_down
Enumerations.H, 1271
fl_dir_chooser
Common Dialogs classes and functions, 350
fl_draw
Drawing functions, 291, 292
fl_draw.H, 1289
fl_draw_box
Drawing functions, 292
fl_draw_image
Drawing functions, 293
fl_draw_image_mono
Drawing functions, 294
fl_draw_pixmap
Drawing functions, 295
fl_color
Color & Font functions, 268, 269
fl_color_average
Color & Font functions, 269
fl_color_cube
Common Dialogs classes and functions, 348, 349
fl_color_cube
Common Dialogs classes and functions, 348, 349
fl_contrast
Color & Font functions, 270
fl_decode_uri
File names and URI utility functions, 360
fl_dir_chooser
Common Dialogs classes and functions, 350
fl_draw_pixmap
Drawing functions, 295
Drawing functions, 295
fl_draw_symbol
Drawing functions, 296
fl_eventnames
Events handling functions, 255
fl_expand_text
Drawing functions, 296
fl_filechooser
Common Dialogs classes and functions, 351
fl_filechooser_callback
Common Dialogs classes and functions, 352
fl_filechooser_ok_label
Common Dialogs classes and functions, 352
fl_filename_absolute
File names and URI utility functions, 360
fl_filename_expand
File names and URI utility functions, 361
fl_filename_ext
File names and URI utility functions, 361
fl_filename_free_list
File names and URI utility functions, 362
fl_filename_isdir
File names and URI utility functions, 362
fl_filename_list
File names and URI utility functions, 362
fl_filename_match
File names and URI utility functions, 363
fl_filename_setext
File names and URI utility functions, 363
fl_font
Color & Font functions, 270
fl_fontnames
Events handling functions, 255
fl_fopen
Unicode and UTF-8 functions, 325
fl_frame
Drawing functions, 296
Enumerations.H, 1271
fl_frame2
Drawing functions, 297
fl_gap
Drawing functions, 297
fl_getcwd
Unicode and UTF-8 functions, 326
fl_getenv
Unicode and UTF-8 functions, 326
fl_gray_ramp
Enumerations.H, 1272
fl_height
Color & Font functions, 270, 271
fl_input
Common Dialogs classes and functions, 352
fl_internal_boxtype
Fl_boxtype.cxx, 1286
fl intptr_t
platform_types.h, 1319
fl_latin1_to_local
Color & Font functions, 271
fl_lighter
Enumerations.H, 1272
fl_line_style
Drawing functions, 297
fl_local_to_latin1
Color & Font functions, 271
fl_local_to_mac_roman
Color & Font functions, 272
fl_make_path
Unicode and UTF-8 functions, 327
fl_make_path_for_file
Unicode and UTF-8 functions, 327
fl_measure
Drawing functions, 298
fl_measure_pixmap
Drawing functions, 299
fl_message
Common Dialogs classes and functions, 353
fl_message_hotspot
Common Dialogs classes and functions, 353, 354
fl_message_icon
Common Dialogs classes and functions, 354
fl_message_position
Common Dialogs classes and functions, 354, 355
fl_ask.H, 1284
fl_message_title
Common Dialogs classes and functions, 356
fl_message_title_default
Common Dialogs classes and functions, 356
fl_mkdir
Unicode and UTF-8 functions, 327
fl_mult_matrix
Drawing functions, 299
fl_nonsspacing
Unicode and UTF-8 functions, 328
fl_not_clipped
Drawing functions, 300
fl_numericsort
numsort.c, 1317
fl_old_shortcut
Drawing functions, 300
fl_open
Unicode and UTF-8 functions, 328
fl_open_callback
Mac OS X-specific symbols, 342
fl_open_display
Fl.cxx, 1278
fl_open_ext
Unicode and UTF-8 functions, 329

fl_open_uri
  File names and URI utility functions, 365
fl_password
  Common Dialogs classes and functions, 357
fl_pie
  Drawing functions, 301
fl_polygon
  Drawing functions, 302
fl_pop_clip
  Drawing functions, 303
fl_push_clip
  Drawing functions, 303
fl_push_matrix
  Drawing functions, 303
fl_putenv
  Unicode and UTF-8 functions, 329
fl_read_image
  Drawing functions, 303
fl_rect
  Drawing functions, 304
fl_rect.cxx, 1298
fl_rectbound
  fl_boxtype.cxx, 1286
fl_rectf
  Drawing functions, 304
fl_register_images
  Fl_Shared_Image.H, 1299
fl_rename
  Unicode and UTF-8 functions, 330
fl_resize_offscreen
  Drawing functions, 305
fl_reset_spot
  Drawing functions, 305
fl_rgb_color
  Enumerations.H, 1272
fl_rmdir
  Unicode and UTF-8 functions, 330
fl_rotate
  Drawing functions, 305
fl_scale
  Drawing functions, 306
fl_scroll
  Drawing functions, 306
fl_set_spot
  Drawing functions, 307
fl_set_status
  Drawing functions, 307
flShortcut_label
  Drawing functions, 307, 309
fl_show_colormap
  Color & Font functions, 273
fl_show_colormap.H, 1299
fl_size
  Color & Font functions, 273
fl_stat
  Unicode and UTF-8 functions, 331
fl_system
  Unicode and UTF-8 functions, 331

fl_text_extents
  Color & Font functions, 274
fl_transform_dx
  Drawing functions, 309
fl_transform_dy
  Drawing functions, 309
fl_transform_x
  Drawing functions, 311
fl_transform_y
  Drawing functions, 311
fl_transformed_vertex
  Drawing functions, 311
fl_translate
  Drawing functions, 311
fl_types.h, 1305
  Fl_CString, 1305
  Fl_Shortcut, 1306
  Fl_String, 1306
fl_ucs_to_Utf16
  Unicode and UTF-8 functions, 332
fl_uintptr_t
  platform_types.h, 1319
fl_unlink
  Unicode and UTF-8 functions, 332
fl_utf8.H, 1307
fl_utf8back
  Unicode and UTF-8 functions, 333
fl_utf8bytes
  Unicode and UTF-8 functions, 333
fl_utf8decode
  Unicode and UTF-8 functions, 333
fl_utf8encode
  Unicode and UTF-8 functions, 334
fl_utf8from_mb
  Unicode and UTF-8 functions, 334
fl_utf8froma
  Unicode and UTF-8 functions, 334
fl_utf8fromwc
  Unicode and UTF-8 functions, 334
fl_utf8fwd
  Unicode and UTF-8 functions, 335
fl_utf8fromwc
  Unicode and UTF-8 functions, 335
fl_utf8len
  Unicode and UTF-8 functions, 335
fl_utf8len1
  Unicode and UTF-8 functions, 336
fl_utf8locale
  Unicode and UTF-8 functions, 336
fl_utf8test
  Unicode and UTF-8 functions, 337
fl_utf8to_mb
  Unicode and UTF-8 functions, 337
fl_utf8toUtf16
  Unicode and UTF-8 functions, 338
fl_utf8toa
  Unicode and UTF-8 functions, 337
fl_utf8towc
  Unicode and UTF-8 functions, 338
fl_utf_strcasecmp
  Unicode and UTF-8 functions, 339
fl_utf_strncasecmp
  Unicode and UTF-8 functions, 339
fl_utf_tolower
  Unicode and UTF-8 functions, 340
fl_utf_toupper
  Unicode and UTF-8 functions, 340
fl_vertex
  Drawing functions, 312
fl_vertex.cxx, 1309
fl_wcwidth
  Unicode and UTF-8 functions, 340
fl_wcwidth_
  Unicode and UTF-8 functions, 341
fl_width
  Color & Font functions, 274, 275
flush
  Fl, 386
  Fl_Double_Window, 520
  Fl_Gl_Window, 571
  Fl_Menu_Window, 719
  Fl_Overlay_Window, 739
  Fl_Preferences, 782
  Fl_Window, 1220
focus
  Events handling functions, 251
  Fl_Group, 588
font
  Fl_Tooltip, 1040
force_position
  Fl_Window, 1221
foreground
  Fl, 387
format
  Fl_Spinner, 869
  Fl_Valuator, 1138
format_char
  Fl_Browser, 420, 421
free_color
  Color & Font functions, 275
free_icons
  Fl_Window, 1221
free_position
  Fl_Window, 1221
full_height
  Fl_Browser, 421
  Fl_Browser_, 446
full_width
  Fl_Browser_, 446
fullscreen
  Fl_Window, 1222
fullscreen_screens
  Fl_Window, 1222
g
  Fl_Color_Chooser, 501
get
  Fl_Preferences, 782–785
  Fl_Shared_Image, 845, 846
get_absolute_top_line_number
  Fl_Text_Display, 979
get_awake_handler_
  Fl, 387
get_boxtype
  Fl, 387
get_color
  Color & Font functions, 275
get_font
  Color & Font functions, 276
get_font_name
  Color & Font functions, 276
get_font_sizes
  Color & Font functions, 276
get_key
  Events handling functions, 252
get_mouse
  Events handling functions, 252
get_selected_items
  Fl_Tree, 1066
get_selection
  Fl_Table, 915
get_system_colors
  Fl, 387
getUserdataPath
  Fl_Preferences, 786
gl.h, 1311
gl_color, 1312
gl_draw, 1312–1314
gl_font, 1314
gl_rect, 1314
gl_rectf, 1314
gl_texture_pile_height, 1315
gl_color
  gl.h, 1312
gl_draw
  gl.h, 1312–1314
gl_font
  gl.h, 1314
gl_rect
  gl.h, 1314
gl_rectf
  gl.h, 1314
gl_texture_pile_height
  gl.h, 1315
gl_visual
  Fl, 387
global
  Fl_Menu_, 685
global_key_bindings
  Fl_Text_Editor, 1019
grab
  Windows handling functions, 236
group
  Fl_Preferences, 787
groupExists
  Fl_Preferences, 787
groups
  Fl_Preferences, 787

h
  Fl_Help_Dialog, 596
  Fl_Image, 623, 624
  Fl_Widget, 1181
  Screen functions, 261
handle
  Events handling functions, 252
  Fl_Adjuster, 402
  Fl_Box, 410
  Fl_Browser_, 446
  Fl_Button, 464
  Fl_Check_Browser, 479
  Fl_Choice, 488
  Fl_Clock, 493
  Fl_Color_Chooser, 501
  Fl_Counter, 512
  Fl_Dial, 517
  Fl_File_Input, 550
  Fl_Free, 562
  Fl_Group, 589
  Fl_Help_View, 607
  Fl_Input, 636
  Fl_Light_Button, 672
  Fl_Menu_Bar, 695
  Fl_Menu_Button, 699
  Fl_Positioner, 761
  Fl_Repeat_Button, 809
  Fl_Return_Button, 811
  Fl_Roller, 820
  Fl_Scroll, 828
  Fl_Scrollbar, 833
  Fl_Secret_Input, 837
  Fl_Slider, 865
  Fl_Spinner, 869
  Fl_Spinner::Fl_Spinner_Input, 875
  Fl_Tabs, 933
  Fl_Tile, 1027
  Fl_Timer, 1034
  Fl_Tree, 1067
  Fl_Value_Input, 1144
  Fl_Value_Output, 1149
  Fl_Value_Slider, 1153
  Fl_Widget, 1181
  Fl_Window, 1222
handle_
  Events handling functions, 253
handle_drag
  Fl_Valuator, 1138
handle_mouse
  Fl_Input_, 644
handle_release
  Fl_Valuator, 1138
handle_vline
  Fl_Text_Display, 979
handle_text
  Fl_Input_, 644
has_scrollbar
  Fl_Browser_, 447
help
  Fl, 400
hide
  Fl_Browser, 422
  Fl_Double_Window, 520
  Fl_File_Chooser, 536
  Fl_Help_Dialog, 596
  Fl_Menu_Item, 710
  Fl_Menu_Window, 719
  Fl_Overlay_Window, 739
  Fl_Widget, 1182
  Fl_Window, 1224
hide_overlay
  Fl_Gl_Window, 571
hide_widgets
  Fl_Tree_Item, 1111
hidedelay
  Fl_Tooltip, 1040, 1041
highlight_data
  Fl_Text_Display, 980
highlight_text
  Fl_Text_Buffer, 945
highres_image
  Fl_Image_Surface, 631
history_lines
  Fl_Simple_Terminal, 855, 856
horizontal
  Fl_Pack, 743
hotspot
  Fl_Window, 1224
hour
  Fl_Clock_Output, 496
hoverdelay
  Fl_Tooltip, 1041
hposition
  Fl_Browser_, 447
  Fl_Tree, 1067
hscrollbar
  Fl_Browser_, 460
hsv
  Fl_Color_Chooser, 502
hsv2rgb
  Fl_Color_Chooser, 503
hue
  Fl_Color_Chooser, 503
icon
  Fl_Browser, 422, 423
  Fl_Window, 1224, 1226
iconize
  Fl_Window, 1226
iconlabel
  Fl_Window, 1226
icons
  Fl_Window, 1227
iconszie
  Fl_File_Browser, 528
INDEX

FI_File_Chooser, 536
ID
FI_Preferences, 775
idle
FI, 400
image
FI_Image_Surface, 631
FI_Menu_Item, 710
FI_Widget, 1182, 1183
in_selection
FI_Text_Display, 981
inactive
FI_Image, 624
includes
FI_Text_Selection, 1022
incr_height
FI_Browser, 423
FI_Browser_, 448
increment
FI_Valuator, 1139
index
FI_Input_, 645
init_sizes
FI_Group, 589
FI_Table, 916
input
FI_Input_CHOICE, 663
input_type
FI_Input_, 645
insert
FI_Browser, 423, 424
FI_Chart, 472
FI_Group, 590
FI_Input_, 646
FI_Menu_, 685
FI_Menu_Item, 710
FI_Sys_Menu_Bar, 897
FI_Table, 916
FI_Text_Buffer, 945
FI_Text_Display, 981
FI_Tree, 1068
FI_Tree_Item, 1111
FI_Tree_Item_Array, 1126
insert_
FI_Text_Buffer, 946
insert_above
FI_Tree, 1069
FI_Tree_Item, 1111
insert_mode
FI_Text_Editor, 1011
insert_position
FI_Text_Display, 982
insertfile
FI_Text_Buffer, 946
inserting
FI_Browser_, 448
insertion_point_location
FI, 388
inside
FI_Widget, 1184
is_close
FI_Tree, 1069, 1070
is_hscroll_visible
FI_Tree, 1070
is_interactive_resize
FI_Table, 916
is_label_copied
FI_Widget, 1184
is_open
FI_Tree, 1070, 1071
is_scheme
FI, 388
is_scrollbar
FI_Tree, 1071
is_selected
FI_Table, 916
FI_Tree, 1072
is_vscroll_visible
FI_Tree, 1073
is_word_separator
FI_Text_Buffer, 946
item_at
FI_Browser, 424
FI_Browser_, 448
FI_Check_Browser, 479
item_clicked
FI_Tree, 1073
item_draw
FI_Browser, 425
item_draw_mode
FI_Tree, 1074
FI_TreePrefs, 1132
item_first
FI_Browser, 425
FI_Browser_, 450
FI_Check_Browser, 480
item_height
FI_Browser, 426
FI_Browser_, 450
FI_Check_Browser, 480
item_labelbgcolor
FI_Tree, 1075
FI_TreePrefs, 1132
item_labelbgcolor
FI_Tree, 1075
item_labelbold
FI_Tree, 1075
item_labelfont
FI_Tree, 1075
item_labelsize
FI_Tree, 1075
item_last
FI_Browser, 426
FI_Browser_, 450
item_next
FI_Browser, 427
FI_Browser_, 451
FI_Check_Browser, 480

Generated by Doxygen
item_pathname
  FL_Menu, 686
  FL_Tree, 1076
item_prev
  FL_Browser, 427
  FL_Browser_, 451
  FL_Check_Browser, 481
item_quick_height
  FL_Browser, 451
item_reselect_mode
  FL_Tree, 1076
item_select
  FL_Browser, 428
  FL_Browser_, 452
  FL_Check_Browser, 481
item_selected
  FL_Browser, 428
  FL_Browser_, 452
  FL_Check_Browser, 481
item_swap
  FL_Browser, 428
  FL_Browser_, 452
  FL_Check_Browser, 482
item_text
  FL_Browser, 429
  FL_Browser_, 453
  FL_Check_Browser, 482
item_width
  FL_Browser, 429
  FL_Browser_, 453
  FL_Check_Browser, 483

Key_Func
  FL_Text_Editor, 1009
keyboard_screen_scaling
  Screen functions, 261
kf_backspace
  FL_Text_Editor, 1011
kf_c_s_move
  FL_Text_Editor, 1011
kf_copy
  FL_Text_Editor, 1011
ekf_ctrl_move
  FL_Text_Editor, 1012
ekf_cut
  FL_Text_Editor, 1012
ekf_default
  FL_Text_Editor, 1012
ekf_delete
  FL_Text_Editor, 1012
ekf_down
  FL_Text_Editor, 1012
kf_end
  FL_Text_Editor, 1013
kf_home
  FL_Text_Editor, 1013
kf_ignore
  FL_Text_Editor, 1013
kf_insert
  FL_Text_Editor, 1014
kf_left
  FL_Text_Editor, 1014
kf_m_s_move
  FL_Text_Editor, 1014
kf_meta_move
  FL_Text_Editor, 1014
kf_move
  FL_Text_Editor, 1015
kf_page_down
  FL_Text_Editor, 1015
kf_page_up
  FL_Text_Editor, 1015
kf_paste
  FL_Text_Editor, 1015
kf_right
  FL_Text_Editor, 1015
kf_select_all
  FL_Text_Editor, 1016
kf_shift_move
  FL_Text_Editor, 1016
kf_undo
  FL_Text_Editor, 1016
kf_up
  FL_Text_Editor, 1016

label
  FL_Bitmap, 406
  FL_File_Chooser, 536, 537
  FL_File_Icon, 545
  FL_Image, 624
  FL_Menu_Item, 711
  FL_Multi_Label, 723, 724
  FL_Pixmap, 752
  FL_RGB_Image, 816, 817
  FL_Tree_Item, 1112
  FL_Widget, 1184, 1185
  FL_Window, 1227
label_h
  FL_Tree_Item, 1112
label_shortcut
  FL_Widget, 1185
label_w
  FL_Widget, 1185
label_x
  FL_Tree_Item, 1112
label_y
  FL_Tree_Item, 1112
labela
  FL_Tree_Item, 1113
labelb
  FL_Multi_Label, 724
labelbgcolor
  FL_Tree_Item, 1113
labelcolor
  FL_Menu_Item, 711
  FL_Widget, 1186
labelfont
  FL_Menu_Item, 711, 712
  FL_Widget, 1186, 1187
labelmarginleft
  FL_Tree, 1077
labels
  FL_Menu_Item, 712
  FL_Widget, 1187
labeltype
  FL_File_Icon, 545
  FL_Menu_Item, 712
  FL_Widget, 1188
last
  FL_Tree, 1077
last_selected_item
  FL_Tree, 1077
last_visible
  FL_Tree, 1078
last_visible_item
  FL_Tree, 1078
ld
  FL_Image, 624, 625
leftedge
  FL_Browser_, 454
leftline
  FL_Help_View, 607
length
  FL_Text_Buffer, 947
  FL_Text_Selection, 1022
line_end
  FL_Input_, 646
  FL_Text_Buffer, 947
  FL_Text_Display, 982
line_start
  FL_Input_, 646
  FL_Text_Buffer, 947
  FL_Text_Display, 983
line_text
  FL_Text_Buffer, 948
lineno
  FL_Browser, 430
linenumber_align
  FL_Text_Display, 983
linenumberbgcolor
  FL_Text_Display, 983
linenumberbgcolor
  FL_Text_Display, 984
linenumberbgcolor
  FL_Text_Display, 984
linenumberfont
  FL_Text_Display, 984
linenumberformat
  FL_Text_Display, 984
linenumber_size
  FL_Text_Display, 984
linenumber_width
  FL_Text_Display, 985
lineposition
  FL_Browser, 430
linesize
  FL_Scrollbar, 834
link
  FL_Help_View, 607
load
  FL_Browser, 431
  FL_File_Browser, 528
  FL_File_Icon, 545
  FL_Help_Dialog, 596
  FL_Help_View, 608
  FL_Plugin_Manager, 755
  FL_Tree, 1078
load_fti
  FL_File_Icon, 546
load_image
  FL_File_Icon, 546
load_system_icons
  FL_File_Icon, 546
loadfile
  FL_Text_Buffer, 948
lock
  Multithreading support functions, 314
longest_vline
  FL_Text_Display, 985
lstep
  FL.Counter, 512
mPredeleteProcs
  FL_Text_Buffer, 957
mTabDist
  FL_Text_Buffer, 958
Mac OS X-specific symbols, 342
  fl_mac_os_version, 343
  fl_mac_set_about, 342
  fl_open_callback, 342
mac.H, 1315
maintain_absolute_top_line_number
  FL_Text_Display, 985
maintaining_absolute_top_line_number
  FL_Text_Display, 986
make_current
  FL_Gl_Window, 571
  FL_Window, 1227
make_overlay_current
  FL_Gl_Window, 571
make_visible
  FL_Browser, 431
manage_item_destroy
  FL_Tree_Item_Array, 1127
margin_height
  FL_Tooltip, 1041
margin_width
  FL_Tooltip, 1041, 1042
marginbottom
  FL_Tree_Prefs, 1132, 1133
margins
  FL_Paged_Device, 747
  FL_PostScript_File_Device, 768
  FL_Printer, 796
mark
INDEX 1353

Fl_Input_  647
max_size
    Fl_RGB_Image, 817
maximum
    Fl_Progress, 801
    Fl_Spinner, 870
    Fl_Valuator, 1139
maximum_size
    Fl_Input_, 648
maxsize
    Fl_Chart, 473
measure
    Fl_Label, 669
    Fl_Menu_Item, 713
measure_deleted_lines
    Fl_Text_Display, 986
measure_label
    Fl_Widget, 1188
measure_proportional_character
    Fl_Text_Display, 986
measure_vline
    Fl_Text_Display, 987
menu
    Fl_Input_Choice, 663, 664
    Fl_Menu_, 686, 687
    Fl_Sys_Menu_Bar, 898
menu_end
    Fl_Menu_, 687
menu_linespacing
    Fl, 389
menu_window
    Fl_Window, 1228
menubutton
    Fl_Input_Choice, 664
middleline
    Fl_Browser, 432
minimum
    Fl_Progress, 801
    Fl_Spinner, 870, 871
    Fl_Valuator, 1139
minute
    Fl_Clock_Output, 496
modal
    Fl_Window, 1228
    Windows handling functions, 236
mode
    Fl_Color_Chooser, 503
    Fl_GL_Window, 571, 572
    Fl_Menu_, 688
    Fl_Sys_Menu_Bar, 898
move
    Fl_Browser, 432
    Fl_Tree_Item, 1113, 1114
    Fl_Tree_Item_Array, 1127
move_above
    Fl_Tree_Item, 1114
move_below
    Fl_Tree_Item, 1115
move_cursor
    Fl_Table, 917
move_down
    Fl_Text_Display, 987
move_into
    Fl_Tree_Item, 1115
move_left
    Fl_Text_Display, 987
move_right
    Fl_Text_Display, 987
move_up
    Fl_Text_Display, 988
Multithreading support functions, 313
    awake, 313
    lock, 314
    thread_message, 314
    unlock, 314
mvalue
    Fl_Menu_, 688
NONE
    Fl_Preferences, 792
Name
    Fl_Preferences::Name, 1242
nchecked
    Fl_Check_Browser, 483
new_list
    Fl_Browser_, 454
newUUID
    Fl_Preferences, 788
next
    Fl_File_Icon, 546
    Fl_Menu_Item, 713
    Fl_Tree, 1079
    Fl_Tree_Item, 1115
    Fl_Wizard, 1237
next_char
    Fl_Text_Buffer, 948
next_displayed
    Fl_Tree_Item, 1116
next_item
    Fl_Tree, 1079
next_selected_item
    Fl_Tree, 1080
next_sibling
    Fl_Tree_Item, 1116
next_visible
    Fl_Tree_Item, 1116
next_visible_item
    Fl_Tree, 1081
next_window
    Windows handling functions, 237
nitems
    Fl_Check_Browser, 483
non_modal
    Fl_Window, 1228
normal_style_index
    Fl_Simple.Terminal, 856
normalize

Generated by Doxygen
set_draw_cb
  Fl_Cairo_Window, 468
set_font
  Color & Font functions, 277
set_fonts
  Color & Font functions, 277
set_idle
  Fl, 396
set_item_focus
  Fl_Tree, 1091
set_labeltype
  Fl, 397
set_menu_window
  Fl_Window, 1229
set_modal
  Fl_Window, 1229
set_non_modal
  Fl_Window, 1229
set_output
  Fl_Widget, 1192
set_overlay
  Fl_Menu_Window, 720
set_selection
  Fl_Table, 920
set_tooltip_window
  Fl_Widget, 1192
set_visible
  Fl_Widget, 1192
set_visible_focus
  Fl_Widget, 1193
setonly
  Fl_Menu_, 690
  Fl_Menu_Item, 714
  Fl_Sys_Menu_Bar, 899
shadow
  Fl_Clock_Output, 497
shape
  Fl_Window, 1230, 1231
shortcut
  Fl_Button, 465
  Fl_Field_, 651
  Fl_Menu_, 690
  Fl_Menu_Item, 715
  Fl_Text_Display, 996
  Fl_Value_Input, 1145, 1146
shortcut_label
  Fl_Widget, 1193
show
  Fl_Browser, 434
  Fl_Double_Window, 520
  Fl_File_Chooser, 537
  Fl_GI_Window, 574
  Fl_Help_Dialog, 597
  Fl_Menu_Item, 715
  Fl_Menu_Window, 720
  Fl_Native_File_Chooser, 734
  Fl_Overlay_Window, 740
  Fl_Single_window, 862
  Fl_Widget, 1193
  Fl_Window, 1231, 1232
show_cursor
  Fl_Text_Display, 997
show_insert_position
  Fl_Text_Display, 997
show_item
  Fl_Tree, 1091, 1092
show_item_bottom
  Fl_Tree, 1092
show_item_middle
  Fl_Tree, 1092
show_item_top
  Fl_Tree, 1093
show_self
  Fl_Tree, 1093
  Fl_Tree_Item, 1121
show_widgets
  Fl_Tree_Item, 1121
showHiddenButton
  Fl_File_Chooser, 540
showcollapse
  Fl_Tree, 1093
  Fl_Tree_Prefs, 1134
shown
  Fl_File_Chooser, 537
  Fl_Window, 1232
showroot
  Fl_Tree, 1094
  Fl_Tree_Prefs, 1134
size
  Fl_Browser, 435
  Fl_Field_, 652
  Fl_Menu_, 690
  Fl_Menu_Item, 715
  Fl_Preferences, 791
  Fl_Tooltip, 1042
  Fl_Widget, 1194
size_range
  Fl_Window, 1233
sizes
  Fl_Group, 593
skip_displayed_characters
  Fl_Text_Buffer, 954
skip_lines
  Fl_Text_Display, 997
slider
  Fl_Slider, 866
slider_size
  Fl_Slider, 866
soft
  Fl_Adjuster, 403
  Fl_Value_Input, 1146
INDEX

Fl_Value_Output, 1150
sort
Fl_Browser_, 458
sortorder
Fl_Tree, 1094
Fl_Tree_Prefs, 1134
start
Fl_Text_Selection, 1024
start_job
Fl_Paged_Device, 748
Fl_PostScript_File_Device, 770
start_page
Fl_Paged_Device, 748
static_value
Fl_Input_, 652, 653
stay_at_bottom
Fl_Simple_Terminal, 857, 858
step
Fl_Counter, 513
Fl_Spinner, 871, 872
Fl_Valuator, 1140
str
FL_CHART_ENTRY, 475
string_width
Fl_Text_Display, 998
style_table
Fl_Simple_Terminal, 858, 859
style_table_size
Fl_Simple_Terminal, 859
submenu
Fl_Menu_Item, 716
surface
Fl_Surface_Device, 879
suspended
Fl_Timer, 1035
swap
Fl_Browser, 435
swap_buffers
Fl_Gl_Window, 575
swap_children
Fl_Tree_Item, 1121, 1122
swapping
Fl_Browser_, 459
tab_align
Fl_Tabs, 934
tab_cell_nav
Fl_Table, 920, 921
tab_distance
Fl_Text_Buffer, 954
tab_nav
Fl_Input_, 653, 654
Fl_Text_Editor, 1017, 1019
table_box
Fl_Table, 921
table_resized
Fl_Table, 921
table_scrolled
Fl_Table, 921
TableContext
Fl_Table, 908
take_focus
Fl_Widget, 1194
takesevents
Fl_Widget, 1195
test_shortcut
Events handling functions, 254
Fl_Menu_, 690
Fl_Menu_Item, 716
Fl_Widget, 1195
text
Fl_Browser, 436
Fl_Check_Browser, 484
Fl_Menu_, 690, 691
Fl_Simple_Terminal, 860
Fl_Text_Buffer, 954
text_range
Fl_Text_Buffer, 956
textcolor
Fl_Browser, 436
Fl_Menu_, 690, 691
Fl_Simple_Terminal, 860
Fl_Text_Buffer, 954

Generated by Doxygen
update_v_scrollbar
   Fl_Text_Display, 1000
use_high_res_GL
   Fl, 397
user_data
   Fl_File_Chooser, 539
   Fl_Widget, 1198
userdeicon
   Fl_Tree_Item, 1123
   Fl_Tree_Prefs, 1134
usericon
   Fl_Tree, 1094
   Fl_Tree_Item, 1123
usericonmarginleft
   Fl_Tree, 1095
val
   FL_CHART_ENTRY, 475
valid
   Fl_Gl_Window, 575
value
   Fl_Browser, 437, 438
   Fl_Button, 466
   Fl_Check_Browser, 484
   Fl_Choice, 489, 490
   Fl_Clock_Output, 497, 498
   Fl_Color_Chooser, 505
   Fl_File_Chooser, 539
   Fl_File_Icon, 547
   Fl_File_Input, 551
   Fl_Help_Dialog, 598
   Fl_Help_View, 612
   Fl_Input_, 657, 658
   Fl_Input_Choice, 665
   Fl_Menu_, 692
   Fl_Menu_Item, 716
   Fl_Positioner, 762
   Fl_Progress, 801, 802
   Fl_Scrollbar, 834
   Fl_Spinner, 873, 874
   Fl_Tabs, 934
   Fl_Valuator, 1141
   Fl_Wizard, 1238
version
   Fl, 398
visible
   Fl_Browser, 438
   Fl_File_Chooser, 540
   Fl_Help_Dialog, 598
   Fl_Menu_Item, 717
   Fl_Widget, 1198
visible_cells
   Fl_Table, 922
visible_focus
   Fl, 398
   Fl_Widget, 1199
visible_r
   Fl_Tree_Item, 1124
   Fl_Widget, 1199
visual
   Fl, 398
vline_length
   Fl_Text_Display, 1000
vposition
   Fl_Tree, 1095
vprintf
   Fl_Simple_Terminal, 860
   Fl_Text_Buffer, 956
w
   Fl_Help_Dialog, 598
   Fl_Image, 627
   Fl_Tree_Item, 1124
   Fl_Widget, 1200
   Screen functions, 266
wait
   Fl, 399
wait_for_expose
   Fl_Window, 1234
warning
   Common Dialogs classes and functions, 358
watch_widget_pointer
   Safe widget deletion support functions, 317
when
   Fl_Table, 922
   Fl_Widget, 1200
which
   Fl_Tabs, 935
widget
   Fl_Widget_Tracker, 1209
widgetmarginleft
   Fl_Tree, 1096
window
   Fl_Widget, 1201
window_menu_style
   Fl_Sys_Menu_Bar, 899
window_menu_style_enum
   Fl_Sys_Menu_Bar, 892
Windows handling functions, 235
   atclose, 237
default_atclose, 235
   first_window, 235, 236
   grab, 236
   modal, 236
   next_window, 237
   set_atclose, 237
word_end
   Fl_Input_, 658
   Fl_Text_Buffer, 957
   Fl_Text_Display, 1001
word_start
   Fl_Input_, 659
   Fl_Text_Buffer, 957
   Fl_Text_Display, 1001
wrap
   Fl_Input_, 659
   Fl_Spinner, 874
wrap_mode

Generated by Doxygen
Fl_Text_Display, 1002
wrapUsesCharacter
Fl_Text_Display, 1002
wrap_width
Fl_Tooltip, 1042, 1043
wrapped_column
Fl_Text_Display, 1003
wrapped_line_counter
Fl_Text_Display, 1003
wrapped_row
Fl_Text_Display, 1004
x
Fl_Help_Dialog, 598
Fl_Widget, 1202
Screen functions, 266
x_to_col
Fl_Text_Display, 1005
xbounds
Fl_Positioner, 762
xclass
Fl_Window, 1234, 1235
xposition
Fl_Scroll, 831
xstep
Fl_Positioner, 762
xvalue
Fl_Positioner, 762, 763
xy_to_position
Fl_Text_Display, 1005
xy_to_rowcol
Fl_Text_Display, 1005
y
Fl_Help_Dialog, 598
Fl_Widget, 1202
Screen functions, 266
ybounds
Fl_Positioner, 763
yposition
Fl_Scroll, 831
ystep
Fl_Positioner, 763
yvalue
Fl_Positioner, 763