# 1 FLTK Programming Manual

## 2 Preface

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

## 3 Introduction to FLTK

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 with CMake ......... 7
3.6 Building and Installing FLTK Under UNIX and macOS with make .......... 7
3.7 Building FLTK Under Microsoft Windows ......... 10
   3.7.1 GNU toolsets (Cygwin or MinGW) hosted on Windows .......... 10
   3.7.2 Using the Visual C++ DLL Library ............ 11
3.8 Internet Resources .................. 11
3.9 Reporting Bugs ...................... 11

## 4 FLTK Basics

4.1 Writing Your First FLTK Program .......... 13
   4.1.1 Creating the Widgets ............. 14
   4.1.2 Creating Widget Hierarchies .......... 14
   4.1.3 Get/Set Methods .................. 14
   4.1.4 Redrawing After Changing Attributes .......... 15
   4.1.5 Labels .................................. 15
   4.1.6 Showing the Window ............... 15
   4.1.7 The Main Event Loop .............. 15
4.2 Naming Conventions ................... 16
4.3 Header Files .......................... 16
4.4 Compiling Programs that Use FLTK ............ 16
   4.4.1 Compiling Programs with Standard Compilers .......... 16
   4.4.2 Compiling Programs with Makefiles ............ 18
   4.4.3 Compiling Programs with Microsoft Visual C++ .......... 18

## 5 Common Widgets and Attributes

5.1 Buttons ................................ 19
5.2 Text .................................. 20
5.3 Valuators .............................. 20
5.4 Groups ................................ 21
5.5 Setting the Size and Position of Widgets ............ 22
5.6 Colors ................................ 22
5.7 Box Types ................................................................. 23
  5.7.1 Making Your Own Boxtypes ...................................... 23
5.8 Labels and Label Types .................................................. 25
5.9 Callbacks ..................................................................... 29
5.10 Shortcuts .................................................................... 29

6 Coordinates and Layout Widgets .................................................... 31
  6.1 The widget coordinate system ............................................... 31
  6.2 Layout and container widgets ................................................ 32
    6.2.1 The Fl_Pack layout widget ........................................... 32
    6.2.2 The Fl_Scroll container widget ...................................... 33
    6.2.3 The Fl_Tabs container widget ........................................ 33
    6.2.4 The Fl_Tile layout widget .............................................. 33
    6.2.5 The Fl_Wizard container widget ..................................... 34

7 How Does Resizing Work? ............................................................. 35
  7.1 Resizing can be disabled ...................................................... 35
  7.2 Resizing can be simple ........................................................ 35
  7.3 Resizing can be complex ....................................................... 36
  7.4 Practical examples ............................................................. 37

8 Designing a Simple Text Editor ....................................................... 41
  8.1 Determining the Goals of the Text Editor ................................. 41
  8.2 Designing the Main Window .................................................. 41
  8.3 Variables ..................................................................... 42
  8.4 Menubars and Menus .......................................................... 42
  8.5 Editing the Text ................................................................ 42
  8.6 The Replace Dialog ............................................................. 43
  8.7 Callbacks .................................................................... 43
    8.7.1 changed_cb() ................................................................ 43
    8.7.2 copy_cb() ................................................................... 43
    8.7.3 cut_cb() ...................................................................... 43
    8.7.4 delete_cb() .................................................................. 44
    8.7.5 find_cb() ..................................................................... 44
    8.7.6 find2_cb() .................................................................... 44
    8.7.7 new_cb() ...................................................................... 44
    8.7.8 open_cb() ..................................................................... 45
    8.7.9 paste_cb() ..................................................................... 45
    8.7.10 quit_cb() ..................................................................... 45
    8.7.11 replace_cb() ................................................................ 45
    8.7.12 replace2_cb() ............................................................... 45
    8.7.13 replall_cb() ................................................................ 46
    8.7.14 replcan_cb() ............................................................... 46
12.2.2 The draw() Method ........................................ 92
12.2.3 The handle() Method .................................... 92
12.3 OpenGL and support of HighDPI displays ................. 93
12.4 Using OpenGL in Normal FLTK Windows .................. 93
12.5 Using FLTK widgets in OpenGL Windows .................. 94
12.6 OpenGL Drawing Functions ................................ 95
12.7 Speeding up OpenGL ........................................ 96
12.8 Using OpenGL Optimizer with FLTK ....................... 97
12.9 Using OpenGL 3.0 (or higher versions) ................. 98

13 Programming with FLUID ........................................ 101
13.1 What is FLUID? ............................................. 101
13.2 Running FLUID Under UNIX ................................ 102
13.3 Running FLUID Under Microsoft Windows ............... 102
13.4 Compiling .fl Files .......................................... 103
13.5 A Short Tutorial ............................................. 103
  13.5.1 The CubeView Class ..................................... 104
  13.5.2 The CubeViewUI Class .................................. 107
  13.5.3 Adding Constructor Initialization Code ............... 110
  13.5.4 Generating the Code ..................................... 110
13.6 FLUID Reference ............................................. 110
  13.6.1 The Widget Browser ...................................... 110
  13.6.2 Menu Items ................................................ 111
  13.6.3 The Widget Panel ....................................... 120
13.7 GUI Attributes ................................................ 120
  13.7.1 Style Attributes ......................................... 123
  13.7.2 C++ Attributes ........................................... 125
13.8 Selecting and Moving Widgets ............................. 128
13.9 Image Labels .................................................. 129
13.10 FLUID Templates ............................................ 131
13.11 Internationalization with FLUID ......................... 131
  13.11.1 I18N Methods ........................................... 131
  13.11.2 Using GNU gettext for I18N ......................... 132
  13.11.3 Using POSIX catgets for I18N ....................... 133
13.12 Known Limitations ......................................... 133
13.13 Keyboard Shortcuts .......................................... 133

14 Advanced FLTK .................................................. 137
14.1 Multithreading ............................................... 137
14.2 FLTK multithread locking - Fl::lock() and Fl::unlock() 137
14.3 Simple multithreaded examples using Fl::lock ......... 138
14.4 FLTK multithreaded “lockless programming” ........... 140
14.5 FLTK multithreaded Constraints .......................... 141
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1</td>
<td>About Unicode, ISO 10646 and UTF-8</td>
<td>143</td>
</tr>
<tr>
<td>15.2</td>
<td>Unicode in FLTK</td>
<td>145</td>
</tr>
<tr>
<td>15.3</td>
<td>Illegal Unicode and UTF-8 Sequences</td>
<td>146</td>
</tr>
<tr>
<td>15.4</td>
<td>FLTK Unicode and UTF-8 Functions</td>
<td>146</td>
</tr>
<tr>
<td>15.5</td>
<td>FLTK Unicode Versions of System Calls</td>
<td>150</td>
</tr>
<tr>
<td>16.1</td>
<td>Version Numbers</td>
<td>151</td>
</tr>
<tr>
<td>16.2</td>
<td>Events</td>
<td>152</td>
</tr>
<tr>
<td>16.3</td>
<td>Callback &quot;When&quot; Conditions</td>
<td>153</td>
</tr>
<tr>
<td>16.4</td>
<td>Fl::event_button() Values</td>
<td>153</td>
</tr>
<tr>
<td>16.5</td>
<td>Fl::event_key() Values</td>
<td>153</td>
</tr>
<tr>
<td>16.6</td>
<td>Fl::event_state() Values</td>
<td>154</td>
</tr>
<tr>
<td>16.7</td>
<td>Alignment Values</td>
<td>155</td>
</tr>
<tr>
<td>16.8</td>
<td>Fonts</td>
<td>155</td>
</tr>
<tr>
<td>16.9</td>
<td>Colors</td>
<td>156</td>
</tr>
<tr>
<td>16.9.1</td>
<td>Color Constants</td>
<td>156</td>
</tr>
<tr>
<td>16.10</td>
<td>Cursors</td>
<td>157</td>
</tr>
<tr>
<td>16.11</td>
<td>FD &quot;When&quot; Conditions</td>
<td>158</td>
</tr>
<tr>
<td>16.12</td>
<td>Damage Masks</td>
<td>158</td>
</tr>
<tr>
<td>17.1</td>
<td>Using the GLUT Compatibility Header File</td>
<td>159</td>
</tr>
<tr>
<td>17.2</td>
<td>Known Problems</td>
<td>159</td>
</tr>
<tr>
<td>17.3</td>
<td>Mixing GLUT and FLTK Code</td>
<td>160</td>
</tr>
<tr>
<td>17.4</td>
<td>class Fl_Glut_Window</td>
<td>161</td>
</tr>
<tr>
<td>17.4.1</td>
<td>Class Hierarchy</td>
<td>161</td>
</tr>
<tr>
<td>17.4.2</td>
<td>Include Files</td>
<td>161</td>
</tr>
<tr>
<td>17.4.3</td>
<td>Description</td>
<td>161</td>
</tr>
<tr>
<td>17.4.4</td>
<td>Members</td>
<td>161</td>
</tr>
<tr>
<td>17.4.5</td>
<td>Methods</td>
<td>162</td>
</tr>
<tr>
<td>18.1</td>
<td>Importing Forms Layout Files</td>
<td>163</td>
</tr>
<tr>
<td>18.2</td>
<td>Using the Compatibility Header File</td>
<td>163</td>
</tr>
<tr>
<td>18.3</td>
<td>Problems You Will Encounter</td>
<td>164</td>
</tr>
<tr>
<td>18.4</td>
<td>Additional Notes</td>
<td>165</td>
</tr>
<tr>
<td>19.1</td>
<td>Accessing the OS Interfaces</td>
<td>169</td>
</tr>
<tr>
<td>19.2</td>
<td>The UNIX (X11) Interface</td>
<td>170</td>
</tr>
<tr>
<td>19.2.1</td>
<td>Handling Other X Events</td>
<td>170</td>
</tr>
<tr>
<td>19.2.2</td>
<td>Drawing using Xlib</td>
<td>171</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>19.2.3 Changing the Display, Screen, or X Visual</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>19.2.4 Using a Subclass of Fl_Window for Special X Stuff</td>
<td>173</td>
<td></td>
</tr>
<tr>
<td>19.2.5 Setting the Icon of a Window</td>
<td>174</td>
<td></td>
</tr>
<tr>
<td>19.2.6 X Resources</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>19.2.7 Display Scaling Factor</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>19.3 The Windows Interface</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>19.3.1 Using filenames with non-ASCII characters</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>19.3.2 Responding to WM_QUIT</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>19.3.3 Handling Other Windows API Messages</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>19.3.4 Drawing Things Using the Windows GDI</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>19.3.5 HighDPI support</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>19.3.6 Display Scaling Factor</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>19.3.7 Setting the Icon of a Window</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>19.3.8 How to Not Get a MSDOS Console Window</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>19.3.9 Known Windows Bugs and Problems</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>19.4 The Apple OS X Interface</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>19.4.1 Setting the icon of an application</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>19.4.2 Drawing Things Using Quartz</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>19.4.3 Internationalization</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>19.4.4 OpenGL and ‘retina’ displays</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>19.4.5 Fl_Double_Window</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>19.4.6 Mac File System Specifics</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>19.5 The Wayland Interface</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>19.5.1 HiDPI display support</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>19.5.2 Window icons</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>19.5.3 Window titlebars</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>20 Migrating Code from FLTK 1.3 to 1.4</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>20.1 Migrating from FLTK 1.0 or 1.1 to 1.4</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>20.2 Changes in Header Files</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>20.3 Fl_Preferences</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>20.4 Fl::add_timeout and friends</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>20.5 Fl_Image::copy() ‘const’</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>21 Developer Information</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>21.1 Non-ASCII Characters</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>21.2 Document Structure</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>21.3 Creating Links</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>21.4 Paragraph Layout</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>21.5 Navigation Elements</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>22 Software License</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>23 Example Source Code</td>
<td>199</td>
<td></td>
</tr>
</tbody>
</table>
24.1 Where do I start learning FLTK? .................................................. 211
24.2 How do I make a box with text? .................................................. 211
24.3 Can I use FLTK to make closed-source commercial applications? ........ 211
24.4 Hitting the 'Escape' key closes windows - how do I prevent this? .......... 212

25 Todo List .......................................................... 213

26 Deprecated List ........................................................................ 217

27 Module Index ............................................................................ 221
27.1 Modules .................................................................................. 221

28 Hierarchical Index ................................................................. 223
28.1 Class Hierarchy .................................................................... 223

29 Class Index .............................................................................. 227
29.1 Class List .............................................................................. 227

30 File Index ................................................................................. 233
30.1 File List ................................................................................ 233

31 Module Documentation .......................................................... 239
31.1 Callback Function Typedefs ..................................................... 239
31.1.1 Detailed Description .......................................................... 240
31.1.2 Typedef Documentation ..................................................... 240
31.1.2.1 Fl_Event_Dispatch ....................................................... 240
31.2 Windows handling functions .................................................. 240
31.2.1 Detailed Description .......................................................... 240
31.2.2 Function Documentation ................................................... 241
31.2.2.1 default_atclose() ......................................................... 241
31.2.2.2 first_window() [1/2] .................................................... 241
31.2.2.3 first_window() [2/2] .................................................... 241
31.2.2.4 grab() [1/2] .................................................................. 241
31.2.2.5 grab() [2/2] .................................................................. 241
31.2.2.6 modal() ....................................................................... 241
31.2.2.7 next_window() ........................................................... 242
31.2.2.8 set_atclose() ................................................................ 242
31.2.3 Variable Documentation .................................................. 242
31.2.3.1 atclose ....................................................................... 242
31.3 Events handling functions ..................................................... 242
31.3.1 Detailed Description .......................................................... 242
31.3.2 Function Documentation ................................................... 245
31.3.2.1 add_handler() .......................................................... 245
31.3.2.2 add_system_handler() ............................................... 245
31.3.2.3 belowmouse() [1/2] .................................................... 246
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.3.2.4 belowmouse()</td>
<td>246</td>
</tr>
<tr>
<td>31.3.2.5 compose()</td>
<td>246</td>
</tr>
<tr>
<td>31.3.2.6 compose_reset()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.7 disable_im()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.8 enable_im()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.9 event()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.10 event_button()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.11 event_button1()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.12 event_button2()</td>
<td>247</td>
</tr>
<tr>
<td>31.3.2.13 event_button3()</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.14 event_buttons()</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.15 event_clicks()[1/2]</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.16 event_clicks()[2/2]</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.17 event_clipboard()</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.18 event_clipboard_type()</td>
<td>248</td>
</tr>
<tr>
<td>31.3.2.19 event_dispatch()</td>
<td>249</td>
</tr>
<tr>
<td>31.3.2.20 event_dx()</td>
<td>249</td>
</tr>
<tr>
<td>31.3.2.21 event_dy()</td>
<td>249</td>
</tr>
<tr>
<td>31.3.2.22 event_inside()[1/2]</td>
<td>249</td>
</tr>
<tr>
<td>31.3.2.23 event_inside()[2/2]</td>
<td>250</td>
</tr>
<tr>
<td>31.3.2.24 event_is_click()[1/2]</td>
<td>250</td>
</tr>
<tr>
<td>31.3.2.25 event_is_click()[2/2]</td>
<td>250</td>
</tr>
<tr>
<td>31.3.2.26 event_key()[1/2]</td>
<td>251</td>
</tr>
<tr>
<td>31.3.2.27 event_key()[2/2]</td>
<td>251</td>
</tr>
<tr>
<td>31.3.2.28 event_length()</td>
<td>251</td>
</tr>
<tr>
<td>31.3.2.29 event_original_key()</td>
<td>251</td>
</tr>
<tr>
<td>31.3.2.30 event_state()[1/2]</td>
<td>252</td>
</tr>
<tr>
<td>31.3.2.31 event_state()[2/2]</td>
<td>252</td>
</tr>
<tr>
<td>31.3.2.32 event_text()</td>
<td>252</td>
</tr>
<tr>
<td>31.3.2.33 event_x_root()</td>
<td>252</td>
</tr>
<tr>
<td>31.3.2.34 event_y_root()</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.35 focus()[1/2]</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.36 focus()[2/2]</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.37 get_key()</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.38 get_mouse()</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.39 handle()</td>
<td>253</td>
</tr>
<tr>
<td>31.3.2.40 handle_()</td>
<td>254</td>
</tr>
<tr>
<td>31.3.2.41 pushed()[1/2]</td>
<td>254</td>
</tr>
<tr>
<td>31.3.2.42 pushed()[2/2]</td>
<td>254</td>
</tr>
<tr>
<td>31.3.2.43 remove_handler()</td>
<td>255</td>
</tr>
<tr>
<td>31.3.2.44 remove_system_handler()</td>
<td>255</td>
</tr>
<tr>
<td>31.3.2.45 testShortcut()</td>
<td>255</td>
</tr>
</tbody>
</table>
### 31.3.3 Variable Documentation
- 31.3.3.1 `fl_eventnames` .................................................. 255
- 31.3.3.2 `fl_fontnames` .................................................... 256

### 31.4 Selection & Clipboard functions
- 31.4.1 Detailed Description .................................................. 256
- 31.4.2 Function Documentation .............................................. 257
  - 31.4.2.1 `add_clipboard_notify()` ...................................... 257
  - 31.4.2.2 `clipboard_contains()` ...................................... 257
  - 31.4.2.3 `copy()` .......................................................... 257
  - 31.4.2.4 `dnd()` .................................................................. 258
  - 31.4.2.5 `paste()` .......................................................... 258
  - 31.4.2.6 `paste()` .......................................................... 258
  - 31.4.2.7 `selection()` ....................................................... 259
  - 31.4.2.8 `selection_owner()` ........................................... 259

### 31.5 Screen functions
- 31.5.1 Detailed Description .................................................. 259
- 31.5.2 Function Documentation .............................................. 260
  - 31.5.2.1 `keyboard_screen_scaling()` .................................... 260
  - 31.5.2.2 `screen_dpi()` .................................................... 261
  - 31.5.2.3 `screen_num()` ................................................... 261
  - 31.5.2.4 `screen_num()` ................................................... 261
  - 31.5.2.5 `screen_scale()` .................................................. 262
  - 31.5.2.6 `screen_scaling_supported()` .................................. 262
  - 31.5.2.7 `screen_work_area()` ......................................... 262
  - 31.5.2.8 `screen_work_area()` ......................................... 262
  - 31.5.2.9 `screen_work_area()` ......................................... 263
  - 31.5.2.10 `screen_xywh()` ............................................... 263
  - 31.5.2.11 `screen_xywh()` ............................................... 263
  - 31.5.2.12 `screen_xywh()` ............................................... 264
  - 31.5.2.13 `screen_xywh()` ............................................... 264

### 31.6 Color & Font functions
- 31.6.1 Detailed Description .................................................. 264
- 31.6.2 Function Documentation .............................................. 266
  - 31.6.2.1 `fl_color()` ....................................................... 266
  - 31.6.2.2 `fl_color()` ....................................................... 266
  - 31.6.2.3 `fl_color()` ....................................................... 266
  - 31.6.2.4 `fl_color_average()` .......................................... 267
  - 31.6.2.5 `fl_contrast()` ................................................... 267
  - 31.6.2.6 `fl_font()` ......................................................... 267
  - 31.6.2.7 `fl_font()` ......................................................... 267
  - 31.6.2.8 `fl_height()` ..................................................... 268
31.6.2.9 fl_height() [2/2] .......................................................... 268
31.6.2.10 fl_latin1_to_local() .................................................. 268
31.6.2.11 fl_local_to_latin1() .................................................. 268
31.6.2.12 fl_local_to_mac_roman() ........................................... 268
31.6.2.13 fl_mac_roman_to_local() ........................................... 269
31.6.2.14 fl_show_colormap() .................................................. 269
31.6.2.15 fl_size() .............................................................. 270
31.6.2.16 fl_text_extents() [1/2] .............................................. 270
31.6.2.17 fl_text_extents() [2/2] .............................................. 271
31.6.2.18 fl_width() ............................................................ 271
31.6.2.19 free_color() ........................................................... 271
31.6.2.20 get_color() [1/3] ..................................................... 271
31.6.2.21 get_color() [2/3] ..................................................... 272
31.6.2.22 get_color() [3/3] ..................................................... 272
31.6.2.23 get_font() ........................................................... 272
31.6.2.24 get_font_name() ..................................................... 272
31.6.2.25 get_font_sizes() ..................................................... 272
31.6.2.26 set_color() [1/3] ..................................................... 273
31.6.2.27 set_color() [2/3] ..................................................... 273
31.6.2.28 set_color() [3/3] ..................................................... 273
31.6.2.29 set_font() ........................................................... 273
31.6.2.30 set_fonts() ........................................................... 274

31.7 Drawing functions .......................................................... 274

31.7.1 Detailed Description ..................................................... 278

31.7.2 Enumeration Type Documentation .................................... 279

31.7.2.1 anonymous enum .................................................... 279

31.7.3 Function Documentation ................................................ 279

31.7.3.1 fl_add_symbol() ..................................................... 279
31.7.3.2 fl_antialias() ........................................................ 279
31.7.3.3 fl_arc() [1/2] ......................................................... 280
31.7.3.4 fl_arc() [2/2] ......................................................... 280
31.7.3.5 fl_begin_complex_polygon() ..................................... 281
31.7.3.6 fl_begin_offscreen() .............................................. 281
31.7.3.7 fl_begin_points() .................................................. 281
31.7.3.8 fl_can_do_alpha_blending() ..................................... 282
31.7.3.9 fl_capture_window() ................................................ 282
31.7.3.10 fl_circle() .......................................................... 282
31.7.3.11 fl_clip() ............................................................. 283
31.7.3.12 fl_clip_box() ....................................................... 283
31.7.3.13 fl_clip_region() [1/2] ........................................... 284
31.7.3.14 fl_clip_region() [2/2] ........................................... 284
31.7.3.15 fl_copy_offscreen() .............................................. 284
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_create_offscreen()</td>
<td>285</td>
</tr>
<tr>
<td>fl_cursor()</td>
<td>285</td>
</tr>
<tr>
<td>fl_curve()</td>
<td>285</td>
</tr>
<tr>
<td>fl_delete_offscreen()</td>
<td>286</td>
</tr>
<tr>
<td>fl_draw()</td>
<td>286</td>
</tr>
<tr>
<td>fl_draw()</td>
<td>286</td>
</tr>
<tr>
<td>fl_draw()</td>
<td>286</td>
</tr>
<tr>
<td>fl_draw_box()</td>
<td>287</td>
</tr>
<tr>
<td>fl_draw_check()</td>
<td>287</td>
</tr>
<tr>
<td>fl_draw_image()</td>
<td>288</td>
</tr>
<tr>
<td>fl_draw_image()</td>
<td>288</td>
</tr>
<tr>
<td>fl_draw_image_mono()</td>
<td>289</td>
</tr>
<tr>
<td>fl_draw_image_mono()</td>
<td>289</td>
</tr>
<tr>
<td>fl_draw_pixmap()</td>
<td>290</td>
</tr>
<tr>
<td>fl_draw_pixmap()</td>
<td>290</td>
</tr>
<tr>
<td>fl_draw_symbol()</td>
<td>290</td>
</tr>
<tr>
<td>fl_expand_text()</td>
<td>291</td>
</tr>
<tr>
<td>fl_focus_rect()</td>
<td>291</td>
</tr>
<tr>
<td>fl_frame()</td>
<td>291</td>
</tr>
<tr>
<td>fl_frame2()</td>
<td>291</td>
</tr>
<tr>
<td>fl_gap()</td>
<td>292</td>
</tr>
<tr>
<td>fl_line_style()</td>
<td>292</td>
</tr>
<tr>
<td>fl_measure()</td>
<td>292</td>
</tr>
<tr>
<td>fl_measure_pixmap()</td>
<td>293</td>
</tr>
<tr>
<td>fl_measure_pixmap()</td>
<td>293</td>
</tr>
<tr>
<td>fl_mult_matrix()</td>
<td>294</td>
</tr>
<tr>
<td>fl_not_clipped()</td>
<td>294</td>
</tr>
<tr>
<td>fl_old_shortcut()</td>
<td>294</td>
</tr>
<tr>
<td>fl_overwrite_scale()</td>
<td>295</td>
</tr>
<tr>
<td>fl_overscan()</td>
<td>295</td>
</tr>
<tr>
<td>fl_pie()</td>
<td>295</td>
</tr>
<tr>
<td>fl_polygon()</td>
<td>296</td>
</tr>
<tr>
<td>fl_polygon()</td>
<td>296</td>
</tr>
<tr>
<td>fl_pop_clip()</td>
<td>296</td>
</tr>
<tr>
<td>fl_push_clip()</td>
<td>296</td>
</tr>
<tr>
<td>fl_push_matrix()</td>
<td>297</td>
</tr>
<tr>
<td>fl_read_image()</td>
<td>297</td>
</tr>
<tr>
<td>fl_rect()</td>
<td>297</td>
</tr>
<tr>
<td>fl_rect()</td>
<td>297</td>
</tr>
<tr>
<td>fl_rectf()</td>
<td>298</td>
</tr>
<tr>
<td>fl_rectf()</td>
<td>298</td>
</tr>
<tr>
<td>fl_rescale_offscreen()</td>
<td>298</td>
</tr>
<tr>
<td>fl_rescale_offscreen()</td>
<td>298</td>
</tr>
</tbody>
</table>
31.7.3.58 fl_reset_spot() .................................................. 298
31.7.3.59 fl_restore_scale() ................................................. 299
31.7.3.60 fl_rotate() .......................................................... 299
31.7.3.61 fl_scale() [1/2] ..................................................... 299
31.7.3.62 fl_scale() [2/2] ..................................................... 299
31.7.3.63 fl_scroll() .......................................................... 299
31.7.3.64 fl_set_spot() ....................................................... 300
31.7.3.65 fl_set_status() ..................................................... 300
31.7.3.66 fl_shortcut_label() [1/2] ....................................... 300
31.7.3.67 fl_shortcut_label() [2/2] ....................................... 301
31.7.3.68 fl_transform_dx() ................................................ 302
31.7.3.69 fl_transform_dy() ................................................ 302
31.7.3.70 fl_transform_x() ................................................ 302
31.7.3.71 fl_transform_y() ................................................ 302
31.7.3.72 fl_transformed_vertex() ....................................... 302
31.7.3.73 fl_translate() ................................................... 303
31.7.3.74 fl_vertex() ........................................................ 303

31.8 Multithreading support functions .................................. 303
31.8.1 Detailed Description .................................................. 303
31.8.2 Function Documentation ............................................. 303
31.8.2.1 awake() [1/2] ...................................................... 304
31.8.2.2 awake() [2/2] ...................................................... 304
31.8.2.3 lock() ............................................................... 304
31.8.2.4 thread_message() ................................................ 304
31.8.2.5 unlock() ........................................................... 304

31.9 Safe widget deletion support functions ............................ 305
31.9.1 Detailed Description .................................................. 305
31.9.2 Function Documentation ............................................. 305
31.9.2.1 clear_widget_pointer() ......................................... 305
31.9.2.2 delete_widget() .................................................. 306
31.9.2.3 do_widget_deletion() ............................................ 306
31.9.2.4 release_widget_pointer() ....................................... 307
31.9.2.5 watch_widget_pointer() ....................................... 307

31.10 Cairo Support Functions and Classes ............................. 307
31.10.1 Detailed Description .................................................. 308
31.10.2 Function Documentation ............................................. 308
31.10.2.1 cairo_autolink_context() [1/2] ............................. 308
31.10.2.2 cairo_autolink_context() [2/2] ............................. 308
31.10.2.3 cairo_cc() ....................................................... 309
31.10.2.4 cairo_make_current() ........................................ 309

31.11 Unicode and UTF-8 functions ...................................... 309
31.11.1 Detailed Description .................................................. 311
31.11.3.38 fl_utf_tolower() ................................. 324
31.11.3.39 fl_utf_toupper() ................................. 324
31.11.3.40 fl_wcwidth() .................................. 324
31.11.3.41 fl_wcwidth_() ................................ 325

31.12 Mac OS X-specific symbols .......................... 325
31.12.1 Detailed Description ............................... 326
31.12.2 Function Documentation ............................ 326
31.12.2.1 fl_mac_set_about() .............................. 326
31.12.2.2 fl_open_callback() .............................. 326

31.12.3 Variable Documentation ............................ 326
31.12.3.1 fl_mac_os_version ............................... 326

31.13 Common Dialog Classes and Functions .............. 326
31.13.1 Detailed Description ............................... 328
31.13.2 Function Documentation ............................ 328
31.13.2.1 fl_alert() ...................................... 328
31.13.2.2 fl_ask() ......................................... 329
31.13.2.3 fl_beep() ....................................... 329
31.13.2.4 fl_choice() ..................................... 329
31.13.2.5 fl_choice_n() .................................. 331
31.13.2.6 fl_color_chooser() [1/2] ........................ 331
31.13.2.7 fl_color_chooser() [2/2] ........................ 332
31.13.2.8 fl_dir_chooser() ................................ 333
31.13.2.9 fl_file_chooser() ................................ 334
31.13.2.10 fl_file_chooser_callback() .................... 335
31.13.2.11 fl_file_chooser_ok_label() .................... 335
31.13.2.12 fl_input() ...................................... 335
31.13.2.13 fl_input_str() .................................. 335
31.13.2.14 fl_message() ................................... 336
31.13.2.15 fl_message_hotspot() [1/2] .................... 336
31.13.2.16 fl_message_hotspot() [2/2] .................... 337
31.13.2.17 fl_message_icon() ............................. 337
31.13.2.18 fl_message_icon_label() ....................... 338
31.13.2.19 fl_message_position() [1/3] ................... 338
31.13.2.20 fl_message_position() [2/3] ................... 338
31.13.2.22 fl_message_title() ............................. 340
31.13.2.23 fl_message_title_default() ................... 341
31.13.2.24 fl_password() .................................. 341
31.13.2.25 fl_password_str() .............................. 342

31.13.3 Variable Documentation ............................ 342
31.13.3.1 error ............................................ 342
31.13.3.2 fatal ............................................. 342
31.13.3 warning ................................................. 343
31.14 File names and URI utility functions .............................................. 343
  31.14.1 Detailed Description ................................................. 344
  31.14.2 Typedef Documentation ................................................. 344
    31.14.2.1 Fl_File_Sort_F ................................................. 344
  31.14.3 Function Documentation ................................................. 344
    31.14.3.1 fl_decode_uri() ................................................. 344
    31.14.3.2 fl_filename_absolute() ........................................ 344
    31.14.3.3 fl_filename_expand() .......................................... 345
    31.14.3.4 fl_filename_ext() .............................................. 345
    31.14.3.5 fl_filename_free_list() ...................................... 345
    31.14.3.6 fl_filename_isdir() ........................................... 346
    31.14.3.7 fl_filename_list() ............................................. 346
    31.14.3.8 fl_filename_match() .......................................... 347
    31.14.3.9 fl_filename_name() ............................................ 348
    31.14.3.10 fl_filename_relative() ...................................... 348
    31.14.3.11 fl_filename_setext() ....................................... 349
    31.14.3.12 fl_open_uri() ................................................... 349
31.15 Fl_string ......................................................... 350
  31.15.1 Detailed Description .................................................... 350
  31.15.2 Function Documentation ................................................... 350
    31.15.2.1 fl_strdup() ...................................................... 350
32 Class Documentation .......................................................... 351
  32.1 Fl_Preferences::Entry Struct Reference ....................................... 351
  32.2 Fl Class Reference ................................................................ 351
    32.2.1 Detailed Description .................................................... 360
    32.2.2 Member Enumeration Documentation .................................... 360
      32.2.2.1 Fl_Option ............................................................. 360
    32.2.3 Member Function Documentation .......................................... 361
      32.2.3.1 abi_check() ........................................................ 361
      32.2.3.2 abi_version() .................................................... 361
      32.2.3.3 add_check() ..................................................... 361
      32.2.3.4 add_fd() [1/2] .................................................. 362
      32.2.3.5 add_fd() [2/2] .................................................. 362
      32.2.3.6 add_idle() ....................................................... 362
      32.2.3.7 add_timeout() .................................................. 362
      32.2.3.8 api_version() .................................................. 363
      32.2.3.9 arg() .............................................................. 363
      32.2.3.10 arg() [1/2] ..................................................... 364
      32.2.3.11 arg() [2/2] ..................................................... 364
      32.2.3.12 background() .................................................. 364
32.2.3.13 background2() .................................................. 365
32.2.3.14 box_border_radius_max() [1/2] .......................... 365
32.2.3.15 box_border_radius_max() [2/2] .......................... 365
32.2.3.16 box_color() ......................................................... 365
32.2.3.17 box_dh() .......................................................... 365
32.2.3.18 box_dw() .......................................................... 366
32.2.3.19 box_dx() .......................................................... 366
32.2.3.20 box_dy() .......................................................... 366
32.2.3.21 box_shadow_width() [1/2] ................................. 366
32.2.3.22 box_shadow_width() [2/2] ................................. 366
32.2.3.23 check() ............................................................ 367
32.2.3.24 display() .......................................................... 367
32.2.3.25 dnd_text_ops() [1/2] .......................................... 367
32.2.3.26 dnd_text_ops() [2/2] .......................................... 367
32.2.3.27 draw_box_active() ............................................. 367
32.2.3.28 draw_GL_text_with_textures() [1/2] .................... 367
32.2.3.29 draw_GL_text_with_textures() [2/2] .................... 368
32.2.3.30 flush() .............................................................. 368
32.2.3.31 get_system_colors() .......................................... 368
32.2.3.32 gl_visual() ........................................................ 368
32.2.3.33 is_scheme() ....................................................... 368
32.2.3.34 menu_linespacing() [1/2] .................................... 369
32.2.3.35 menu_linespacing() [2/2] .................................... 369
32.2.3.36 option() [1/2] ...................................................... 369
32.2.3.37 option() [2/2] ...................................................... 370
32.2.3.38 own_colormap() .................................................. 370
32.2.3.39 program_should_quit() [1/2] ............................... 371
32.2.3.40 program_should_quit() [2/2] ............................... 371
32.2.3.41 readqueue() ....................................................... 371
32.2.3.42 ready() ............................................................ 371
32.2.3.43 release() .......................................................... 372
32.2.3.44 reload_scheme() ............................................... 372
32.2.3.45 remove_check() .................................................. 372
32.2.3.46 remove_timeout() .............................................. 372
32.2.3.47 repeat_timeout() .............................................. 372
32.2.3.48 run() ............................................................... 373
32.2.3.49 scheme() .......................................................... 373
32.2.3.50 scrollbar_size() [1/2] ........................................ 373
32.2.3.51 scrollbar_size() [2/2] ........................................ 373
32.2.3.52 set_box_color() ................................................ 374
32.2.3.53 set_idle() ........................................................ 374
32.2.3.54 use_high_res_GL() [1/2] ..................................... 374
32.2.3.55 use_high_res_GL()  [2/2] .............................................................. 374
32.2.3.56 version() .................................................................................. 375
32.2.3.57 visible_focus() [1/2] ................................................................. 375
32.2.3.58 visible_focus() [2/2] ................................................................. 375
32.2.3.59 visual() ...................................................................................... 375
32.2.3.60 wait() [1/2] ............................................................................... 375
32.2.3.61 wait() [2/2] ............................................................................... 376
32.2.4 Member Data Documentation ........................................................... 376
32.2.4.1 help .............................................................................................. 376
32.2.4.2 idle .............................................................................................. 376
32.3 Fl_Adjuster Class Reference ............................................................... 377
32.3.1 Detailed Description ....................................................................... 377
32.3.2 Constructor & Destructor Documentation ........................................ 378
32.3.2.1 Fl_Adjuster() .............................................................................. 378
32.3.3 Member Function Documentation .................................................. 378
32.3.3.1 draw() ......................................................................................... 378
32.3.3.2 handle() ...................................................................................... 378
32.3.3.3 soft() [1/2] .................................................................................. 379
32.3.3.4 soft() [2/2] .................................................................................. 379
32.4 Fl_Bitmap Class Reference .................................................................. 379
32.4.1 Detailed Description ....................................................................... 380
32.4.2 Member Function Documentation .................................................. 380
32.4.2.1 copy() ......................................................................................... 380
32.4.2.2 draw() ......................................................................................... 380
32.4.2.3 label() [1/2] ................................................................................ 381
32.4.2.4 label() [2/2] ................................................................................ 381
32.4.2.5 uncache() .................................................................................... 381
32.5 Fl_BMP_Image Class Reference .......................................................... 381
32.5.1 Detailed Description ....................................................................... 382
32.5.2 Constructor & Destructor Documentation ........................................ 382
32.5.2.1 Fl_BMP_Image() [1/2] ................................................................ 382
32.5.2.2 Fl_BMP_Image() [2/2] ................................................................ 382
32.6 Fl_Box Class Reference ...................................................................... 383
32.6.1 Detailed Description ....................................................................... 383
32.6.2 Constructor & Destructor Documentation ........................................ 384
32.6.2.1 Fl_Box() ....................................................................................... 384
32.6.3 Member Function Documentation .................................................. 384
32.6.3.1 draw() ......................................................................................... 384
32.6.3.2 handle() ...................................................................................... 384
32.7 Fl_Browser Class Reference ............................................................... 385
32.7.1 Detailed Description ....................................................................... 388
32.7.2 Constructor & Destructor Documentation ........................................ 388
32.7.3.41 remove() .............................................. 401
32.7.3.42 remove_icon() ...................................... 402
32.7.3.43 select() ............................................. 402
32.7.3.44 selected() .......................................... 402
32.7.3.45 show() .............................................. 403
32.7.3.46 size() .............................................. 403
32.7.3.47 swap() [1/2] ....................................... 403
32.7.3.48 swap() [2/2] ....................................... 403
32.7.3.49 text() [1/2] ......................................... 404
32.7.3.50 text() [2/2] ......................................... 404
32.7.3.51 textsize() ........................................... 404
32.7.3.52 topline() [1/2] ..................................... 404
32.7.3.53 topline() [2/2] ..................................... 405
32.7.3.54 value() [1/2] ....................................... 405
32.7.3.55 value() [2/2] ....................................... 405
32.7.3.56 visible() ........................................... 405

32.8 Fl_Browser_ Class Reference ........................................ 406
32.8.1 Detailed Description ..................................... 409
32.8.1.1 Keyboard navigation of browser items .................. 409
32.8.2 Member Enumeration Documentation ..................... 409
32.8.2.1 anonymous enum ................................... 409
32.8.3 Constructor & Destructor Documentation .................. 409
32.8.3.1 Fl_Browser_() ...................................... 410
32.8.4 Member Function Documentation .......................... 410
32.8.4.1 bbox() ............................................... 410
32.8.4.2 deleting() ............................................ 410
32.8.4.3 deselect() .......................................... 410
32.8.4.4 display() ............................................ 411
32.8.4.5 displayed() ......................................... 411
32.8.4.6 find_item() .......................................... 411
32.8.4.7 full_height() ........................................ 412
32.8.4.8 full_width() ......................................... 412
32.8.4.9 handle() ............................................. 412
32.8.4.10 has_scrollbar() .................................... 412
32.8.4.11 hposition() [1/2] .................................. 413
32.8.4.12 hposition() [2/2] .................................. 413
32.8.4.13 incr_height() ....................................... 413
32.8.4.14 inserting() ......................................... 413
32.8.4.15 item_at() .......................................... 414
32.8.4.16 item_first() ....................................... 414
32.8.4.17 item_height() ...................................... 414
32.8.4.18 item_last() ........................................ 414
32.8.4.19 item_next() ......................................................... 415
32.8.4.20 item_prev() ....................................................... 415
32.8.4.21 item_quick_height() ........................................... 415
32.8.4.22 item_select() ..................................................... 415
32.8.4.23 item_selected() .................................................. 416
32.8.4.24 item_swap() ....................................................... 416
32.8.4.25 item_selected() .................................................. 416
32.8.4.26 item_width() ...................................................... 416
32.8.4.27 leftedge() ........................................................ 417
32.8.4.28 new_list() ........................................................ 417
32.8.4.29 position() [1/2] ................................................... 417
32.8.4.30 position() [2/2] ................................................... 417
32.8.4.31 redraw_line() ..................................................... 418
32.8.4.32 redraw_lines() .................................................... 418
32.8.4.33 replacing() ........................................................ 418
32.8.4.34 resize() ............................................................ 418
32.8.4.35 scrollbar_left() .................................................. 419
32.8.4.36 scrollbar_right() ................................................ 419
32.8.4.37 scrollbar_size() [1/2] .......................................... 419
32.8.4.38 scrollbar_size() [2/2] .......................................... 419
32.8.4.39 scrollbar_width() [1/2] ....................................... 419
32.8.4.40 scrollbar_width() [2/2] ....................................... 420
32.8.4.41 select() ............................................................ 420
32.8.4.42 select_only() ..................................................... 420
32.8.4.43 selection() ......................................................... 421
32.8.4.44 sort() ............................................................... 421
32.8.4.45 swapping() ....................................................... 421
32.8.4.46 textfont() ........................................................ 421

32.8.5 Member Data Documentation ....................................... 421
32.8.5.1 hscrollbar .......................................................... 421
32.8.5.2 scrollbar ........................................................... 422

32.9 Fl_Button Class Reference ........................................... 422
32.9.1 Detailed Description ................................................ 423
32.9.2 Constructor & Destructor Documentation .......................... 423
32.9.2.1 Fl_Button() ......................................................... 424

32.9.3 Member Function Documentation .................................. 424
32.9.3.1 clear() ............................................................. 424
32.9.3.2 down_box() [1/2] ............................................... 424
32.9.3.3 down_box() [2/2] ............................................... 424
32.9.3.4 draw() ............................................................. 425
32.9.3.5 handle() ........................................................... 425
32.9.3.6 set() ............................................................... 425
32.14.2.13 item_width() .................................................. 439
32.14.2.14 nitems() .......................................................... 439
32.14.2.15 remove() .......................................................... 439

32.15 Fl_Check_Button Class Reference .................................. 439
  32.15.1 Detailed Description ............................................ 440
  32.15.2 Constructor & Destructor Documentation ....................... 440
  32.15.2.1 Fl_Check_Button() ............................................. 440

32.16 Fl_Choice Class Reference ........................................ 440
  32.16.1 Detailed Description ............................................ 441
  32.16.2 Constructor & Destructor Documentation ....................... 442
     32.16.2.1 Fl_Choice() ................................................ 442
  32.16.3 Member Function Documentation ................................ 443
     32.16.3.1 draw() ................................................. 443
     32.16.3.2 handle() ............................................. 443
     32.16.3.3 value() [1/3] .......................................... 443
     32.16.3.4 value() [2/3] .......................................... 443
     32.16.3.5 value() [3/3] .......................................... 444

32.17 Fl_Clock Class Reference .......................................... 444
  32.17.1 Detailed Description ............................................ 445
  32.17.2 Constructor & Destructor Documentation ....................... 445
     32.17.2.1 Fl_Clock() [1/2] ....................................... 445
     32.17.2.2 Fl_Clock() [2/2] ....................................... 446
  32.17.3 Member Function Documentation ................................ 446
     32.17.3.1 handle() ............................................. 446

32.18 Fl_Clock_Output Class Reference ................................ 447
  32.18.1 Detailed Description ............................................ 447
  32.18.2 Constructor & Destructor Documentation ....................... 448
     32.18.2.1 Fl_Clock_Output() ..................................... 448
  32.18.3 Member Function Documentation ................................ 448
     32.18.3.1 draw() ............................................. 449
     32.18.3.2 hour() ............................................. 449
     32.18.3.3 minute() ........................................... 449
     32.18.3.4 second() .......................................... 449
     32.18.3.5 shadow() [1/2] ....................................... 449
     32.18.3.6 shadow() [2/2] ....................................... 450
     32.18.3.7 value() [1/3] .......................................... 450
     32.18.3.8 value() [2/3] .......................................... 450
     32.18.3.9 value() [3/3] .......................................... 450

32.19 Fl_Color_Chooser Class Reference ................................ 451
  32.19.1 Detailed Description ............................................ 452
  32.19.2 Constructor & Destructor Documentation ....................... 452
     32.19.2.1 Fl_Color_Chooser() ..................................... 452
32.23.3 Member Function Documentation .......................................................... 465
  32.23.3.1 angle1() ................................................................. 465
  32.23.3.2 draw() ................................................................. 465
  32.23.3.3 handle() ............................................................... 465
32.24 Fl_Display_Device Class Reference ......................................................... 465
  32.24.1 Detailed Description ........................................................................... 466
32.25 Fl_Double_Window Class Reference .......................................................... 466
  32.25.1 Detailed Description ........................................................................... 467
  32.25.2 Constructor & Destructor Documentation ............................................. 467
    32.25.2.1 ~Fl_Double_Window() ....................................................... 467
  32.25.3 Member Function Documentation ....................................................... 467
    32.25.3.1 hide() .......................................................................... 467
    32.25.3.2 resize() ...................................................................... 467
    32.25.3.3 show() ........................................................................ 468
32.26 Fl_End Class Reference ............................................................................. 468
  32.26.1 Detailed Description ........................................................................... 468
32.27 Fl_EPS_File_Surface Class Reference ....................................................... 469
  32.27.1 Detailed Description ........................................................................... 469
  32.27.2 Constructor & Destructor Documentation ............................................. 470
    32.27.2.1 Fl_EPS_File_Surface() ....................................................... 470
    32.27.2.2 ~Fl_EPS_File_Surface() ................................................... 470
  32.27.3 Member Function Documentation ....................................................... 470
    32.27.3.1 close() .......................................................................... 470
    32.27.3.2 origin() [1/2] .................................................................. 470
    32.27.3.3 origin() [2/2] .................................................................. 471
    32.27.3.4 printable_rect() .............................................................. 471
    32.27.3.5 translate() ..................................................................... 471
32.28 Fl_File_Browser Class Reference ............................................................. 471
  32.28.1 Detailed Description ........................................................................... 472
  32.28.2 Constructor & Destructor Documentation ............................................. 472
    32.28.2.1 Fl_File_Browser() ........................................................... 473
  32.28.3 Member Function Documentation ....................................................... 473
    32.28.3.1 errmsg() [1/2] .................................................................. 473
    32.28.3.2 errmsg() [2/2] .................................................................. 473
    32.28.3.3 filetype() [1/2] .................................................................. 473
    32.28.3.4 filetype() [2/2] .................................................................. 473
    32.28.3.5 iconsize() [1/2] ................................................................. 473
    32.28.3.6 iconsize() [2/2] .................................................................. 474
    32.28.3.7 load() ............................................................................. 474
32.29 Fl_File_Chooser Class Reference ............................................................. 474
  32.29.1 Detailed Description ........................................................................... 474
32.37.1 Detailed Description ................................................. 502
32.37.2 Constructor & Destructor Documentation .............................. 502
   32.37.2.1 Fl_FormsPixmap() ........................................... 502
32.37.3 Member Function Documentation .................................... 503
   32.37.3.1 draw() ..................................................... 503
   32.37.3.2 Pixmap() ..................................................... 503
   32.37.3.3 set() ....................................................... 503
32.38 Fl_FormsText Class Reference ......................................... 504
   32.38.1 Member Function Documentation .................................... 504
   32.38.1.1 draw() ..................................................... 504
32.39 Fl_Free Class Reference .................................................. 504
   32.39.1 Detailed Description ............................................. 505
   32.39.2 Constructor & Destructor Documentation .............................. 505
       32.39.2.1 Fl_Free() .................................................. 505
   32.39.3 Member Function Documentation .................................... 506
       32.39.3.1 draw() ..................................................... 506
       32.39.3.2 handle() .................................................. 506
32.40 Fl_GIF_Image Class Reference ........................................ 506
   32.40.1 Detailed Description ............................................. 507
   32.40.2 Constructor & Destructor Documentation .............................. 507
       32.40.2.1 Fl_GIF_Image() [1/3] ....................................... 507
       32.40.2.2 Fl_GIF_Image() [2/3] ....................................... 507
       32.40.2.3 Fl_GIF_Image() [3/3] ....................................... 508
32.41 Fl_Gl_Choice Class Reference .......................................... 509
32.42 Fl_Gl_Window Class Reference ......................................... 509
   32.42.1 Detailed Description ............................................. 511
   32.42.2 Constructor & Destructor Documentation .............................. 511
       32.42.2.1 Fl_Gl_Window() [1/2] ....................................... 511
       32.42.2.2 Fl_Gl_Window() [2/2] ....................................... 512
   32.42.3 Member Function Documentation .................................... 512
       32.42.3.1 as_gl_window() .............................................. 512
       32.42.3.2 can_do() ................................................... 512
       32.42.3.3 can_do_overlay() .......................................... 512
       32.42.3.4 context() [1/2] .............................................. 512
       32.42.3.5 context() [2/2] .............................................. 513
       32.42.3.6 context_valid() ............................................ 513
       32.42.3.7 draw() ..................................................... 513
       32.42.3.8 draw_begin() ............................................... 514
       32.42.3.9 draw_end() .................................................. 514
       32.42.3.10 make_current() ............................................. 514
       32.42.3.11 make_overlay_current() .................................... 514
       32.42.3.12 mode() [1/3] ............................................... 514
32.49.3.15 end() ......................................................... 528
32.49.3.16 find() ...................................................... 528
32.49.3.17 focus() ..................................................... 528
32.49.3.18 handle() ................................................... 528
32.49.3.19 init_sizes() ................................................. 529
32.49.3.20 insert() [1/2] ............................................... 529
32.49.3.21 insert() [2/2] ............................................... 529
32.49.3.22 remove() [1/3] .............................................. 529
32.49.3.23 remove() [2/3] .............................................. 529
32.49.3.24 remove() [3/3] .............................................. 530
32.49.3.25 resizable() [1/3] .......................................... 530
32.49.3.26 resizable() [2/3] .......................................... 530
32.49.3.27 resizable() [3/3] .......................................... 530
32.49.3.28 resize() .................................................... 531
32.49.3.29 sizes() .................................................... 531
32.49.3.30 update_child() ........................................... 532

32.50 Fl_Help_Block Struct Reference ........................................ 532
32.51 Fl_Help_Dialog Class Reference ....................................... 533
  32.51.1 Detailed Description ........................................ 534
  32.51.2 Member Function Documentation ................................ 534
    32.51.2.1 load() ................................................ 534
    32.51.2.2 show() ................................................. 534
    32.51.2.3 textsize() .............................................. 534
    32.51.2.4 value() [1/2] .......................................... 535
    32.51.2.5 value() [2/2] .......................................... 535

32.52 Fl_Help_Font_Stack Struct Reference ................................ 535
32.53 Fl_Help_Font_Style Struct Reference ................................ 535
  32.53.1 Detailed Description ........................................ 536
32.54 Fl_Help_Link Struct Reference ........................................ 536
  32.54.1 Detailed Description ........................................ 536
32.55 Fl_Help_Target Struct Reference ..................................... 536
  32.55.1 Detailed Description ........................................ 537
32.56 Fl_Help_View Class Reference ........................................ 537
  32.56.1 Detailed Description ........................................ 538
  32.56.2 Constructor & Destructor Documentation ......................... 540
    32.56.2.1 ~Fl_Help_View() ....................................... 540
  32.56.3 Member Function Documentation ................................ 540
    32.56.3.1 find() ................................................ 540
    32.56.3.2 leftline() ............................................ 541
    32.56.3.3 link() ................................................ 541
    32.56.3.4 load() ................................................ 541
    32.56.3.5 scrollbar_size() [1/2] ................................ 542
32.56.3.6 scrollbar_size() [2/2] .................................................. 542
32.56.3.7 topline() [1/2] .................................................. 542
32.56.3.8 topline() [2/2] .................................................. 544
32.56.3.9 value() ............................................................. 544

32.57 Fl_Hold_Browser Class Reference ..................................... 544
32.57.1 Detailed Description .................................................. 545
32.57.2 Constructor & Destructor Documentation .............................. 545
32.57.2.1 Fl_Hold_Browser() ............................................... 545

32.58 Fl_Hor_Fill_Slider Class Reference .................................... 545
32.59 Fl_Hor_Nice_Slider Class Reference .................................... 546
32.60 Fl_Hor_Slider Class Reference ......................................... 546
32.60.1 Detailed Description .................................................. 547
32.61 Fl_Hor_Value_Slider Class Reference ................................... 547

32.62 Fl_Image Class Reference ................................................ 547
32.62.1 Detailed Description .................................................. 550
32.62.2 Constructor & Destructor Documentation .............................. 550
32.62.2.1 Fl_Image() ........................................................ 550
32.62.3 Member Function Documentation ...................................... 550
32.62.3.1 as_shared_image() ................................................. 550
32.62.3.2 color_average() .................................................. 551
32.62.3.3 copy() [1/2] ...................................................... 551
32.62.3.4 copy() [2/2] ...................................................... 551
32.62.3.5 count() ............................................................ 552
32.62.3.6 d() ................................................................. 552
32.62.3.7 data() [1/2] ...................................................... 552
32.62.3.8 data() [2/2] ...................................................... 553
32.62.3.9 desaturate() ........................................................ 553
32.62.3.10 draw() [1/2] ...................................................... 553
32.62.3.11 draw() [2/2] ...................................................... 553
32.62.3.12 draw_empty() ................................................... 553
32.62.3.13 draw_scaled() ................................................... 554
32.62.3.14 fail() ............................................................. 554
32.62.3.15 h() [1/2] .......................................................... 554
32.62.3.16 h() [2/2] .......................................................... 555
32.62.3.17 inactive() ........................................................ 555
32.62.3.18 label() [1/2] ...................................................... 555
32.62.3.19 label() [2/2] ...................................................... 555
32.62.3.20 ld() [1/2] .......................................................... 555
32.62.3.21 ld() [2/2] .......................................................... 555
32.62.3.22 release() .......................................................... 556
32.62.3.23 RGB_scaling() ................................................... 556
32.62.3.24 scale() ............................................................. 556
32.74.1 print ................................................................. 596

32.75 Fl_Menu_ Class Reference .............................................. 596

32.75.1 Detailed Description .................................................. 599

32.75.2 Constructor & Destructor Documentation .......................... 599

32.75.2.1 Fl_Menu_() ......................................................... 599

32.75.3 Member Function Documentation .................................. 599

32.75.3.1 add() [1/2] ....................................................... 599

32.75.3.2 add() [2/2] ....................................................... 600

32.75.3.3 clear() .......................................................... 602

32.75.3.4 clear_submenu() .................................................. 602

32.75.3.5 copy() .......................................................... 603

32.75.3.6 down_box() ....................................................... 603

32.75.3.7 find_index() [1/3] .................................................. 603

32.75.3.8 find_index() [2/3] .................................................. 603

32.75.3.9 find_index() [3/3] .................................................. 604

32.75.3.10 find_item() [1/2] ............................................... 604

32.75.3.11 find_item() [2/2] ............................................... 605

32.75.3.12 global() ......................................................... 605

32.75.3.13 insert() ........................................................ 605

32.75.3.14 item_pathname() ............................................... 606

32.75.3.15 menu() [1/2] ...................................................... 607

32.75.3.16 menu() [2/2] ...................................................... 607

32.75.3.17 menu_end() ...................................................... 607

32.75.3.18 mode() [1/2] ...................................................... 608

32.75.3.19 mode() [2/2] ...................................................... 608

32.75.3.20 mvalue() ......................................................... 608

32.75.3.21 picked() ......................................................... 608

32.75.3.22 remove() ........................................................ 608

32.75.3.23 replace() ......................................................... 609

32.75.3.24 size() .......................................................... 609

32.75.3.25 test_shortcut() ................................................... 609

32.75.3.26 text() [1/2] ....................................................... 609

32.75.3.27 text() [2/2] ....................................................... 609

32.75.3.28 textcolor() ....................................................... 609

32.75.3.29 textfont() [1/2] .................................................. 610

32.75.3.30 textfont() [2/2] .................................................. 610

32.75.3.31 textsize() [1/2] .................................................... 610

32.75.3.32 textsize() [2/2] .................................................... 610

32.75.3.33 value() [1/3] ...................................................... 610

32.75.3.34 value() [2/3] ...................................................... 610

32.75.3.35 value() [3/3] ...................................................... 610

32.76 Fl_Menu_Bar Class Reference .......................................... 611
32.78.2.23 labelfont() [1/2] .................................................. 625
32.78.2.24 labelfont() [2/2] .................................................. 625
32.78.2.25 labeltype() [1/2] .................................................. 625
32.78.2.26 labeltype() [2/2] .................................................. 625
32.78.2.27 measure() .................................................. 625
32.78.2.28 next() [1/2] .................................................. 625
32.78.2.29 next() [2/2] .................................................. 625
32.78.2.30 popup() .................................................. 626
32.78.2.31 pulldown() .................................................. 626
32.78.2.32 radio() .................................................. 626
32.78.2.33 set() .................................................. 626
32.78.2.34 setonly() .................................................. 627
32.78.2.35 shortcut() .................................................. 627
32.78.2.36 size() .................................................. 627
32.78.2.37 submenu() .................................................. 627
32.78.2.38 test_shortcut() .................................................. 627
32.78.2.39 uncheck() .................................................. 628
32.78.2.40 value() .................................................. 628
32.79 Fl_Menu_Window Class Reference .................................................. 628
32.79.1 Detailed Description .................................................. 629
32.80 Fl_Multi_Browser Class Reference .................................................. 629
32.80.1 Detailed Description .................................................. 629
32.80.2 Constructor & Destructor Documentation .................................................. 629
32.80.2.1 Fl_Multi_Browser() .................................................. 630
32.81 Fl_Multi_Label Struct Reference .................................................. 630
32.81.1 Detailed Description .................................................. 630
32.81.2 Member Data Documentation .................................................. 631
32.81.2.1 labela .................................................. 631
32.81.2.2 labelb .................................................. 631
32.81.2.3 typea .................................................. 631
32.81.2.4 typeb .................................................. 631
32.82 Fl_Multiline_Input Class Reference .................................................. 631
32.82.1 Detailed Description .................................................. 632
32.82.2 Constructor & Destructor Documentation .................................................. 632
32.82.2.1 Fl_Multiline_Input() .................................................. 632
32.83 Fl_Multiline_Output Class Reference .................................................. 632
32.83.1 Detailed Description .................................................. 633
32.83.2 Constructor & Destructor Documentation .................................................. 633
32.83.2.1 Fl_Multiline_Output() .................................................. 633
32.84 Fl_Native_File_Chooser Class Reference .................................................. 633
32.84.1 Detailed Description .................................................. 635
32.84.2 Member Enumeration Documentation .................................................. 636
32.84.1 Option .................................................. 636
32.84.2 Type .................................................. 636

32.84.3 Constructor & Destructor Documentation .................. 637
32.84.3.1 Fl_Native_File_Chooser() .................................. 637
32.84.3.2 ~Fl_Native_File_Chooser() .................................. 637

32.84.4 Member Function Documentation ................................ 637
32.84.4.1 count() ........................................... 637
32.84.4.2 directory() .......................................... 637
32.84.4.3 errmsg() ........................................... 637
32.84.4.4 filename() [1/2] .................................... 637
32.84.4.5 filename() [2/2] .................................... 638
32.84.4.6 filter() [1/2] ....................................... 638
32.84.4.7 filter() [2/2] ....................................... 638
32.84.4.8 filter_value() [1/2] ................................ 638
32.84.4.9 filter_value() [2/2] ................................ 638
32.84.4.10 options() .......................................... 638
32.84.4.11 preset_file() ...................................... 639
32.84.4.12 show() ............................................ 639
32.84.4.13 title() [1/2] ....................................... 639
32.84.4.14 title() [2/2] ....................................... 639

32.85 Fl_Nice_Slider Class Reference ............................... 639

32.86 Fl_Output Class Reference .................................. 640
32.86.1 Detailed Description .................................... 640
32.86.2 Constructor & Destructor Documentation .................. 641
32.86.2.1 Fl_Output() ........................................ 641

32.87 Fl_Overlay_Window Class Reference ........................ 641
32.87.1 Detailed Description .................................... 642
32.87.2 Constructor & Destructor Documentation .................. 643
32.87.2.1 Fl_Overlay_Window() ................................ 643
32.87.3 Member Function Documentation .......................... 643
32.87.3.1 draw_overlay() ..................................... 643
32.87.3.2 hide() ............................................. 643
32.87.3.3 redraw_overlay() ................................... 643
32.87.3.4 resize() ............................................ 643
32.87.3.5 show() ............................................. 644

32.88 Fl_Pack Class Reference .................................... 644
32.88.1 Detailed Description .................................... 645
32.88.2 Constructor & Destructor Documentation .................. 645
32.88.2.1 Fl_Pack() ........................................ 645
32.88.3 Member Function Documentation .......................... 646
32.88.3.1 draw() ............................................. 646
32.88.3.2 horizontal() ....................................... 646
32.97.4.2 Fl_Preferences() [2/7] ........................................ 675
32.97.4.3 Fl_Preferences() [3/7] ........................................ 676
32.97.4.4 Fl_Preferences() [4/7] ........................................ 676
32.97.4.5 Fl_Preferences() [5/7] ........................................ 676
32.97.4.6 Fl_Preferences() [6/7] ........................................ 677
32.97.4.7 Fl_Preferences() [7/7] ........................................ 677
32.97.4.8 ~Fl_Preferences() ............................................. 677

32.97.5 Member Function Documentation .................................. 677
32.97.5.1 delete_entry() ................................................. 677
32.97.5.2 delete_group() .................................................. 678
32.97.5.3 dirty() ........................................................... 678
32.97.5.4 entries() .......................................................... 678
32.97.5.5 entry() ............................................................ 678
32.97.5.6 entry_exists() .................................................... 679
32.97.5.7 file_access() [1/2] .............................................. 679
32.97.5.8 file_access() [2/2] .............................................. 679
32.97.5.9 filename() [1/2] .................................................. 679
32.97.5.10 filename() [2/2] ................................................ 680
32.97.5.11 flush() .......................................................... 680
32.97.5.12 get() [1/8] ....................................................... 681
32.97.5.13 get() [2/8] ....................................................... 681
32.97.5.14 get() [3/8] ....................................................... 681
32.97.5.15 get() [4/8] ....................................................... 682
32.97.5.16 get() [5/8] ....................................................... 682
32.97.5.17 get() [6/8] ....................................................... 683
32.97.5.18 get() [7/8] ....................................................... 683
32.97.5.19 get() [8/8] ....................................................... 684
32.97.5.20 get_userdata_path() .......................................... 684
32.97.5.21 group() .......................................................... 685
32.97.5.22 group_exists() ................................................ 685
32.97.5.23 groups() ........................................................ 685
32.97.5.24 new_UUID() .................................................... 686
32.97.5.25 set() [1/7] ....................................................... 686
32.97.5.26 set() [2/7] ....................................................... 686
32.97.5.27 set() [3/7] ....................................................... 686
32.97.5.28 set() [4/7] ....................................................... 687
32.97.5.29 set() [5/7] ....................................................... 687
32.97.5.30 set() [6/7] ....................................................... 688
32.97.5.31 set() [7/7] ....................................................... 688
32.97.5.32 size() ........................................................... 688

32.97.6 Member Data Documentation ..................................... 689
32.97.6.1 CORE_READ_OK ............................................... 689
32.110.2 Constructor & Destructor Documentation ........................................ 712
32.110.2.1 Fl_Round_Button() ................................................................. 712

32.111 Fl_Round_Clock Class Reference ......................................................... 713
32.111.1 Detailed Description ......................................................................... 713
32.111.2 Constructor & Destructor Documentation ........................................... 713
32.111.2.1 Fl_Round_Clock() ....................................................................... 713

32.112 Fl_Scroll Class Reference ................................................................. 714
32.112.1 Detailed Description ......................................................................... 715
32.112.2 Constructor & Destructor Documentation ........................................... 716
32.112.2.1 Fl_Scroll() ................................................................................. 716
32.112.3 Member Function Documentation ...................................................... 716
32.112.3.1 bbox() ...................................................................................... 717
32.112.3.2 delete_child() ............................................................................ 717
32.112.3.3 draw() ....................................................................................... 717
32.112.3.4 handle() ..................................................................................... 718
32.112.3.5 recalc_scrollbackers() ................................................................. 718
32.112.3.6 resize() ...................................................................................... 718
32.112.3.7 scroll_to() ................................................................................. 719
32.112.3.8 scrollbar_size() [1/2] ................................................................ 719
32.112.3.9 scrollbar_size() [2/2] ................................................................ 719
32.112.3.10 xposition() .............................................................................. 720
32.112.3.11 yposition() .............................................................................. 720

32.113 Fl_Scrollbar Class Reference ............................................................. 720
32.113.1 Detailed Description ......................................................................... 721
32.113.2 Constructor & Destructor Documentation ........................................... 721
32.113.2.1 Fl_Scrollbar() ........................................................................... 721
32.113.3 Member Function Documentation ...................................................... 722
32.113.3.1 draw() ...................................................................................... 722
32.113.3.2 handle() ..................................................................................... 722
32.113.3.3 linesize() ................................................................................... 722
32.113.3.4 value() [1/3] .............................................................................. 722
32.113.3.5 value() [2/3] .............................................................................. 723
32.113.3.6 value() [3/3] .............................................................................. 723

32.114 Fl_Scroll::Fl_Scrollbar_Data Struct Reference ................................. 723
32.114.1 Detailed Description ......................................................................... 724

32.115 Fl_Secret_Input Class Reference ......................................................... 724
32.115.1 Detailed Description ......................................................................... 724
32.115.2 Constructor & Destructor Documentation ........................................... 724
32.115.2.1 Fl_Secret_Input() ....................................................................... 724
32.115.3 Member Function Documentation ...................................................... 725
32.115.3.1 handle() ..................................................................................... 725

32.116 Fl_Select_Browser Class Reference ...................................................... 725

Generated by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.129.4.17</td>
<td>draw()</td>
</tr>
<tr>
<td>32.129.4.18</td>
<td>draw_cell()</td>
</tr>
<tr>
<td>32.129.4.19</td>
<td>find_cell()</td>
</tr>
<tr>
<td>32.129.4.20</td>
<td>get_selection()</td>
</tr>
<tr>
<td>32.129.4.21</td>
<td>init_sizes()</td>
</tr>
<tr>
<td>32.129.4.22</td>
<td>insert()</td>
</tr>
<tr>
<td>32.129.4.23</td>
<td>is_interactive_resize()</td>
</tr>
<tr>
<td>32.129.4.24</td>
<td>is_selected()</td>
</tr>
<tr>
<td>32.129.4.25</td>
<td>move_cursor()</td>
</tr>
<tr>
<td>32.129.4.26</td>
<td>recalc_dimensions()</td>
</tr>
<tr>
<td>32.129.4.27</td>
<td>redraw_range()</td>
</tr>
<tr>
<td>32.129.4.28</td>
<td>resize()</td>
</tr>
<tr>
<td>32.129.4.29</td>
<td>row_col_clamp()</td>
</tr>
<tr>
<td>32.129.4.30</td>
<td>row_header()</td>
</tr>
<tr>
<td>32.129.4.31</td>
<td>row_height()</td>
</tr>
<tr>
<td>32.129.4.32</td>
<td>row_height_all()</td>
</tr>
<tr>
<td>32.129.4.33</td>
<td>row_resize()</td>
</tr>
<tr>
<td>32.129.4.34</td>
<td>row_resize_min()</td>
</tr>
<tr>
<td>32.129.4.35</td>
<td>scrollbar_size() [1/2]</td>
</tr>
<tr>
<td>32.129.4.36</td>
<td>scrollbar_size() [2/2]</td>
</tr>
<tr>
<td>32.129.4.37</td>
<td>set_selection()</td>
</tr>
<tr>
<td>32.129.4.38</td>
<td>tab_cell_nav() [1/2]</td>
</tr>
<tr>
<td>32.129.4.39</td>
<td>tab_cell_nav() [2/2]</td>
</tr>
<tr>
<td>32.129.4.40</td>
<td>table_box()</td>
</tr>
<tr>
<td>32.129.4.41</td>
<td>table_resized()</td>
</tr>
<tr>
<td>32.129.4.42</td>
<td>table_scrolled()</td>
</tr>
<tr>
<td>32.129.4.43</td>
<td>top_row() [1/2]</td>
</tr>
<tr>
<td>32.129.4.44</td>
<td>top_row() [2/2]</td>
</tr>
<tr>
<td>32.129.4.45</td>
<td>visible_cells()</td>
</tr>
<tr>
<td>32.129.4.46</td>
<td>when()</td>
</tr>
</tbody>
</table>

**32.130 Fl_Table_Row Class Reference**

**32.130.1 Detailed Description**

**32.130.2 Constructor & Destructor Documentation**

- 32.130.2.1 Fl_Table_Row()
- 32.130.2.2 ~Fl_Table_Row()

**32.130.3 Member Function Documentation**

- 32.130.3.1 clear()
- 32.130.3.2 row_selected()
- 32.130.3.3 select_all_rows()
- 32.130.3.4 select_row()
- 32.130.3.5 type()

**32.131 Fl_Tabs Class Reference**

Generated by Doxygen
32.131.1 Detailed Description ......................................................... 794
32.131.2 Constructor & Destructor Documentation ............................... 797
   32.131.2.1 Fl_Tabs() .......................................................... 797
32.131.3 Member Function Documentation ........................................ 798
   32.131.3.1 client_area() .................................................... 798
   32.131.3.2 draw() .......................................................... 798
   32.131.3.3 handle() .......................................................... 798
   32.131.3.4 push() [1/2] ..................................................... 799
   32.131.3.5 push() [2/2] ..................................................... 799
   32.131.3.6 tab_align() [1/2] .............................................. 799
   32.131.3.7 tab_align() [2/2] .............................................. 800
   32.131.3.8 value() [1/2] .................................................... 800
   32.131.3.9 value() [2/2] .................................................... 800
   32.131.3.10 which() ......................................................... 800
32.132 Fl_Text_Buffer Class Reference ........................................... 800
   32.132.1 Detailed Description ................................................ 805
   32.132.2 Constructor & Destructor Documentation ............................... 805
      32.132.2.1 Fl_Text_Buffer() ............................................ 805
   32.132.3 Member Function Documentation ........................................ 806
      32.132.3.1 add_modify_callback() ..................................... 806
      32.132.3.2 address() [1/2] ............................................. 806
      32.132.3.3 address() [2/2] ............................................. 806
      32.132.3.4 append() ..................................................... 806
      32.132.3.5 appendfile() ................................................. 807
      32.132.3.6 byte_at() ..................................................... 807
      32.132.3.7 char_at() ..................................................... 807
      32.132.3.8 copy() ......................................................... 807
      32.132.3.9 count_displayed_characters() ................................ 808
      32.132.3.10 count_lines() .............................................. 808
      32.132.3.11 findchar_backward() ...................................... 808
      32.132.3.12 findchar_forward() ...................................... 808
      32.132.3.13 highlight_text() .......................................... 809
      32.132.3.14 insert() ..................................................... 809
      32.132.3.15 insert_() ................................................... 809
      32.132.3.16 insertfile() ............................................... 809
      32.132.3.17 is_word_separator() ...................................... 810
      32.132.3.18 length() ..................................................... 810
      32.132.3.19 line_end() ................................................. 810
      32.132.3.20 line_start() ............................................... 810
      32.132.3.21 line_text() ................................................. 810
      32.132.3.22 loadfile() .................................................. 811
      32.132.3.23 next_char() ................................................ 811
lv

32.133.4.51 measure_vline() .................................................. 842
32.133.4.52 move_down() ....................................................... 842
32.133.4.53 move_left() ......................................................... 843
32.133.4.54 move_right() ....................................................... 843
32.133.4.55 move_up() .......................................................... 843
32.133.4.56 offset_line_starts() .............................................. 843
32.133.4.57 overstrike() .......................................................... 843
32.133.4.58 position_style() .................................................. 844
32.133.4.59 position_to_line() ............................................... 844
32.133.4.60 position_to_linecol() .......................................... 845
32.133.4.61 position_to_xy() .................................................. 845
32.133.4.62 redisplay_range() ................................................ 845
32.133.4.63 reset_absolute_top_line_number() .......................... 846
32.133.4.64 resize() ............................................................. 846
32.133.4.65 rewind_lines() .................................................... 846
32.133.4.66 scroll() ............................................................. 846
32.133.4.67 scroll_() ............................................................ 847
32.133.4.68 scroll_timer_cb() ................................................ 847
32.133.4.69 scrollbar_align() [1/2] ......................................... 847
32.133.4.70 scrollbar_align() [2/2] ......................................... 847
32.133.4.71 scrollbar_size() [1/2] .......................................... 847
32.133.4.72 scrollbar_size() [2/2] .......................................... 848
32.133.4.73 scrollbar_width() [1/2] ......................................... 848
32.133.4.74 scrollbar_width() [2/2] ......................................... 848
32.133.4.75 secondary_selection_color() [1/2] ......................... 848
32.133.4.76 secondary_selection_color() [2/2] ......................... 849
32.133.4.77 shortcut() [1/2] ..................................................... 849
32.133.4.78 shortcut() [2/2] ..................................................... 849
32.133.4.79 show_cursor() ....................................................... 849
32.133.4.80 show_insert_position() ........................................ 849
32.133.4.81 skip_lines() .......................................................... 850
32.133.4.82 spelling_underline_color() [1/2] .......................... 850
32.133.4.83 spelling_underline_color() [2/2] .......................... 850
32.133.4.84 string_width() ..................................................... 850
32.133.4.85 style_buffer() ...................................................... 851
32.133.4.86 textcolor() [1/2] .................................................. 851
32.133.4.87 textcolor() [2/2] .................................................. 851
32.133.4.88 textfont() [1/2] ..................................................... 851
32.133.4.89 textfont() [2/2] ..................................................... 851
32.133.4.90 textsize() [1/2] ...................................................... 852
32.133.4.91 textsize() [2/2] ...................................................... 852
32.133.4.92 update_h_scrollbar() ........................................... 852

Generated by Doxygen
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_line_starts()</td>
<td>852</td>
</tr>
<tr>
<td>update_v_scrollbar()</td>
<td>853</td>
</tr>
<tr>
<td>vline_length()</td>
<td>853</td>
</tr>
<tr>
<td>word_end()</td>
<td>853</td>
</tr>
<tr>
<td>word_start()</td>
<td>853</td>
</tr>
<tr>
<td>wrap_mode()</td>
<td>854</td>
</tr>
<tr>
<td>wrapUses_character()</td>
<td>854</td>
</tr>
<tr>
<td>wrapped_column()</td>
<td>855</td>
</tr>
<tr>
<td>wrapped_line_counter()</td>
<td>855</td>
</tr>
<tr>
<td>wrapped_row()</td>
<td>856</td>
</tr>
<tr>
<td>xy_to_position()</td>
<td>856</td>
</tr>
<tr>
<td>xy_to_rowcol()</td>
<td>857</td>
</tr>
</tbody>
</table>

32.134 Fl_Text_Editor Class Reference

32.134.1 Detailed Description

32.134.2 Member Function Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_key_binding()</td>
<td>860</td>
</tr>
<tr>
<td>insert_mode()</td>
<td>860</td>
</tr>
<tr>
<td>insert_mode() [1/2]</td>
<td>860</td>
</tr>
<tr>
<td>insert_mode() [2/2]</td>
<td>860</td>
</tr>
<tr>
<td>kf_backspace()</td>
<td>860</td>
</tr>
<tr>
<td>kf_c_s_move()</td>
<td>860</td>
</tr>
<tr>
<td>kf_copy()</td>
<td>861</td>
</tr>
<tr>
<td>kf_ctrl_move()</td>
<td>861</td>
</tr>
<tr>
<td>kf_cut()</td>
<td>861</td>
</tr>
<tr>
<td>kf_default()</td>
<td>861</td>
</tr>
<tr>
<td>kf_delete()</td>
<td>861</td>
</tr>
<tr>
<td>kf_down()</td>
<td>861</td>
</tr>
<tr>
<td>kf_end()</td>
<td>861</td>
</tr>
<tr>
<td>kf_enter()</td>
<td>862</td>
</tr>
<tr>
<td>kf_home()</td>
<td>862</td>
</tr>
<tr>
<td>kf_ignore()</td>
<td>862</td>
</tr>
<tr>
<td>kf_insert()</td>
<td>862</td>
</tr>
<tr>
<td>kf_left()</td>
<td>862</td>
</tr>
<tr>
<td>kf_m_s_move()</td>
<td>862</td>
</tr>
<tr>
<td>kf_meta_move()</td>
<td>862</td>
</tr>
<tr>
<td>kf_move()</td>
<td>863</td>
</tr>
<tr>
<td>kf_page_down()</td>
<td>863</td>
</tr>
<tr>
<td>kf_page_up()</td>
<td>863</td>
</tr>
<tr>
<td>kf_paste()</td>
<td>863</td>
</tr>
<tr>
<td>kf_right()</td>
<td>863</td>
</tr>
<tr>
<td>kf_select_all()</td>
<td>864</td>
</tr>
<tr>
<td>kf_shift_move()</td>
<td>864</td>
</tr>
</tbody>
</table>

Generated by Doxygen
32.142 Fl_Tree Class Reference ........................................ 889
  32.142.1 Detailed Description ...................................... 896
  32.142.2 Member Function Documentation .......................... 900
    32.142.2.1 add() [1/2] ........................................ 900
    32.142.2.2 add() [2/2] ........................................ 901
    32.142.2.3 calc_dimensions() .................................. 901
    32.142.2.4 calc_tree() ........................................ 901
    32.142.2.5 callback_item() [1/2] .............................. 902
    32.142.2.6 callback_item() [2/2] .............................. 902
    32.142.2.7 callback_reason() [1/2] ........................... 902
    32.142.2.8 callback_reason() [2/2] ........................... 902
    32.142.2.9 clear() ............................................ 902
    32.142.2.10 clear_children() ................................ 903
    32.142.2.11 close() [1/2] ..................................... 903
    32.142.2.12 close() [2/2] ..................................... 903
    32.142.2.13 closeicon() [1/2] ................................ 904
    32.142.2.14 closeicon() [2/2] ................................ 904
    32.142.2.15 connectorstyle() ................................ 904
    32.142.2.16 deselect() [1/2] ................................ 904
    32.142.2.17 deselect() [2/2] ................................ 905
    32.142.2.18 deselect_all() ................................... 905
    32.142.2.19 display() .......................................... 906
    32.142.2.20 displayed() ....................................... 906
    32.142.2.21 extend_selection() ................................ 906
    32.142.2.22 extend_selection_dir() ........................... 907
    32.142.2.23 find_clicked() .................................... 907
    32.142.2.24 find_item() ...................................... 908
    32.142.2.25 first() ............................................ 908
    32.142.2.26 first_selected_item() ............................. 909
    32.142.2.27 first_visible() .................................. 909
    32.142.2.28 first_visible_item() .............................. 909
    32.142.2.29 get_selected_items() .............................. 909
32.142.2.30 handle() .................................................. 910
32.142.2.31 hposition() [1/2] ........................................... 910
32.142.2.32 hposition() [2/2] ........................................... 910
32.142.2.33 insert() .................................................... 911
32.142.2.34 insert_above() .............................................. 911
32.142.2.35 is_close() [1/2] ............................................. 912
32.142.2.36 is_close() [2/2] ............................................. 912
32.142.2.37 is_hscroll_visible() ....................................... 912
32.142.2.38 is_open() [1/2] ............................................. 913
32.142.2.39 is_open() [2/2] ............................................. 913
32.142.2.40 is_scrollbar() .............................................. 913
32.142.2.41 is_selected() [1/2] ......................................... 914
32.142.2.42 is_selected() [2/2] ......................................... 914
32.142.2.43 is_vscroll_visible() ....................................... 914
32.142.2.44 item_clicked() [1/2] ....................................... 915
32.142.2.45 item_clicked() [2/2] ....................................... 915
32.142.2.46 item_draw_mode() [1/3] ................................... 915
32.142.2.47 item_draw_mode() [2/3] ................................... 915
32.142.2.48 item_draw_mode() [3/3] ................................... 915
32.142.2.49 item_labelbgcolor() [1/2] ............................... 916
32.142.2.50 item_labelbgcolor() [2/2] ............................... 916
32.142.2.51 item_labelfgcolor() ....................................... 916
32.142.2.52 item_labelfont() ......................................... 916
32.142.2.53 item_labelsize() ......................................... 916
32.142.2.54 item_pathname() .......................................... 916
32.142.2.55 item_reselect_mode() [1/2] ............................. 917
32.142.2.56 item_reselect_mode() [2/2] ............................. 917
32.142.2.57 last() ..................................................... 917
32.142.2.58 last_selected_item() ...................................... 917
32.142.2.59 last_visible() ............................................ 918
32.142.2.60 last_visible_item() ....................................... 918
32.142.2.61 load() ................................................... 918
32.142.2.62 next() .................................................... 918
32.142.2.63 next_item() ............................................... 919
32.142.2.64 next_selected_item() .................................... 920
32.142.2.65 next_visible_item() ...................................... 920
32.142.2.66 open() [1/2] .............................................. 921
32.142.2.67 open() [2/2] .............................................. 922
32.142.2.68 open_toggle() ............................................. 922
32.142.2.69 openicon() [1/2] .......................................... 923
32.142.2.70 openicon() [2/2] .......................................... 923
32.142.2.71 prev() .................................................... 923
32.143.3.48 prev_visible() .......................................................... 949
32.143.3.49 recalc_tree() .......................................................... 949
32.143.3.50 remove_child() [1/2] .................................................. 949
32.143.3.51 remove_child() [2/2] .................................................. 949
32.143.3.52 reparent() .............................................................. 949
32.143.3.53 replace() ............................................................... 950
32.143.3.54 replace_child() ...................................................... 950
32.143.3.55 select() ................................................................. 951
32.143.3.56 select_all() ........................................................... 951
32.143.3.57 show_self() ........................................................... 951
32.143.3.58 show_widgets() ...................................................... 951
32.143.3.59 swap_children() [1/2] ............................................. 951
32.143.3.60 swap_children() [2/2] ............................................. 951
32.143.3.61 tree() [1/2] ............................................................ 952
32.143.3.62 tree() [2/2] ............................................................ 952
32.143.3.63 update_prev_next() .............................................. 952
32.143.3.64 userdeicon() [1/2] ................................................. 952
32.143.3.65 userdeicon() [2/2] ................................................. 952
32.143.3.66 usericon() ............................................................. 953
32.143.3.67 visible_r() ........................................................... 953

32.144 Fl_Tree_Item_Array Class Reference ........................................ 953
32.144.1 Detailed Description .................................................. 954
32.144.2 Constructor & Destructor Documentation ............................. 954
32.144.2.1 Fl_Tree_Item_Array() ........................................... 954
32.144.3 Member Function Documentation .................................... 954
32.144.3.1 add() ................................................................. 955
32.144.3.2 clear() ............................................................... 955
32.144.3.3 deparent() .......................................................... 955
32.144.3.4 insert() ............................................................. 955
32.144.3.5 manage_item_destroy() ........................................ 955
32.144.3.6 move() ............................................................... 955
32.144.3.7 remove() [1/2] ...................................................... 956
32.144.3.8 remove() [2/2] ...................................................... 956
32.144.3.9 reparent() ........................................................... 956
32.144.3.10 replace() ........................................................... 956

32.145 Fl_Tree_Prefs Class Reference .......................................... 956
32.145.1 Detailed Description .................................................. 959
32.145.2 Member Function Documentation ................................... 959
32.145.2.1 closedeicon() ..................................................... 959
32.145.2.2 closeicon() ........................................................ 960
32.145.2.3 item_draw_mode() ................................................ 961
32.145.2.4 item_labelbgcolor() [1/2] .................................... 961

Generated by Doxygen
32.145.2.5 item_labelbgcolor() [2/2] ......................................................... 961
32.145.2.6 marginbottom() ................................................................. 961
32.145.2.7 opendeicon() .................................................................. 961
32.145.2.8 openicon() [1/2] ................................................................. 961
32.145.2.9 openicon() [2/2] ................................................................. 961
32.145.2.10 selectmode() ................................................................. 962
32.145.2.11 showcollapse() .............................................................. 962
32.145.2.12 showroot() ................................................................. 962
32.145.2.13 sortorder() ................................................................. 962
32.145.2.14 userdeicon() ................................................................. 962

32.146 Fl_Valuator Class Reference ......................................................... 962
32.146.1 Detailed Description ............................................................... 964
32.146.2 Constructor & Destructor Documentation .............................. 965
32.146.2.1 Fl_Valuator() ................................................................. 965
32.146.3 Member Function Documentation .......................................... 965
32.146.3.1 format() ............................................................... 965
32.146.3.2 increment() ............................................................... 965
32.146.3.3 maximum() [1/2] ........................................................... 965
32.146.3.4 maximum() [2/2] ........................................................... 965
32.146.3.5 minimum() [1/2] ........................................................... 965
32.146.3.6 minimum() [2/2] ........................................................... 965
32.146.3.7 precision() ............................................................... 966
32.146.3.8 range() ............................................................... 966
32.146.3.9 round() ............................................................... 966
32.146.3.10 step() ............................................................... 966
32.146.3.11 value() [1/2] ............................................................... 966
32.146.3.12 value() [2/2] ............................................................... 967

32.147 Fl_Value.Input Class Reference ..................................................... 967
32.147.1 Detailed Description .............................................................. 968
32.147.2 Constructor & Destructor Documentation ................................ 968
32.147.2.1 Fl_Value.Input() ............................................................ 968
32.147.3 Member Function Documentation .......................................... 969
32.147.3.1 cursor_color() [1/2] ...................................................... 969
32.147.3.2 cursor_color() [2/2] ...................................................... 969
32.147.3.3 draw() .................................................................. 969
32.147.3.4 handle() ............................................................... 969
32.147.3.5 resize() ............................................................... 969
32.147.3.6 shortcut() [1/2] ............................................................ 970
32.147.3.7 shortcut() [2/2] ............................................................ 970
32.147.3.8 soft() ............................................................... 970
32.147.3.9 textcolor() ............................................................... 970
32.147.3.10 textfont() [1/2] ............................................................ 971
32.150.4.89 shortcut_label() [1/2] ........................................... 1008
32.150.4.90 shortcut_label() [2/2] ........................................... 1008
32.150.4.91 show() ............................................................... 1008
32.150.4.92 size() ............................................................... 1009
32.150.4.93 take_focus() ....................................................... 1009
32.150.4.94 take_several() .................................................... 1009
32.150.4.95 test_menu() [1/2] ............................................... 1009
32.150.4.96 test_menu() [2/2] ............................................... 1010
32.150.4.97 tooltip() [1/2] .................................................... 1010
32.150.4.98 tooltip() [2/2] .................................................... 1010
32.150.4.99 top_window() ..................................................... 1011
32.150.4.100 top_window_offset() ........................................ 1011
32.150.4.101 type() [1/2] ...................................................... 1011
32.150.4.102 type() [2/2] ...................................................... 1011
32.150.4.103 user_data() [1/2] ............................................. 1011
32.150.4.104 user_data() [2/2] ............................................. 1012
32.150.4.105 visible() ......................................................... 1012
32.150.4.106 visible_focus() [1/2] ................................. 1012
32.150.4.107 visible_focus() [2/2] ................................. 1012
32.150.4.108 visible_r() ..................................................... 1013
32.150.4.109 w() [1/2] ......................................................... 1013
32.150.4.110 w() [2/2] ......................................................... 1013
32.150.4.111 when() [1/2] ..................................................... 1013
32.150.4.112 when() [2/2] ..................................................... 1013
32.150.4.113 window() ......................................................... 1014
32.150.4.114 x() [1/2] ......................................................... 1014
32.150.4.115 x() [2/2] ......................................................... 1014
32.150.4.116 y() [1/2] ......................................................... 1015
32.150.4.117 y() [2/2] ......................................................... 1015

32.151 Fl_Widget_Surface Class Reference ........................................... 1015
  32.151.1 Detailed Description .................................................. 1016
  32.151.2 Constructor & Destructor Documentation .......................... 1016
    32.151.2.1 Fl_Widget_Surface() ........................................ 1016
  32.151.3 Member Function Documentation .................................... 1016
    32.151.3.1 draw() ........................................................... 1016
    32.151.3.2 draw_decorated_window() ..................................... 1017
    32.151.3.3 origin() [1/2] ................................................. 1017
    32.151.3.4 origin() [2/2] ................................................. 1017
    32.151.3.5 print_window_part() ........................................... 1017
    32.151.3.6 printable_rect() ............................................. 1018
    32.151.3.7 translate() ..................................................... 1018

32.152 Fl_Widget_Tracker Class Reference ........................................... 1018

Generated by Doxygen
32.152.1 Detailed Description ................................................. 1019
32.152.2 Member Function Documentation .................................. 1019
  32.152.2.1 deleted() ......................................................... 1019
  32.152.2.2 exists() ........................................................ 1019
  32.152.2.3 widget() ......................................................... 1020
32.153 Fl_Window Class Reference .............................................. 1020
32.153.1 Detailed Description ................................................ 1024
32.153.2 Constructor & Destructor Documentation .......................... 1024
  32.153.2.1 Fl_Window() [1/2] ................................................ 1024
  32.153.2.2 Fl_Window() [2/2] ................................................ 1025
  32.153.2.3 ~Fl_Window() ..................................................... 1025
32.153.3 Member Function Documentation .................................... 1025
  32.153.3.1 as_window() ....................................................... 1025
  32.153.3.2 border() ........................................................ 1025
  32.153.3.3 clear_border() ................................................ 1025
  32.153.3.4 clear_modal_states() ......................................... 1026
  32.153.3.5 current() ........................................................ 1026
  32.153.3.6 cursor() [1/3] ................................................... 1026
  32.153.3.7 cursor() [2/3] ................................................... 1026
  32.153.3.8 cursor() [3/3] ................................................... 1027
  32.153.3.9 decorated_h() .................................................. 1027
  32.153.3.10 decorated_w() ............................................... 1027
  32.153.3.11 default_cursor() [1/2] ...................................... 1027
  32.153.3.12 default_cursor() [2/2] ...................................... 1027
  32.153.3.13 default_icon() ................................................. 1028
  32.153.3.14 default_icons() [1/2] ....................................... 1028
  32.153.3.15 default_icons() [2/2] ....................................... 1029
  32.153.3.16 default_size_range() ..................................... 1029
  32.153.3.17 default_xclass() [1/2] .................................... 1030
  32.153.3.18 default_xclass() [2/2] .................................... 1030
  32.153.3.19 draw() ......................................................... 1030
  32.153.3.20 force_position() [1/2] ..................................... 1031
  32.153.3.21 force_position() [2/2] ..................................... 1031
  32.153.3.22 free_icons() ................................................ 1031
  32.153.3.23 free_position() ............................................. 1031
  32.153.3.24 fullscreen() ................................................ 1031
  32.153.3.25 fullscreen_screens() ..................................... 1032
  32.153.3.26 handle() ...................................................... 1032
  32.153.3.27 hide() ........................................................ 1032
  32.153.3.28 hotspot() .................................................... 1033
  32.153.3.29 icon() [1/3] ................................................... 1033
  32.153.3.30 icon() [2/3] ................................................... 1033
33.1.5.5 fl_define_FL_ICON_LABEL() .......................................................... 1070
33.1.5.6 fl_define_FL_IMAGE_LABEL() ......................................................... 1070
33.1.5.7 fl_define_FL_MULTI_LABEL() .......................................................... 1070
33.1.5.8 fl_define_FL_SHADOW_LABEL() ......................................................... 1070
33.1.5.9 fl_down() ....................................................................................... 1071
33.1.5.10 fl_frame() ..................................................................................... 1071
33.1.5.11 fl_gray_ramp() ............................................................................. 1071

33.1.6 Variable Documentation .................................................................. 1071
33.1.6.1 FL_ALIGN_LEFT .............................................................................. 1071
33.1.6.2 FL_ALIGN_TOP ............................................................................... 1071
33.1.6.3 FL_NORMAL_SIZE ........................................................................... 1071

33.2 filename.H File Reference ................................................................. 1071
33.2.1 Detailed Description ........................................................................ 1072

33.3 Fl.cxx File Reference .......................................................................... 1072
33.3.1 Detailed Description ........................................................................ 1073
33.3.2 Function Documentation .................................................................. 1073
33.3.2.1 fl_disable_wayland() .................................................................... 1073
33.3.2.2 fl_find() ......................................................................................... 1074
33.3.2.3 fl_open_display() .......................................................................... 1074

33.4 Fl.H File Reference ............................................................................... 1074
33.4.1 Detailed Description ........................................................................ 1075
33.4.2 Function Documentation .................................................................. 1075
33.4.2.1 fl_disable_wayland() .................................................................... 1075

33.5 fl_arc.cxx File Reference ..................................................................... 1076
33.5.1 Detailed Description ........................................................................ 1076

33.6 fl_ask.cxx File Reference ..................................................................... 1076
33.6.1 Detailed Description ........................................................................ 1077

33.7 fl_ask.H File Reference ....................................................................... 1078
33.7.1 Detailed Description ........................................................................ 1079
33.7.2 Enumeration Type Documentation ................................................... 1079
33.7.2.1 Fl_Beep ......................................................................................... 1079
33.7.3 Function Documentation .................................................................. 1079
33.7.3.1 fl_message_position() .................................................................. 1080

33.8 fl_boxtype.cxx File Reference ............................................................. 1080
33.8.1 Detailed Description ........................................................................ 1081
33.8.2 Function Documentation .................................................................. 1081
33.8.2.1 fl_internal_boxtype() .................................................................... 1081
33.8.2.2 fl_rectbound() ................................................................................ 1081

33.9 fl_color.cxx File Reference ................................................................. 1081
33.9.1 Detailed Description ........................................................................ 1082
33.9.2 Variable Documentation .................................................................. 1082
33.9.2.1 fl_cmap ......................................................................................... 1082

Generated by Doxygen
### Index

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.49 x11.H File Reference</td>
<td>1116</td>
</tr>
<tr>
<td>33.49.1 Detailed Description</td>
<td>1116</td>
</tr>
<tr>
<td>33.49.2 Function Documentation</td>
<td>1116</td>
</tr>
<tr>
<td>33.49.2.1 fl_x11_find()</td>
<td>1117</td>
</tr>
<tr>
<td>33.49.2.2 fl_x11_gc()</td>
<td>1117</td>
</tr>
<tr>
<td>33.49.2.3 fl_x11_xid()</td>
<td>1117</td>
</tr>
<tr>
<td>Index</td>
<td>1119</td>
</tr>
<tr>
<td>FLTK Programming Manual</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>Preface</td>
<td></td>
</tr>
<tr>
<td>Introduction to FLTK</td>
<td></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></td>
</tr>
<tr>
<td>• Box Types</td>
<td></td>
</tr>
<tr>
<td>• Labels and Label Types</td>
<td></td>
</tr>
<tr>
<td>• Drawing Images</td>
<td></td>
</tr>
<tr>
<td>Coordinates and Layout Widgets</td>
<td></td>
</tr>
<tr>
<td>How Does Resizing Work?</td>
<td></td>
</tr>
<tr>
<td>Designing a Simple Text Editor</td>
<td></td>
</tr>
<tr>
<td>Drawing Things in FLTK</td>
<td></td>
</tr>
<tr>
<td>• When Can You Draw Things in FLTK?</td>
<td></td>
</tr>
<tr>
<td>• What Drawing Units Do FLTK Drawing Functions Use?</td>
<td></td>
</tr>
<tr>
<td>• Drawing Functions</td>
<td></td>
</tr>
<tr>
<td>• Drawing Images</td>
<td></td>
</tr>
<tr>
<td>• Offscreen Drawing</td>
<td></td>
</tr>
<tr>
<td>Handling Events</td>
<td></td>
</tr>
<tr>
<td>• Fl::event_*() methods</td>
<td></td>
</tr>
<tr>
<td>• Event Propagation</td>
<td></td>
</tr>
<tr>
<td>Adding and Extending Widgets</td>
<td></td>
</tr>
<tr>
<td>Using OpenGL</td>
<td></td>
</tr>
<tr>
<td>Programming with FLUID</td>
<td></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>
<tr>
<td>Advanced FLTK</td>
<td></td>
</tr>
<tr>
<td>Unicode and UTF-8 Support</td>
<td></td>
</tr>
<tr>
<td>Appendices:</td>
<td></td>
</tr>
<tr>
<td>• Constants and Enumerations</td>
<td></td>
</tr>
<tr>
<td>• GLUT Compatibility</td>
<td></td>
</tr>
<tr>
<td>– class Fl_Glut_Window</td>
<td></td>
</tr>
<tr>
<td>• Forms Compatibility</td>
<td></td>
</tr>
<tr>
<td>• Operating System Issues</td>
<td></td>
</tr>
<tr>
<td>• Migrating Code from FLTK 1.3 to 1.4</td>
<td></td>
</tr>
<tr>
<td>• Developer Information</td>
<td></td>
</tr>
<tr>
<td>• Software License</td>
<td></td>
</tr>
<tr>
<td>• Example Source Code</td>
<td></td>
</tr>
<tr>
<td>• FAQ (Frequently Asked Questions)</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.

Version 1.4.0 introduces support for a new windowing system under LINUX: Wayland. Thus, FLTK applications under LINUX can now be built to use either X11 or Wayland as windowing system.

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
- Constants and Enumerations
- GLUT Compatibility
• Forms Compatibility
• Operating System Issues
• Migrating Code from FLTK 1.3 to 1.4
• Developer Information
• Software License
• Example Source Code

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

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

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 - 2022 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 or Wayland), Microsoft® Windows®, and Apple® macOS®.

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

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

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

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

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

- Very small & fast portable 2-D drawing library to hide Xlib, 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 with CMake

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

3.6 Building and Installing FLTK Under UNIX and macOS with make

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

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

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

For C shell and tcsh, use:

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

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

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

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

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

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

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

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

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

-disable-print

Disable print support for an X11 platform

-enable-shared

Enable generation of shared libraries

-enable-threads

Enable multithreading support

-enable-wayland

Enable the use of Wayland for all window operations, of Cairo for all graphics and of Pango for text drawing (Linux and FreeBSD only). Resulting FLTK apps use Wayland if a Wayland compositor is available at run-time, and use the equivalent of "--enable-x11 --enable-usecairo" otherwise.

-disable-xft

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

-enable-usecairo

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

-enable-pango

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

-enable-x11

This is the default under Unix and Linux. When targeting cygwin, build with X11 GUI instead of windows GDI. Also applicable to macOS platforms supplemented with XQuartz.

-enable-wayland

Enable use of the Wayland system for window handling.

-enable-cairo

Enable support of class Fl_Cairo_Window (all platforms, requires the Cairo library).
3.6 Building and Installing FLTK Under UNIX and macOS with make

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

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

- `--enable-cp936`
  Under X11, enable use of the GB2312 locale

- `--bindir=/path`
  Set the location for executables [default = $prefix/bin]

- `--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".
3.7 Building FLTK Under Microsoft Windows

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

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

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

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

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

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

3.7.1 GNU toolsets (Cygwin or MinGW) hosted on Windows

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

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

```
./configure
make
```

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

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

```
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.
3.7.2 Using the Visual C++ DLL Library

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

When compiling an application or DLL that uses the FLTK DLL, you will need to define the \texttt{FL\_DLL} preprocessor symbol to get the correct linkage commands embedded within the FLTK header files.

3.8 Internet Resources

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

WWW

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

NNTP Newsgroups

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

3.9 Reporting Bugs

To report a bug in FLTK, or for feature requests, please use the form at \url{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 \url{https://www.fltk.org/newsgroups.php}.
Chapter 4

FLTK Basics

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

4.1 Writing Your First FLTK Program

All programs must include the file `<FL/Fl.H>`. This file should be included as the first FLTK header file. 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"

```cxx
#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.

```cxx
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.

```cxx
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:

```cxx
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.

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

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

```cxx
window->show(argc, argv);
return Fl::run();
```
The resulting program will display the "Hello, World!" window:

![Hello, World!](image.png)

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

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

### 4.1 Creating the Widgets

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

```cpp
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 `myGroup->begin()` and `myGroup->end()`. In this example, `myGroup` would be the current group.

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

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

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

### 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 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 Writing Your First FLTK Program

4.1.4 Redrawing After Changing Attributes

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

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`.

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:

```cpp
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 Naming Conventions

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

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

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

- Constants and Enumerations are uppercase: `FL_FOO`.

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

4.3 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>
```

4.4 Compiling Programs that Use FLTK

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

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

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

4.4.1 Compiling Programs with Standard Compilers

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

```c
cc ++ -I/usr/local/include ...
```
4.4 Compiling Programs that Use FLTK

Note
You need a C++ compiler to build FLTK. The commands given in this chapter are examples using `c++`. Please replace this command with the C++ compiler suitable for your system or use the `fltk-config` script as described below (this is recommended).

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

```sh
fltk-config --cc
fltk-config --cxx
```

return the C and C++ compiler commands used to build FLTK.
```
c++ 'fltk-config --cxxflags' ...
```

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

Similarly, when linking your application you will need to tell the compiler to use the FLTK library:
```
c++ ... -L/usr/local/lib -lfltk -lXext -lX11 ... -lm -ldl
```

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

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

Note
The separate `fltk_cairo` library will likely be removed in FLTK 1.4.0 (this is work in progress).

The libraries are named "fltk.lib", "fltk_gl.lib", "fltk_forms.lib", "fltk_images.lib", and fltk_cairo.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:
```
c++ ... 'fltk-config --ldflags'
```

The forms, GL, and images libraries are included with the "--use-foo" options, as follows:
```
c++ ... 'fltk-config --use-forms --ldflags'
c++ ... 'fltk-config --use-gl --ldflags'
c++ ... 'fltk-config --use-images --ldflags'
c++ ... 'fltk-config --use-cairo --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-cairo --compile filename.cpp
fltk-config --use-forms --use-gl --use-images --compile filename.cpp
```

Any of these will create an executable named `filename` (or `filename.exe` under Windows).

Note
`fltk-config --compile` accepts only a limited set of file extensions for C++ source files: `.cpp`, `.cxx`, `.cc`, and `.C` (capital 'C').
```
fltk-config --help
```

displays all available options.
4.4.2 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:

```
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
OBJJS = 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) $(OBJJS) $(LDSTATIC)
CubeMain.o: CubeMain.cxx CubeViewUI.h
CubeView.o: CubeView.cxx CubeViewUI.h CubeViewUIUI.h
CubeViewUI.o: CubeViewUI.cxx CubeView.h
clean: $(TARGET) $(OBJJS)
  rm -f *.o 2> /dev/null
  rm -f $(TARGET) 2> /dev/null
```

4.4.3 Compiling Programs with Microsoft Visual C++

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

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

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

Note

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

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

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

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

![Figure 5.1 FLTK Button Widgets](image)

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

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

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

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

For toggle and radio buttons, the `value()` method returns the current button state (0 = off, 1 = on). The `set()` and `clear()` methods can be used on toggle buttons to turn a toggle button on or off, respectively. Radio buttons can be turned on with the `setonly()` method; this will also turn off other radio buttons in the same group.
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.
Figure 5.2 FLTK valuator widgets

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

## 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 \texttt{Colors} section of \textit{Drawing Things in FLTK} for implementation details.

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

- \texttt{FL_BLACK}
- \texttt{FL_RED}
- \texttt{FL_GREEN}
- \texttt{FL_YELLOW}
- \texttt{FL_BLUE}
- \texttt{FL_MAGENTA}
- \texttt{FL_CYAN}
- \texttt{FL_WHITE}

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

- \texttt{FL_FOREGROUND_COLOR}
- \texttt{FL_BACKGROUND_COLOR}
- \texttt{FL_INACTIVE_COLOR}
- \texttt{FL_SELECTION_COLOR}

The full list of named color values can be found in \texttt{FLTK Enumerations}.

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

\begin{verbatim}
Fl_Color c = fl_rgb_color(85, 170, 255); // RGB to Fl_Color
Fl::get_color(c, r, g, b); // Fl_Color to RGB
\end{verbatim}

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

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

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

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

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

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

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

\begin{verbatim}
button->color(0xff000000); // RGB: red
button->color(0x00ff0000); // RGB: green
button->color(0x0000ff00); // RGB: blue
button->color(0xffffffff); // RGB: white
\end{verbatim}
5.7 Box Types

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

These are the standard box types included with FLTK:

![FLTK Standard Box Types](image)

**Figure 5.3 FLTK Standard Box Types**

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

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:

```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()`, `labelsizes()`, `labeltype()`, `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. These are the available symbols:

![Figure 5.4 FLTK label symbols](image)

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

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

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

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

### `align()`

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

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

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 label types 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 \texttt{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 \textit{inside} this bounding box, even if \texttt{FL\_ALIGN\_INSIDE} is not enabled. The function is not called if the label value is \texttt{NULL}.

The measure function is called with a pointer to a \texttt{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 \texttt{w} and \texttt{h} to the size it will occupy.

Adding Your Label Type

The \texttt{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 \texttt{n} can be any integer value starting at the constant \texttt{FL\_FREE\_LABELTYPE}. Once you have added the label type you can use the \texttt{labeltype()} method to select your label type.

The \texttt{Fl::set\_labeltype()} method can also be used to overload an existing label type such as \texttt{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 \texttt{void drawit(Fl\_Color c)} which typically uses the functions described in \textbf{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 \texttt{fl\_add\_symbol()}:

```c
int fl\_add\_symbol(const char *name, void (*drawit)(Fl\_Color), int scalable)
```

\texttt{name} is the name of the symbol without the \texttt{"@"}; \texttt{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_CHANGED);
```

```c
button->when(FL_WHEN_RELEASE);
```

```c
button->when(FL_WHEN_ENTER_KEY);
```

```c
button->when(FL_WHEN_CHANGED | FL_WHEN_NOT_CHANGED);
```

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

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

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

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

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

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(FLCTRL + 'b');
button->shortcut(FLALT + '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

Coordinates and Layout Widgets

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

6.1 The widget coordinate system

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

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

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

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

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

Figure 6.1 FLTK coordinate system
6.2 Layout and container widgets

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

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

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

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

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

6.2.1 The Fl_Pack layout widget

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

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

![Fl_Pack test program screenshot](image)
6.2 Layout and container widgets

6.2.2 The Fl_Scroll container widget

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

![Fl_Scroll container widget](image)

6.2.3 The Fl_Tabs container widget

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

![Fl_Tabs container widget](image)

6.2.4 The Fl_Tile layout widget

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

![Fl_Tile layout widget](image)
6.2.5 The Fl_Wizard container widget

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

![Panel 1]

Figure 6.6 Fl_Wizard container widget
Chapter 7

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.

7.1 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((Fl_Widget*)0); // no resizing
    group->end();
```

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

7.2 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.
7.3 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;
7.4 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:
The message text area would be the logical choice so that the user can expand the dialog to see if there is more of an explanation below the short error message. This results in the behavior shown in the diagram below.

![Figure 7.4 Resizing dialog example (b)](image)

The result is close to what we want, but not quite: the text area will fully resize, the "!" icon box will resize vertically but not horizontally, which we can live with, but the "Darn!" button will - wait a minute - resize horizontally? That's ugly. How do we stop that from happening? Simple: put it in its own group and set the resizable to an invisible box widget, as shown in the diagram below.

![Figure 7.5 Resizing dialog example (c)](image)

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:

```java
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
darn->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?
7.4 Practical examples

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

![figure 7.6](image)

**Figure 7.6 Resizing input fields example (b)**

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!

![figure 7.7](image)

**Figure 7.7 Resizing input fields example (b)**

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

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

![figure 7.8](image)

**Figure 7.8 Resizing a row of widgets (a)**

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

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

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

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

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

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

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

To summarize the key points of the new technique:

- The new resizable widget must overlap the widgets on each side by exactly the same amount.

- The width of the new resizable widget is not fixed, but should probably be a relatively small value to avoid potential problems.

- The total width of the two new groups must equal the width of the existing group and there can be no offsets or gaps between them because margins and gaps will affect the resizing behavior.

- The same principles apply to vertical resizing.
Chapter 8

Designing a Simple Text Editor

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

8.1 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.

8.2 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:

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];
};

Generated by Doxygen
8.3 Variables

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

```c
int changed = 0;
char filename[FL_PATH_MAX] = "";
char title[FL_PATH_MAX];
Fl_Text_Buffer *textbuf = 0;
```

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.

8.4 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 },
    { "Close View", FL_COMMAND + 'w', (Fl_Callback *)close_cb, 0, FL_MENU_DIVIDER },
    { "Exit", FL_COMMAND + 'q', (Fl_Callback *)quit_cb, 0, 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 + 't', replace2_cb },
    { 0, 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.

8.5 Editing the Text

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

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

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

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

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

```c
w->editor->textfont(FL_COURIER);
```
8.6 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:

![Find: Replace: Replace All Replace Next Cancel](image)

Figure 8.1 The search and replace dialog

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 ");

8.7 Callbacks

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

8.7.1 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, 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.

8.7.2 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);
}
```

8.7.3 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);
}
```
8.7.4 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();
}
```

8.7.5 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);
    }
}
```

8.7.6 find2_cb()

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

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

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

8.7.7 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();
}
```
8.7 Callbacks

8.7.8 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.

8.7.9 paste_cb()

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

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

8.7.10 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);
}
```

8.7.11 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();
}
```

8.7.12 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);
}
```
### 8.7.13 replall_cb()

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

```c
void replall_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *find = e->replace_find->value();
    const char *replace = e->replace_with->value();
    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();
        if (found) {
            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->show_insert_position();
                times++;
            } else fl_message("Replaced %d occurrences.", times);
        } else fl_alert("No occurrences of \"%s\" found!", find);
    }
}
```

### 8.7.14 replcan_cb()

This callback just hides the replace dialog:

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

### 8.7.15 save_cb()

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

```c
void save_cb(void) {
    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.

### 8.7.16 saveas_cb()

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

```c
void saveas_cb(void) {
    char *newfile;
    newfile = fl_file_chooser("Save File As?", "+", filename);
    if (newfile != NULL) save_file(newfile);
}
```

The `save_file()` function saves the current file to the specified filename.
8.8 Other Functions

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

8.8.1 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;
}
```

8.8.2 load_file()

This function loads the specified file into the `textbuf` variable:

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

8.8.3 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 \n%s\n: %s.", newfile, strerror(errno));
    else
        strcpy(filename, newfile);
        changed = 0;
        textbuf->call_modify_callbacks();
}
```

8.8.4 set_title()

This function checks the `changed` variable and updates the window label accordingly:

```c
void set_title(Fl_Window* w) {
    if (filename[0] == '\0') strcpy(title, "Untitled");
    else {
        char *slash;
        slash = strrchr(filename, '/');
        #ifdef _WIN32
        if (slash == NULL) slash = strrchr(filename, '\');
        #endif
        if (slash != NULL) strcpy(title, slash + 1);
        else strcpy(title, filename);
        if (changed) strcat(title, " (modified)");
    }
    w->label(title);
}
```
8.9 The main() Function

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

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

8.10 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:

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

or by using the fltk-config script with:

```bash
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!

8.11 The Final Product

The final editor window should look like this:

![Figure 8.2 The completed editor window](image)

---

Generated by Doxygen
8.12 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.

8.12.1 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/, which uses a parallel "style" buffer which tracks the font, color, and size of the text that is drawn.

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

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

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

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

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

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

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

```c
Fl_Text_Buffer *stylebuf; w->editor->highlight_data(stylebuf, styletable, sizeof(styletable) / sizeof(styletable[0]), 'A', style_unfinished_cb, 0);
```

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

```c
textbuf->add_modify_callback(style_update, w->editor);
```

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

```c
// 'style_update()' - Update the style buffer...
void style_update(int pos, // I - Position of update
                 int nInserted, // I - Number of inserted chars
                 int nDeleted, // I - Number of deleted chars
                 int nRestyled, // I - Number of restyled chars
                 const char *deletedText, // I - Text that was deleted
                 const char *insertedText, // I - Text that was inserted
                 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;
    }
```

Generated by Doxygen
50 Designing a Simple Text Editor

```c
// 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);
} else {
    // Just delete characters in the style buffer...
    stylebuf->remove(pos, pos + nDeleted);
}
// Select the area that was just updated to avoid unnecessary
// callbacks...
stylebuf->select(pos, pos + nInserted - nDeleted);
// Re-parse the changed region; we do this by parsing from the
// beginning of the line of the changed region to the end of
// the line of the changed region... Then we check the last
// style character and keep updating if we have a multi-line
// comment character...
start = textbuf->line_start(pos);
end = textbuf->line_end(pos + nInserted - nDeleted);
text = textbuf->text_range(start, end);
style = stylebuf->text_range(start, end);
last = style[end - start - 1];
style_parse(text, style, end - start);
stylebuf->replace(start, end, style);
((Fl_Text_Editor *)cbArg)->redisplay_range(start, end);
}
free(text);
free(style);
}

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

```
\*bufptr = '\0';
bufptr = buf;
if (bsearch(bufptr, code_types,
    sizeof(code_types) / sizeof(code_types[0]),
    sizeof(code_types[0]), compare_keywords)) {
    while (text < temp) {
        *style++ = 'F';
        text ++;
        length --;
        col ++;
    } text --;
    length ++;
    last = 1;
    continue;
} else if (bsearch(bufptr, code_keywords,
    sizeof(code_keywords) / sizeof(code_keywords[0]),
    sizeof(code_keywords[0]), compare_keywords)) {
    while (text < temp) {
        *style++ = 'G';
        text ++;
        length --;
        col ++;
    } text --;
    length ++;
    last = 1;
    continue;
} else if (current == 'C' && strncmp(text, "*/", 2) == 0) {
    // Close a C comment...
    *style++ = current;
    *style++ = current;
    text ++;
    length --;
    current = 'A';
    col += 2;
    continue;
} else if (current == 'D') {
    // Continuing in string...
    if (strncmp(text, '\\"', 2) == 0) {
        // Quoted end quote...
        *style++ = current;
        text ++;
        length --;
        col += 2;
        continue;
    } else if (*text == '\"') {
        // End quote...
        *style++ = current;
        col +=;
        current = 'A';
        continue;
    } else if (strncmp(text, "\\n\n", 2) == 0) {
        // 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 == 'E') current = 'A';
        }
    }
}
Chapter 9

Drawing Things in FLTK

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

9.1 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().
9.2 What Drawing Units Do FLTK Drawing Functions Use?

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

FLTK version 1.4 introduces a new feature, a screen-specific **scale factor** which is a float number with a typical value in the 1-2.5 range and is used as follows: any graphical element with an FLTK value of \( v \) units is drawn on the screen with \( v \times \text{scale} \) units. Thus, a window with width 500 units is 500 \( \times \text{scale} \) pixels wide, a line of length 10 units is 10 \( \times \text{scale} \) pixels long, lines of text written using a 14-point font are 14 \( \times \text{scale} \) 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 `window`, is given by

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

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

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

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

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

GUI rescaling involves also image drawing: the screen area covered by the drawn image contains a number of pixels that grows with the scale factor. When FLTK draws images, it maps the image data (the size of these data is given by `Fl_Image::data_w()` and `Fl_Image::data_h()` to the screen area whose size (in FLTK units) is given by `Fl_Image::w()` and `Fl_Image::h()`. How exactly such mapping is performed depends on the image type, the platform and some hardware features. The most common case for `FL_RGB_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 `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 `Fl_Image_Surface` class is intended to create an `FL_RGB_Image` from a series of FLTK drawing operations. The `Fl_Image_Surface` constructor allows to control whether the size in pixels of the resulting image matches the FLTK units used when performing drawing operations, or matches the number of pixels corresponding to these FLTK units given the current value of the scale factor. The first result is obtained with `new Fl_Image_Surface(w, h)`, the second with `new Fl_Image_Surface(w, h, 1)`.

When drawing to `FL_Printer` or `FL_PostScript_File_Device`, the drawing unit is initially one point, that is, 1/72 of an inch. This unit is changed by calls to `Fl_Paged_Device::scale()`.
9.3 Drawing Functions

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

- Boxes
- Clipping
- Colors
- Line Dashes and Thickness
- 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

9.3.1 Boxes

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

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

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

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

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

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

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

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

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

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

```c
void fl_push_clip(int x, int y, int w, int h)
void fl_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.
9.3.3 Colors

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

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

![FLTK default colormap](image)

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

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 `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.

```c
Fl_Color fl_color()
```

Returns the last color that was set using `fl_color()`. This can be used for state save/restore.

```c
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.

```c
unsigned Fl::get_color(Fl_Color i)
void Fl::get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue)
```

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

```c
Fl::get_system_colors()
Fl::foreground()
Fl::background()
Fl::background2()
```

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

```c
Fl::own_colormap()
Fl::free_color(Fl_Color i, int overlay)
Fl::set_color(Fl_Color i, unsigned c)
```

`Fl::own_colormap()` is used to install a local colormap [X11 only].

`Fl::free_color()` and `Fl::set_color()` are used to remove and replace entries from the colormap.

There are two predefined graphical interfaces for choosing colors. The function `fl_show_colormap()` shows a table of colors and returns an `Fl_Color` index value. The `Fl_Color_Chooser` widget provides a standard RGB color chooser.

As the `Fl_Color` encoding maps to a 32-bit unsigned integer representing RGBI, it is also possible to specify a color using a hex constant as a color map index:
// COLOR MAP INDEX
color(0x000000ff)
-------- |
    |
    | 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.

9.3.4 Line Dashes and Thickness

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

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**  (extends past end point 1/2 line width)
• **FL_CAP_ROUND**
• **FL_CAP_SQUARE**
• **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.

### 9.3.5 Drawing Fast Shapes

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

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

Draw a single pixel at the given coordinates.

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

Color a rectangle that exactly fills the given bounding box.

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

\begin{verbatim}
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)
\end{verbatim}

Draw a 1-pixel border \textit{inside} this bounding box.

\begin{verbatim}
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)
\end{verbatim}

Draw one or two lines between the given points.

\begin{verbatim}
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)
\end{verbatim}

Outline a 3 or 4-sided polygon with lines.

\begin{verbatim}
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)
\end{verbatim}

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

\begin{verbatim}
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)
\end{verbatim}

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

\begin{verbatim}
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)
\end{verbatim}

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

\begin{verbatim}
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)
\end{verbatim}

Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and MS Windows. The advantage over using \texttt{fl\_arc()} with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.
If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3'oclock and are the starting and ending angle of the arc, \( a_2 \) must be greater or equal to \( a_1 \).

![Figure 9.2 fl_pie() and fl_arc()](image)

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

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

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

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

### 9.3.6 Drawing Complex Shapes

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

```c
void fl_push_matrix()
void fl_pop_matrix()
```
Save and restore the current transformation. The maximum depth of the stack is 32 entries.

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

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

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

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

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

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

```c
void fl_begin_line()
void fl_end_line()
```

Start and end drawing lines.

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

Start and end drawing a closed sequence of lines.

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

Start and end drawing a convex filled polygon.

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

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

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

\begin{verbatim}
void fl_vertex(double x,double y)

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 Bézier 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 \texttt{fl_arc()}. The center of the circle is given by \(x\) and \(y\), and \(r\) is its radius. \texttt{fl_arc()} takes \texttt{start} and \texttt{end} angles that are measured in degrees counter-clockwise from 3 o'clock. If \texttt{end} is less than \texttt{start} then it draws the arc in a clockwise direction.

\end{verbatim}

![Figure 9.3 fl_arc(x,y,r,a1,a2)](image_url)

\begin{verbatim}
void fl_circle(double x, double y, double r)

\end{verbatim}
9.3 Drawing Functions

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.

9.3.7 Drawing Text

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

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

Draw a nul-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

```
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()`

```
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.

```
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 nul-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()`

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

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

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

See also

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

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

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

### 9.3.8 Fonts

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

Initially only the first 16 faces are filled in. There are symbolic names for them: \texttt{FL\_HELVETICA}, \texttt{FL\_TIMES}, \texttt{FL\_COURIER}, and modifier values \texttt{FL\_BOLD} and \texttt{FL\_ITALIC} which can be added to these, and \texttt{FL\_SYMBOL} and \texttt{FL\_ZAPF\_DINGBATS}. Faces greater than 255 cannot be used in \texttt{Fl\_Widget} labels, since \texttt{Fl\_Widget} stores the index as a byte.

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

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

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 \texttt{fl\_width()}, but on X this will open the display.

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

\texttt{int fl\_font()}
\texttt{int fl\_size()}

Returns the face and size set by the most recent call to \texttt{fl\_font(a,b)}. This can be used to save/restore the font.
9.3.9 Character Encoding

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

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

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

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

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

9.3.10 Drawing Overlays

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

```c
void fl_overlay_rect(int x, int y, int w, int h)
void fl_overlay_clear()
```

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

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

9.4 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.
9.4 Drawing Images

9.4.1 Direct Image Drawing

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

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

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

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

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

Gray scale (1-channel) images may be drawn. This is done if \( \text{abs}(D) \) 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 \( W \times D \) 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.

```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.

### 9.4.2 Direct Image Reading

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

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

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

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

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

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

- **Fl_Bitmap**
- **Fl_Pixmap**
- **Fl_RGB_Image**

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

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

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

```c
char fl_can_do_alpha_blending()
```

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

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

- **Fl_GIF_Image**
- **Fl_JPEG_Image**
- **Fl_PNG_Image**
- **Fl_PNM_Image**
- **Fl_XBM_Image**
- **Fl_XPM_Image**
- **Fl_SVG_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.

```c
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.

### 9.5 Offscreen Drawing

Sometimes it can be very useful to generate a complex drawing in memory first and copy it to the screen at a later point in time. This technique can significantly reduce the amount of repeated drawing. Offscreen drawing functions are declared in `<FL/fl_draw.H>`.

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

FLTK can draw into an offscreen buffer at any time. There is no need to wait for an `Fl_Widget::draw()` to occur.

Note

In FLTK 1.3.x and earlier versions all offscreen drawing functions described below were implemented as macros and created certain temporary variables to save context information. You needed to create local scope blocks with curly braces `{ ... }` if you used offscreen functions more than once in a function or method. This is no longer necessary since offscreen drawing is now implemented in real functions (no macros).

Example:

```
Fl_Offscreen oscr = fl_create_offscreen(120, 120);
fl_begin_offscreen(oscr);
fl_color(FL_WHITE);
fl_rectf(0, 0, 120, 120);
fl_end_offscreen();
// other code here
fl_begin_offscreen(oscr);
fl_color(FL_BLACK);
fl_rectf(10, 10, 100, 100);
fl_end_offscreen();
// other code here
fl_delete_offscreen(oscr);
```

```c
Fl_Offscreen fl_create_offscreen(int w, int h)
```
9.5 Offscreen Drawing

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

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

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

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

Send all subsequent drawing commands to this offscreen buffer.

```c
void fl_end_offscreen()
```

Quit sending drawing commands to this offscreen buffer.

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

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

```c
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 10

Handling Events

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

10.1 The FLTK Event Model

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

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

Event numbers can be converted to their actual names using the `fl_eventnames[]` array defined in `<FL/names.h>`; see next chapter for details.

In the next chapter, the `MyClass::handle()` example shows how to override the `Fl_Widget::handle()` method to accept and process specific events.

10.2 Mouse Events

10.2.1 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.

10.2.2 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`.
10.2.3  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.

10.2.4  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.

10.2.5  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.

10.3  Focus Events

10.3.1  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.

10.3.2  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.

10.3.3  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.

10.3.4  FL_UNFOCUS

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

10.4.1 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.

10.4.2 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.

10.5 Widget Events

10.5.1 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).

10.5.2 FL_ACTIVATE

This widget is now active, due to activate() being called on it or one of its parents.
10.5.3 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).

10.5.4 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.

10.6 Clipboard Events

10.6.1 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().

10.6.2 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.

10.7 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.
10.7.1 **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.

10.7.2 **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.

10.7.3 **FL_DND_LEAVE**

The mouse has moved out of the widget.

10.7.4 **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.

10.8 **Other events**

10.8.1 **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.

10.8.2 **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.
10.9 Fl::event_*( ) methods

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

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

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

10.10 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_USERFOCUS, 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.

### 10.11 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 11

Adding and Extending Widgets

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

11.1 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.

11.2 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.

11.3 The Constructor

The constructor should have the following arguments:

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

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

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

MyClass: MyClass(int x, int y, int w, int h, const char *label)
: Fl_Widget(x, y, w, h, label) {
    // do initialization stuff...
}
**Fl_Widget**'s protected constructor sets \( x(), y(), w(), h() \), and label() to the passed values and initializes the other instance variables to:

```c
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(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.

```c
void Fl_Widget::set_flag(int c)
```
Calling \texttt{set\_flag(SHORTCUT\_LABEL)} modifies the behavior of \texttt{draw\_label()} so that ‘\&’ characters cause an underscore to be printed under the next letter.

\begin{verbatim}
void Fl_Widget::set_visible()
void Fl_Widget::clear_visible()
\end{verbatim}

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.

\begin{verbatim}
int Fl_Widget::test_shortcut()
static int Fl_Widget::test_shortcut(const char ∗s)
\end{verbatim}

The first version tests Fl_Widget::label() against the current event (which should be a \texttt{FL\_SHORTCUT event}). If the label contains a ‘\&’ character and the character after it matches the keypress, this returns true. This returns false if the \texttt{SHORTCUT\_LABEL} flag is off, if the label is \texttt{NULL}, or does not have a ‘\&’ character in it, or if the keypress does not match the character.

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

\textbf{Todo} Clarify Fl_Widget::test_shortcut() explanations. Fl_Widget.h says Internal Use only, but subclassing chapter gives details!

\begin{verbatim}
uchar Fl_Widget::type() const
void Fl_Widget::type(uchar t)
\end{verbatim}

The property Fl_Widget::type() can return an arbitrary 8-bit identifier, and can be set with the protected method \texttt{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 \texttt{type()} use a unique value. These unique values must be greater than the symbol \texttt{FL\_RESERVED\_TYPE} (which is 100) and less than \texttt{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).
11.5 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
11.6 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.

11.7 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.

11.8 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:

```cpp
void MyClass::static_slider_cb(Fl_Widget* v, void*) { // static method
    ((MyClass*)(v->parent()))->slider_cb();
}
void MyClass::slider_cb() { // normal method
    use(the_slider->value());
}
```
If you make the `handle()` method, you can quickly pass all the events to the children using the `Fl_Group::handle()` method. You don't need to override `handle()` if your composite widget does nothing other than pass events to the children:

```cpp
int MyClass::handle(int event) {
    if (Fl_Group::handle(event)) return 1;
    ... handle events that children don't want ...
}
```

If you override `draw()` you need to draw all the children. If `redraw()` or `damage()` is called on a child, `damage(FL_DAMAGE_CHILD)` is done to the group, so this bit of `damage()` can be used to indicate that a child needs to be drawn. It is fastest if you avoid drawing anything else in this case:

```cpp
int MyClass::draw() {
    Fl_Widget *const*a = array();
    if (damage() == FL_DAMAGE_CHILD) { // only redraw some children
        for (int i = children(); i--; a++) update_child(**a);
    } else { // total redraw
        ... draw background graphics ...
        // now draw all the children atop the background:
        for (int i = children_; i--; a++) {
            draw_child(**a);
            draw_outside_label(**a); // you may not need to do this
        }
    }
}
```

`Fl_Group` provides some protected methods to make drawing easier:

- `draw_child()`
- `draw_children()`
- `draw_outside_label()`
- `update_child()`

```cpp
void Fl_Group::draw_child(Fl_Widget &widget) const
```

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

```cpp
void Fl_Group::draw_children()
```

A convenience function that draws all children of the group. This is useful if you derived a widget from `Fl_Group` and want to draw a special border or background. You can call `draw_children()` from the derived `draw()` method after drawing the box, border, or background.

```cpp
void Fl_Group::draw_outside_label(const Fl_Widget &widget) const
```

Draw the labels that are not drawn by `draw_label()`. If you want more control over the label positions you might want to call `child->draw_label(x,y,w,h,a)`.

```cpp
void Fl_Group::update_child(Fl_Widget &widget) const
```

Draws the child only if its `damage()` is non-zero. You should call this on all the children if your own damage is equal to `FL_DAMAGE_CHILD`. Nothing is done if the child is not `visible()` or if it is clipped.
11.9 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.

11.10 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`

11.11 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 12

Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

12.1 Using OpenGL in FLTK

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

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

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

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

12.2 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
```
12.2.1 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)
};
```

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.)

12.2.2 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 ...
}
```

12.2.3 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 1;
    case FL_SHORTCUT:
        // shortcut, key is in Fl::event_key(), ascii in Fl::event_text()
        // Return 1 if you understand/use the shortcut event, 0 otherwise...
        return 1;
    default:
        // pass other events to the base class...
        return Fl_Gl_Window::handle(event);
    }
}
```

---

**Note:**
If you are using the Mesa graphics library, the call to `glDrawBuffer()` is not required and will slow down drawing considerably. The preprocessor instructions shown above will optimize your code based upon the graphics library used.
When handle() is called, the OpenGL context is not set up! If your display changes, you should call redraw() and let draw() do the work. Don’t call any OpenGL drawing functions from inside handle()!

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

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

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

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

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

You must put glwindow->show() in your main code after calling show() on the window containing the OpenGLE
window.

### 12.3 OpenGL and support of HighDPI displays

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

```c
Fl_Gl_Window *glw = ...;
glViewport(0, 0, glw->pixel_w(), glw->pixel_h());
```

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

**Note**

A further coding rule is necessary to properly support retina displays and OpenGL under macOS (see OpenGL and ‘retina’ displays)

### 12.4 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 `boxtyle` or other places with some care.

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

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

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

```cpp
gl_start();
... put your OpenGL code here ... 
gl_finish();
```

The same context is reused each time. If your code changes the projection transformation or anything else you should use `glPushMatrix()` and `glPopMatrix()` functions to put the state back before calling `gl_finish()`.

You may want to use `Fl_Window::current()`-`h()` to get the drawable height so that you can flip the Y coordinates.

Unfortunately, there are a bunch of limitations you must adhere to for maximum portability:

- You must choose a default visual with `Fl::gl_visual()`.
- You cannot pass `FL_DOUBLE` to `Fl::gl_visual()`.
- You cannot use `Fl_Double_Window` or `Fl_Overlay_Window`.

Do not call `gl_start()` or `gl_finish()` when drawing into an `Fl_Gl_Window`!

### 12.5 Using FLTK widgets in OpenGL Windows

FLTK widgets can be added to `Fl_Gl_Window`s just as they would be added to `Fl_Window`s. They are rendered as an overlay over the user defined OpenGL graphics using ‘fl_’ graphics calls that are implemented in GL.

```cpp
Fl_Gl_Window does not add subsequent widgets as children by default as `Fl_Window` does. Call `myGlWindow->begin()` after creating the GL window to automatically add following widgets. Remember to call `myGlWindow->end()`.
```

```cpp
class My_Gl_Window : public Fl_Gl_Window {
  ... 
  
  void draw();

  ... 
};
```

```cpp
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myGlWindow->end();
```
12.6 OpenGL Drawing Functions

Users can draw into the overlay by using GL graphics calls as well as all FL... graphics calls from the "Drawing Fast Shapes" section.

Widgets can be drawn with transparencies by assigning an alpha value to a colormap entry and using that color in the widget.

Transparencies can also be set directly when drawing. This can be used to create custom box types and RGB overlay drawings with an alpha channel.

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>`.

```c
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!
```

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

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

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

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

```c
int gl_height()
int gl_descent()
float gl_width(const char *s)
float gl_width(const char *s, int n)
float gl_width(uchar c)
```
Returns information about the current OpenGL font.

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

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

```c
void gl_draw(const char *s, int x, int y)
void gl_draw(const char *s, int n, int x, int y)
void gl_draw(const char *s, float x, float y)
void gl_draw(const char *s, int n, float x, float y)
```

Draws a nul-terminated string or an array of \( n \) characters in the current OpenGL font at the given position.

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

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X, and aligned with the edges or center. Exactly the same output as `fl_draw()`.

### 12.7 Speeding up OpenGL

Performance of `Fl_Gl_Window` may be improved on some types of OpenGL implementations, in particular MESA and other software emulators, by setting the `GL_SWAP_TYPE` environment variable. This variable declares what is in the backbuffer after you do a swapbuffers.

- `setenv GL_SWAP_TYPE COPY`

  This indicates that the back buffer is copied to the front buffer, and still contains its old data. This is true of many hardware implementations. Setting this will speed up emulation of overlays, and widgets that can do partial update can take advantage of this as `damage()` will not be cleared to -1.

- `setenv GL_SWAP_TYPE NODAMAGE`

  This indicates that nothing changes the back buffer except drawing into it. This is true of MESA and Win32 software emulation and perhaps some hardware emulation on systems with lots of memory.

- All other values for `GL_SWAP_TYPE`, and not setting the variable, cause FLTK to assume that the back buffer must be completely redrawn after a swap.

This is easily tested by running the `gl_overlay` demo program and seeing if the display is correct when you drag another window over it or if you drag the window off the screen and back on. You have to exit and run the program again for it to see any changes to the environment variable.
12.8 Using OpenGL Optimizer with FLTK

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

OptimizerWindow Class Definition

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

class OptimizerWindow : public Fl_Gl_Window {
    csContext *context_; // Initialized to 0 and set by draw()...
    csDrawAction *draw_action_; // Draw action...
    csGroup *scene_; // Scene to draw...
    csCamera *camera_; // Viewport for scene...
public:
    void draw();

    OptimizerWindow(int X, int Y, int W, int H, const char *L)
        : Fl_Gl_Window(X, Y, W, H, L) {
        context_ = (csContext *)0;
        draw_action_ = (csDrawAction *)0;
        scene_ = (csGroup *)0;
        camera_ = (csCamera *)0;
    }

    void scene(csGroup *g) { scene_ = g; redraw(); }
    void camera(csCamera *c) {
        camera_ = c;
        if (context_) {
            draw_action_ = (csDrawAction *)0;
            scene_ = (csGroup *)0;
            draw_action_->setCamera(camera_);
        }
        scene_->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:

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.

12.9 Using OpenGL 3.0 (or higher versions)

The examples subdirectory contains OpenGL3test.cxx, a toy program showing how to use OpenGL 3.0 (or higher versions) with FLTK in a cross-platform fashion. It contains also OpenGL3-glut-test.cxx which shows how to use FLTK’s GLUT compatibility and OpenGL 3.

To access OpenGL 3.0 (or higher versions), use the FL_OPENGL3 flag when calling Fl_Gl_Window::mode(int a) or glutInitDisplayMode().

On the Windows and Linux platforms, FLTK creates contexts implementing the highest OpenGL version supported by the hardware. Such contexts may also be compatible with lower OpenGL versions. Access to functions from OpenGL versions above 1.1 requires to load function pointers at runtime on these platforms. FLTK recommends to use the GLEW library to perform this. It is therefore necessary to install the GLEW library (see below).

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

GLEW installation (Linux and Windows platforms)

FLTK needs a header file, GL/glew.h, and a library, libGLEW.* or equivalent, to support OpenGL 3 and above.
These can be obtained for most Linux distributions by installing package libglew-dev.
For the Windows platform:

- the header and a Visual Studio static library (glew32.lib) can be downloaded from [http://glew.sourceforge.net/](http://glew.sourceforge.net/);
**12.9 Using OpenGL 3.0 (or higher versions)**

- A MinGW-style static library (libglew32.a) can be built from source (same web site) with the make command. Alternatively, pre-built files are available for these architectures:
  - x86: download files `glew.h` and `libglew32.a`;
  - x86_64: install GLEW as an MSYS2 package with command:
    `pacman -S mingw-w64-x86_64-glew`

**Source-level changes for OpenGL 3:**

- Put this in all OpenGL-using source files (instead of `#include <FL/gl.h>`, and before `#include <FL/glut.h>`) if you use GLUT:
  ```cpp
  #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`:
  ```cpp
  #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
  ```cpp
  #ifndef __APPLE__
  glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  #endif
  ```
  after the first `glutCreateWindow()` call.

If GLEW is installed on the Mac OS development platform, it is possible to use the same code for all platforms, with one exception: put
```cpp
#if defined __APPLE__
  glewExperimental = GL_TRUE;
#endif
```
before the `glewInit()` call.

**Testing for success of the `glewInit()` call**

Testing whether the `glewInit()` call is successful is to be done as follows:
```cpp
#include <FL/platform.H> // defines FLTK_USE_WAYLAND under the Wayland platform
#include <FL/Fl.H> // for Fl::warning()
#endif __APPLE__
# define GLEW_STATIC 1
#endif
#include <GL/glew.h>
GLenum err = glewInit(); // defines pters to functions of OpenGL V 1.2 and above
#ifdef FLTK_USE_WAYLAND
  if (Fl_wl_display() && err == GLEW_ERROR_NO_GLX_DISPLAY) err = GLEW_OK;
#endif
if (err != GLEW_OK) Fl::warning("glewInit() failed returning %u", err);
```

**Changes in the build process**

Link with libGLEW.so (with X11 or Wayland), libglew32.a (with MinGW) or glew32.lib (with MS Visual Studio); no change is needed on the Mac OS platform.
Chapter 13

Programming with FLUID

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

Subchapters:

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

13.1 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.

13.2 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 windowtitle`
- `--name classname`
- `--iconic`
- `--fg color`
- `--bg color`
- `--bg2 color`
- `--scheme schemename`

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

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

13.3 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.
13.4 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.

13.5 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.

### 13.5.1 The CubeView Class

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

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

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

```cpp
#include <FL/Fl.H>
#include <FL/Fl_Gl_Window.H>
#include <FL/gl.h>
class CubeView : public Fl_Gl_Window {
public:
  CubeView(int x, int y, int w, int h, const char *l = 0);
  // This value determines the scaling factor used to draw the cube.
  double size;
  // Set the rotation about the vertical (y) axis.
  double v_angle(double angle)
  { vAng = angle; } // Return the rotation about the vertical (y) axis.
  double v_angle() const { return vAng; }
  // This function is called by the vertical roller in CubeViewUI and the initialize button in CubeViewUI.
  // Set the rotation about the horizontal (x) axis.
  double h_angle(double angle)
  { hAng = angle; } // The rotation about the horizontal (x) axis.
  double h_angle() const { return hAng; }
  // This function is called by the horizontal roller in CubeViewUI and the initialize button in CubeViewUI.
  // Sets the x shift of the cube view camera.
  double panx(double x)
  { xshift = x; } // Sets the y shift of the cube view camera.
  double pany(double y)
  { yshift = y; } // The widget class draw() override.
  void draw();
private:
  // Draw the cube boundaries.
  void drawCube();
  double vAng, hAng;
  double xshift, yshift;
  float boxv0[3]; float boxv1[3];
  float boxv2[3]; float boxv3[3];
  float boxv4[3]; float boxv5[3];
  float boxv6[3]; float boxv7[3];
};
```

The CubeView Class Implementation

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

```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)
  | Fl::use_high_res_GL(1);
  vAng = 0.0;
  hAng = 0.0;
  size = 10.0;
  xshift = 0.0;
```

Generated by Doxygen
yshift = 0.0;
/* The cube definition. These are the vertices of a unit cube
   centered on the origin. */
boxv0[0] = -0.5; boxv0[1] = -0.5; boxv0[2] = -0.5;
boxv1[0] = 0.5; boxv1[1] = -0.5; boxv1[2] = -0.5;
boxv2[0] = 0.5; boxv2[1] = 0.5; boxv2[2] = -0.5;
boxv3[0] = -0.5; boxv3[1] = 0.5; boxv3[2] = -0.5;
boxv4[0] = -0.5; boxv4[1] = -0.5; boxv4[2] = 0.5;
boxv5[0] = 0.5; boxv5[1] = -0.5; boxv5[2] = 0.5;
boxv6[0] = 0.5; boxv6[1] = 0.5; boxv6[2] = 0.5;
boxv7[0] = -0.5; boxv7[1] = 0.5; boxv7[2] = 0.5;
}

void CubeView::drawCube() {
    /* Draw a colored cube */
    #define ALPHA 0.5
    glEnable(GL_QUADS);
    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(boxv2);
    glVertex3fv(boxv3);
    glVertex3fv(boxv3);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv5);
    glVertex3fv(boxv5);
    glVertex3fv(boxv6);
    glVertex3fv(boxv6);
    glVertex3fv(boxv7);
    glVertex3fv(boxv7);
    glVertex3fv(boxv4);
    glVertex3fv(boxv0);
    glVertex3fv(boxv4);
    glVertex3fv(boxv1);
    glVertex3fv(boxv5);
    glVertex3fv(boxv2);
    glVertex3fv(boxv6);
    glVertex3fv(boxv3);
    glVertex3fv(boxv7);
    glEnd();
}

void CubeView::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0, 0, pixel_w(), pixel_h());
        glOrtho(-10, 10, -10, 10, -20050, 100000);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glTranslatef((GLfloat)xshift, (GLfloat)yshift, 0);
13.5 A Short Tutorial

```c
glRotatef((GLfloat)hAng, 0, 1, 0);
glRotatef((GLfloat)vAng, 1, 0, 0);
glScalef(float(size), float(size), float(size));
drawCube();
glPopMatrix();
```

13.5.2 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 13.3 FLUID file for CubeView](image)

Adding the Class Constructor

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

Then add a window to the CubeViewUI class. Highlight the name of the constructor in the FLUID browser window and click on **New->Group->Window**. In a similar manner add the following to the CubeViewUI constructor:
• A horizontal roller named hrot
• A vertical roller named vrot
• A horizontal slider named xpan
• A vertical slider named ypan
• A horizontal value slider named zoom

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

When you are finished you should have something like this:

![Figure 13.4 FLUID window containing CubeView demo](image)

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.
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:

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

13.5.3 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.

13.5.4 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.

13.6 FLUID Reference

The following sections describe each of the windows in FLUID.

13.6.1 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).

### 13.6.2 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)

Generated by Doxygen
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/S婦ot**

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/Latter (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.
13.6 FLUID Reference

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.

Figure 13.7 FLUID Project Settings Window
Edit/GUI Settings... (Shift+Alt+p)

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

Figure 13.8 FLUID GUI Settings Window

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

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

Toolips provide descriptions of each option.

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

Figure 13.9 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 \texttt{Fl\_Window} pointer. The unnamed window will be returned from it (more than one unnamed window is useless). If the function contains only named windows, it will be declared as returning nothing \texttt{(void)}.

It is possible to make the \texttt{.cxx} output be a self-contained program that can be compiled and executed. This is done by deleting the function name so \texttt{main(argc, argv)} is used. The function will call \texttt{show()} on all the windows it creates and then call \texttt{Fl\:\:run()}. This can also be used to test resize behavior or other parts of the user interface.

You can change the function name by double-clicking on the function.

New/Window

Creates a new \texttt{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 \texttt{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.

Figure 13.10 FLUID Layout/Grid Settings Window
Shell/Execute Command... (Alt+x)

Displays the shell command panel. The shell command is commonly used to run a 'make' script to compile the FLTK output.

Shell/Execute Again (Alt+g)

Run the shell command again.

Help/About FLUID

Pops up a panel showing the version of FLUID.

Help/On FLUID

Shows this chapter of the manual.

Help/Manual

Shows the contents page of the manual.
13.6.3 The Widget Panel

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

When you change attributes using this panel, the changes are reflected immediately in the window. It is useful to hit the "no overlay" button (or type Ctrl+Shift+O) to hide the red overlay so you can see the widgets more accurately, especially when setting the box type.

If you have several widgets selected, they may have different values for the fields. In this case the value for one of the widgets is shown. But if you change this value, all of the selected widgets are changed to the new value.

Hitting "OK" makes the changes permanent. Selecting a different widget also makes the changes permanent. FLUID checks for simple syntax errors such as mismatched parenthesis in any code before saving any text.

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

![Figure 13.11 The FLUID widget GUI attributes](image)

13.7 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.

![Figure 13.12 The FLUID widget Style attributes](image)

**13.7.1 Style Attributes**

**Label Font (pulldown menu)**

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

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

Label Color (button)

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

Box (pulldown menu)

The boxtype to draw as a background for the widget.

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

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

Down Box (pulldown menu)

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

Color (button)

The color to draw the box with.

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

Text Font, Size, and Color

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

![FLUID widget C++ attributes](image)

**Figure 13.13 The FLUID widget C++ attributes**

### 13.7.2 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 \( . \text{cxx} \) 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 \( \text{void name(<class>*, void*)} \).

A code snippet is inserted into a static function in the \( . \text{cxx} \) output file. The function prototype is \( \text{void name(class *o, void *v)} \) 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 \( \text{void*} \) 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 \( \text{void*} \) 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.

### 13.8 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:
13.9 Image Labels

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

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

Notes for All Image Types

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

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

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

Don’t rely on how FLTK crops images that are outside the widget, as this may change in future versions! The cropping of inside labels will probably be unchanged.

To more accurately place images, make a new "box" widget and put the image in that as the label.

XBM (X Bitmap) Files

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

FLUID writes code to construct an `Fl_Bitmap` image and use it to label the widget. The '1' bits in the bitmap are drawn using the label color of the widget. You can change this color in the FLUID widget attributes panel. The '0' bits are transparent.
The program "bitmap" on the X distribution does an adequate job of editing bitmaps.

XPM (X Pixmap) Files

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

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

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

BMP Files

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

GIF Files

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

JPEG Files

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

PNG (Portable Network Graphics) Files

If FLTK is compiled with PNG support, FLUID can read PNG image files which are often used in HTML documents. FLUID uses a Fl_PNG_Image image to label the widget, and writes uncompressed RGB or grayscale data to the source file. PNG images can provide a full alpha channel for partial transparency, and FLTK supports this as best as possible on each platform.
13.10 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".

13.11 Internationalization with FLUID

FLUID supports internationalization (I18N for short) of label strings and tooltips used by widgets. The GNU gettext option also supports deferred translation of statically initialised menu item labels. The preferences window (Ctrl+p) provides access to the I18N options.

13.11.1 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.
13.11.2 Using GNU gettext for I18N

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

To use GNU gettext for I18N, open the preferences window and choose "GNU gettext" from the Use chooser. Four new input fields will then appear to control the include file and function/macro names to use when retrieving localized label strings in dynamic allocation and static initialisation.

![Project Settings](image)

Figure 13.14 Internationalization using GNU gettext

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

If the Conditional: field contains a macro name, i18n will only be compiled into the product if this macro is defined. The build system should define the macro only if all required headers and libraries are available. If the macro is not defined, no headers are included and gettext passes text through untranslated.

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

The Static Function: field names a macro that will mark static text fields for extraction with the xgettext tool. The default macro name is gettext_noop and will be defined as #define gettext_noop(text) text right after the #include statement. Fluid will do its best to call gettext on static texts after the textdomain was set by the user.

See also

    GNU gettext special cases
13.11.3 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.

```
#include <nl_types.h>
```

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.

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.

13.12 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.

13.13 Keyboard Shortcuts

On Apple computers, use the Apple Command key instead of Ctrl.
<table>
<thead>
<tr>
<th>Key Combo</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>widget properties dialog</td>
</tr>
<tr>
<td>F2</td>
<td>move widget earlier in tree</td>
</tr>
<tr>
<td>F3</td>
<td>move widget later in tree</td>
</tr>
<tr>
<td>F7</td>
<td>group widgets</td>
</tr>
<tr>
<td>F8</td>
<td>ungroup widgets</td>
</tr>
<tr>
<td>Delete</td>
<td>delete selected widgets</td>
</tr>
<tr>
<td>Ctrl-0..9</td>
<td>load design from history</td>
</tr>
<tr>
<td>Alt-1</td>
<td>label text tiny</td>
</tr>
<tr>
<td>Alt-2</td>
<td>label text small</td>
</tr>
<tr>
<td>Alt-3</td>
<td>label text normal</td>
</tr>
<tr>
<td>Alt-4</td>
<td>label text medium</td>
</tr>
<tr>
<td>Alt-5</td>
<td>label text large</td>
</tr>
<tr>
<td>Alt-6</td>
<td>label text huge</td>
</tr>
<tr>
<td>Ctrl-A</td>
<td>select all</td>
</tr>
<tr>
<td>Shift-Ctrl-A</td>
<td>select none</td>
</tr>
<tr>
<td>Alt-B</td>
<td>widget bin</td>
</tr>
<tr>
<td>Ctrl-C</td>
<td>copy widgets</td>
</tr>
<tr>
<td>Shift-Ctrl-C</td>
<td>generate C code</td>
</tr>
<tr>
<td>Ctrl-G</td>
<td>grid setting dialog</td>
</tr>
<tr>
<td>Alt-G</td>
<td>execute again</td>
</tr>
<tr>
<td>Shift-Alt-G</td>
<td>FLTK settings dialog</td>
</tr>
<tr>
<td>Ctrl-I</td>
<td>insert design into project</td>
</tr>
<tr>
<td>Ctrl-N</td>
<td>new design</td>
</tr>
<tr>
<td>Shift-Ctrl-N</td>
<td>new design from template</td>
</tr>
<tr>
<td>Ctrl-O</td>
<td>open design file</td>
</tr>
<tr>
<td>Shift-Ctrl-O</td>
<td>toggle overlays</td>
</tr>
<tr>
<td>Ctrl-P</td>
<td>print all windows</td>
</tr>
<tr>
<td>Alt-P</td>
<td>project settings</td>
</tr>
<tr>
<td>Shift-Alt-P</td>
<td>Fluid settings dialog</td>
</tr>
<tr>
<td>Ctrl-Q</td>
<td>quit Fluid</td>
</tr>
<tr>
<td>Ctrl-S</td>
<td>save design</td>
</tr>
<tr>
<td>Shift-Ctrl-S</td>
<td>save design with new name</td>
</tr>
<tr>
<td>Shift-Alt-S</td>
<td>source view window</td>
</tr>
<tr>
<td>Ctrl-U</td>
<td>duplicate widgets</td>
</tr>
<tr>
<td>Ctrl-V</td>
<td>paste widgets</td>
</tr>
<tr>
<td>Shift-Ctrl-W</td>
<td>write text strings</td>
</tr>
<tr>
<td>Ctrl-X</td>
<td>cut selected Widgets</td>
</tr>
<tr>
<td>Alt-X</td>
<td>show 'execute command' dialog</td>
</tr>
<tr>
<td>Ctrl-Z</td>
<td>undo</td>
</tr>
<tr>
<td>Shift-Ctrl-Z</td>
<td>redo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Combo</th>
<th>Function in interactive Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMB</td>
<td>select one widget</td>
</tr>
<tr>
<td>Shift-LMB</td>
<td>extend widget selection</td>
</tr>
<tr>
<td>Shift-LMB-Drag</td>
<td>window only: resize proportional</td>
</tr>
<tr>
<td>Tab</td>
<td>select next</td>
</tr>
<tr>
<td>Shift-Tab</td>
<td>select previous</td>
</tr>
<tr>
<td>Arrow</td>
<td>move by one unit</td>
</tr>
<tr>
<td>Shift-Arrow</td>
<td>resize by one unit</td>
</tr>
<tr>
<td>Ctrl-Arrow</td>
<td>move by grid units</td>
</tr>
</tbody>
</table>
### Keyboard Shortcuts

<table>
<thead>
<tr>
<th>Key Combo</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift-Ctrl-Arrow</td>
<td>resize by grid units</td>
</tr>
</tbody>
</table>
Chapter 14

Advanced FLTK

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

14.1 Multithreading

FLTK can be used to implement a GUI for a multithreaded application but, as with multithreaded programming generally, there are some concepts and caveats that must be kept in mind.

Key amongst these is that, for many of the target platforms on which FLTK is supported, only the 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.

14.2 FLTK multithread locking - Fl::lock() and Fl::unlock()

In a multithreaded program, drawing of widgets (in the main() thread) happens asynchronously to widgets being updated by worker threads, so no drawing can occur safely whilst a widget is being modified (and no widget should be modified whilst drawing is in progress).

FLTK supports multithreaded applications using a locking mechanism internally. This allows a worker thread to lock the rendering context, preventing any drawing from taking place, whilst it changes the value of its widget.
Note

The converse is also true; whilst a worker thread holds the lock, the main() thread may not be able to process any drawing requests, nor service any events. So a worker thread that holds the FLTK lock must contrive to do so for the shortest time possible or it could impair operation of the application.

The lock operates broadly as follows.

Using the FLTK library, the main() thread holds the lock whenever it is processing events or redrawing the display. It acquires (locks) and releases (unlocks) the FLTK lock automatically and no "user intervention" is required. Indeed, a function that runs in the context of the main() thread ideally should not acquire / release the FLTK lock explicitly. (Though note that the lock calls are recursive, so calling Fl::lock() from a thread that already holds the lock, including the main() thread, is benign. The only constraint is that every call to Fl::lock() must be balanced by a corresponding call to Fl::unlock() to ensure the lock count is preserved.)

The main() thread must call Fl::lock() once before any windows are shown, to enable the internal lock (it is "off" by default since it is not useful in single-threaded applications) but thereafter the main() thread lock is managed by the library internally.

A worker thread, when it wants to alter the value of a widget, can acquire the lock using Fl::lock(), update the widget, then release the lock using Fl::unlock(). Acquiring the lock ensures that the worker thread can update the widget, without any risk that the main() thread will attempt to redraw the widget whilst it is being updated.

Note that acquiring the lock is a blocking action; the worker thread will stall for as long as it takes to acquire the lock. If the main() thread is engaged in some complex drawing operation this may block the worker thread for a long time, effectively serializing what ought to be parallel operations. (This frequently comes as a surprise to coders less familiar with multithreaded programming issues; see the discussion of "lockless programming" later for strategies for managing this.)

To incorporate the locking mechanism in the library, FLTK must be compiled with -enable-threads set during the configure process. IDE-based versions of FLTK are automatically compiled with the locking mechanism incorporated if possible. Since version 1.3, the configure script that builds the FLTK library also sets -enable-threads by default.

14.3 Simple multithreaded examples using Fl::lock

In main(), call Fl::lock() once before Fl::run() or Fl::wait() to enable the lock and start the runtime multithreading support for your program. All callbacks and derived functions like handle() and draw() will now be properly locked.

This might look something like this:

```c
int main(int argc, char **argv) {
    /* Create your windows and widgets here */
    Fl::lock(); /* "start" the FLTK lock mechanism */
    /* show your window */
    main_win->show(argc, argv);
    /* start your worker threads */
    ... start threads ...
    /* Run the FLTK main loop */
    int result = Fl::run();
    /* terminate any pending worker threads */
    ... stop threads ...
    return result;
}
```

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

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

Generated by Doxygen
14.3 Simple multithreaded examples using Fl::lock

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:

```cpp
int main(int argc, char **argv) {
    /* Create your windows and widgets here */
    Fl::lock(); /* start the FLTK lock mechanism */
    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):

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

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

Using Fl::awake callback messages

You can also request that the main() thread call a function on behalf of the worker thread by using Fl::awake(Fl_Awake_Handler cb, void* userdata).

The main() thread will execute the callback “as soon as possible” when next processing the pending events. This can be used by a worker thread to perform operations (for example showing or hiding windows) that are prohibited in a worker thread.

```cpp
void do_something_cb(void* userdata) {
    // Will run in the context of the main thread
    do_stuff ...
}
// running in worker thread
void *data; // "data" is a pointer to your user data
Fl::awake(do_something_cb, data); // call to execute cb in main thread
```

Note

The main() thread will execute the Fl_Awake_Handler callback do_something_cb asynchronously to the worker thread, at some short but indeterminate time after the worker thread registers the request. When it executes the Fl_Awake_Handler callback, the main() thread will use the contents of *userdata at the time of execution, not necessarily the contents that *userdata had at the time that the worker thread posted the callback request. The worker thread should therefore contrive not to alter the contents of *userdata once it posts the callback, since the worker thread does not know when the main() thread will consume that data. It is often useful that userdata point to a struct, one member of which the main() thread can modify to indicate that it has consumed the data, thereby allowing the worker thread to re-use or update userdata.
Warning

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

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

14.4 FLTK multithreaded "lockless programming"

The simple multithreaded examples shown above, using the FLTK lock, work well for many cases where multiple threads are required. However, when that model is extended to more complex programs, it often produces results that the developer did not anticipate.

A typical case might go something like this. A developer creates a program to process a huge data set. The program has a main() thread and 7 worker threads and is targeted to run on an 8-core computer. When it runs, the program divides the data between the 7 worker threads, and as they process their share of the data, each thread updates its portion of the GUI with the results, locking and unlocking as they do so.

But when this program runs, it is much slower than expected and the developer finds that only one of the eight CPU cores seems to be utilised, despite there being 8 threads in the program. What happened?

The threads in the program all run as expected, but they end up being serialized (that is, not able to run in parallel) because they all depend on the single FLTK lock. Acquiring (and releasing) that lock has an associated cost, and is a blocking action if the lock is already held by any other worker thread or by the main() thread.

If the worker threads are acquiring the lock "too often", then the lock will always be held somewhere and every attempt by any other thread (even main()) to lock will cause that other thread (including main()) to block. And blocking main() also blocks event handling, display refresh...

As a result, only one thread will be running at any given time, and the multithreaded program is effectively reduced to being a (complicated and somewhat less efficient) single thread program.

A "solution" is for the worker threads to lock "less often", such that they do not block each other or the main() thread. But judging what constitutes locking "too often" for any given configuration, and hence will block, is a very tricky question. What works well on one machine, with a given graphics card and CPU configuration may behave very differently on another target machine.

There are "interesting" variations on this theme, too: for example it is possible that a "faulty" multithreaded program such as described above will work adequately on a single-core machine (where all threads are inherently serialized anyway and so are less likely to block each other) but then stall or even deadlock in unexpected ways on a multicore machine when the threads do interfere with each other. (I have seen this - it really happens.)

The "better" solution is to avoid using the FLTK lock so far as possible. Instead, the code should be designed so that the worker threads do not update the GUI themselves and therefore never need to acquire the FLTK lock. This would be FLTK multithreaded "lockless programming".

There are a number of ways this can be achieved (or at least approximated) in practice but the most direct approach is for the worker threads to make use of the Fl::awake(Fl_Awake_Handler cb, void* userdata) method so that GUI updates can all run in the context of the main() thread, alleviating the need for the worker thread to ever lock. The onus is then on the worker threads to manage the userdata so that it is delivered safely to the main() thread, but there are many ways that can be done.
14.5 FLTK multithreaded Constraints

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?

14.5 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 15

Unicode and UTF-8 Support

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

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

15.1 About Unicode, ISO 10646 and UTF-8

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

For further information, please see:

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

The Unicode Standard

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

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

Unicode also provides higher-level concepts needed for text processing and typographic publishing systems, such as algorithms for sorting and comparing text, composite character and text rendering, right-to-left and bi-directional text handling.
Note
There are currently no plans to add this extra functionality to FLTK.

ISO 10646

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

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

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

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

ISO 10646 was originally designed to handle a 31-bit character set from U+00000000 to U+7FFFFFFF, but the current idea is that 21 bits will be sufficient for all future needs, giving characters up to U+10FFFF. The complete character set is sub-divided into **planes**. **Plane 0**, also known as the **Basic Multilingual Plane (BMP)**, ranges from U+0000 to U+FFFD 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+0FFFF. UCS-4 uses four bytes, which lets it handle higher characters, but this is even more wasteful for ASCII or Latin1.

**UTF-8**

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

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

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

Todo  Work through the code and this documentation to harmonize the [OksiD] and [fltk2] functions.

FLTK will be entirely converted to Unicode using UTF-8 encoding. If a different encoding is required by the underlying operating system, FLTK will convert the string as needed.

It is important to note that the initial implementation of Unicode and UTF-8 in FLTK involves three important areas:

- provision of Unicode character tables and some simple related functions;
- conversion of char∗ variables and function parameters from single byte per character representation to UTF-8 variable length sequences;
- modifications to the display font interface to accept general Unicode character or UCS code numbers instead of just ASCII or Latin1 characters.

The current implementation of Unicode / UTF-8 in FLTK will impose the following limitations:

- An implementation note in the [OksiD] code says that all functions are LIMITED to 24 bit Unicode values, but also says that only 16 bits are really used under linux and win32. [Can we verify this?]
- The [fltk2] fl_utf8encode() and fl_utf8decode() functions are designed to handle Unicode characters in the range U+000000 to U+10FFFF inclusive, which covers all UTF-16 characters, as specified in RFC 3629. Note that the user must first convert UTF-16 surrogate pairs to UCS.
- FLTK will only handle single characters, so composed characters consisting of a base character and floating accent characters will be treated as multiple characters.
- FLTK will only compare or sort strings on a byte by byte basis and not on a general Unicode character basis.
- FLTK will not handle right-to-left or bi-directional text.

Todo  Verify 16/24 bit Unicode limit for different character sets? OksiD’s code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in fl_utf8fromwc() and fl_utf8toUtf16().

<table>
<thead>
<tr>
<th>Unicode range</th>
<th>Byte sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+00000000 - U+0000007F</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>U+00000080 - U+000007FF</td>
<td>110xxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00000800 - U+0000FFFF</td>
<td>1110xxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00010000 - U+001FFFFF</td>
<td>11110xxx 10xxxxxx 10xxxxxx 10xxxxxx</td>
</tr>
<tr>
<td>U+00200000 - U+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.
15.3 Illegal Unicode and UTF-8 Sequences

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

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

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

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

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

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

Many of the [fltk2] functions below use fl_utf8decode() and fl_utf8encode() in their own implementation, and are therefore somewhat protected from bad UTF-8 sequences.

The [OksiD] fl_utf8len() function assumes that the byte it is passed is the first byte in a UTF-8 sequence, and returns the length of the sequence. Trailing bytes in a UTF-8 sequence will return -1.

- **WARNING:** fl_utf8len() can not distinguish between single bytes representing Microsoft CP1252 characters 0x80-0x9f and those forming part of a valid UTF-8 sequence. You are strongly advised not to use fl_utf8len() in your own code unless you know that the byte sequence contains only valid UTF-8 sequences.

- **WARNING:** Some of the [OksiD] functions below still use fl_utf8len() in their implementations. These may need further validation.

Please see the individual function description for further details about error handling and return values.

15.4 FLTK Unicode and UTF-8 Functions

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

int fl_utf8locale() FLTK2

fl_utf8locale() returns true if the "locale" seems to indicate that UTF-8 encoding is used.
It is highly recommended that you change your system so this does return true!

```c
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.

```c
int fl_utf_nb_char(const unsigned char *buf, int len) OksiD
```

Returns the number of UTF-8 characters in the first `len` bytes of `buf`.

```c
int fl_unichar_to_utf8_size(Fl_Unichar)
int fl_utf8bytes(unsigned ucs)
```

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

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

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

```c
unsigned int fl_nonspacing(unsigned int ucs) OksiD
```

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

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

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

```c
unsigned int fl_utf8decode(const char *p, const char *end, int *len) FLTK2
int fl_utf8encode(unsigned ucs, char *buf) FLTK2
```
Unicode and UTF-8 Support

`fl_utf8decode()` attempts to decode the UTF-8 character that starts at `p` and may not extend past `end`. It returns the Unicode value, and the length of the UTF-8 character sequence is returned via the `len` argument.

`fl_utf8encode()` writes the UTF-8 encoding of `ucs` into `buf` and returns the number of bytes in the sequence. See the main documentation for the treatment of illegal Unicode and UTF-8 sequences.

```c
unsigned int fl_utf8froma(char *dst, unsigned dstlen, const char *src, unsigned srclen) FLTK2
unsigned int fl_utf8toa(const char *src, unsigned srclen, char *dst, unsigned dstlen) FLTK2
```

`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.

```c
char* fl_utf2mbcs(const char *src) OksiD
```

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

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

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

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

`fl_utf8towc()` converts a UTF-8 string into a "wide character" string. Note that on Windows, some "wide characters" might result in "surrogate pairs" and therefore the return value might be more than the number of characters. `dstlen` specifies the maximum number of `wchar_t` elements to copy, including a zero terminating element. [Is this all worded correctly?]
fl_utf8toUtf16() converts a UTF-8 string into a "wide character" string using UTF-16 encoding to handle the "surrogate pairs" on Windows. dstlen specifies the maximum number of wchar_t elements to copy, including a zero terminating element. [Is this all worded correctly?]

These routines all return the number of elements that would be required for a full conversion of the src string, including the zero terminator. Therefore setting dstlen to zero is a way of measuring how much storage would be needed before doing the real conversion.

unsigned int fl_utf8from_mb(char ∗dst, unsigned dstlen, const char ∗src, unsigned srclen) FLTK2
unsigned int fl_utf8to_mb(const char ∗src, unsigned srclen, char ∗dst, unsigned dstlen) FLTK2

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

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]

int fl_utf_strcasecmp(const char ∗s1, const char ∗s2) OksiD
int fl_utf_strncasecmp(const char ∗s1, const char ∗s2, int n) OksiD

fl_utf_strcasecmp() is a UTF-8 aware string comparison function that converts the strings to lower case Unicode as part of the comparison. flt_utf_strncasecmp() only compares the first n characters [bytes?]
15.5 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 16

Constants and Enumerations

Note

This file is not actively maintained any more, but is left here as a reference, until the doxygen documentation is completed.

See also

FL/Enumerations.H.

This appendix lists the enumerations provided in the `<FL/Enumerations.H>` header file, organized by section. Constants whose value are zero are marked with "(0)", this is often useful to know when programming.

16.1 Version Numbers

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

- `FL_MAJOR_VERSION` - The major release number, currently 1
- `FL_MINOR_VERSION` - The minor release number, currently 4
- `FL_PATCH_VERSION` - The patch release number, currently 0
- `FL_VERSION` - [Deprecated] A combined floating-point version number for the major, minor, and patch 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).
16.2 Events

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

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

The following constants determine when a callback is performed:

- FL_WHEN_NEVER - Never call the callback (0).
- FL_WHEN_CHANGED - Do the callback only when the widget value changes.
- FL_WHEN_NOT_CHANGED - Do the callback whenever the user interacts with the widget.
- FL_WHEN_RELEASE - Do the callback when the button or key is released and the value changes.
- FL_WHEN_ENTER_KEY - Do the callback when the user press 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.

16.4 Fl::event_button() Values

The following constants define the button numbers for FL_PUSH and FL_RELEASE events:

- FL_LEFT_MOUSE - the left mouse button
- FL_MIDDLE_MOUSE - the middle mouse button
- FL_RIGHT_MOUSE - the right mouse button

16.5 Fl::event_key() Values

The following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events:

- FL_Button - A mouse button; use Fl_Button + n for mouse button n.
- FL_BackSpace - The backspace key.
- FL_Tab - The tab key.
- FL_Enter - The enter key.
- FL_Pause - The pause key.
- FL_Scroll_Lock - The scroll lock key.
- FL_Escape - The escape key.
- FL_Home - The home key.
- FL_Left - The left arrow key.
- FL_Up - The up arrow key.
- FL_Right - The right arrow key.
• FL_Down - The down arrow key.
• FL_Page_Up - The page-up key.
• FL_Page_Down - The page-down key.
• FL_End - The end key.
• FL_Print - The print (or print-screen) key.
• FL_Insert - The insert key.
• FL_Menu - The menu key.
• FL_Num_Lock - The num lock key.
• FL_KP - One of the keypad numbers or keys; use FL_KP + 'n' for number n and, say, FL_KP + '*'.
• 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.

16.6  Fl::event_state() Values

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

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

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

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

16.8 Fonts

The following constants define the standard FLTK fonts:

- `FL_HELVETICA` - Helvetica (or Arial) normal (0).
- `FL_HELVETICA_BOLD` - Helvetica (or Arial) bold.
- `FL_HELVETICA_ITALIC` - Helvetica (or Arial) oblique.
- `FL_HELVETICA_BOLD_ITALIC` - Helvetica (or Arial) bold-oblique.
- `FL_COURIER` - Courier normal.
• FL_COURIER_BOLD - Courier bold.
• FL_COURIER_ITALIC - Courier italic.
• FL_COURIER_BOLD_ITALIC - Courier bold-italic.
• FL_TIMES - Times roman.
• FL_TIMES_BOLD - Times bold.
• FL_TIMES_ITALIC - Times italic.
• FL_TIMES_BOLD_ITALIC - Times bold-italic.
• FL_SYMBOL - Standard symbol font.
• FL_SCREEN - Default monospaced screen font.
• FL_SCREEN_BOLD - Default monospaced bold screen font.
• FL_ZAPF_DINGBATS - Zapf-dingbats font.

16.9 Colors

The Fl_Color enumeration type holds a FLTK color value. Colors are either 8-bit indexes into a virtual colormap or 24-bit RGB color values. Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBA.

16.9.1 Color Constants

Constants are defined for the user-defined foreground and background colors, as well as specific colors and the start of the grayscale ramp and color cube in the virtual colormap. Inline functions are provided to retrieve specific grayscale, color cube, or RGB color values.

The following color constants can be used to access the user-defined colors:

• FL_BACKGROUND_COLOR - the default background color
• FL_BACKGROUND2_COLOR - the default background color for text, list, and valuator widgets
• FL_FOREGROUND_COLOR - the default foreground color (0) used for labels and text
• FL_INACTIVE_COLOR - the inactive foreground color
• FL_SELECTION_COLOR - the default selection/highlight color

The following color constants can be used to access the colors from the FLTK standard color cube:

• FL_BLACK
• FL_BLUE
• FL_CYAN
• FL_DARK_BLUE
• FL_DARK_CYAN
16.10 Cursors

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.

16.10 Cursors

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

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

- **FL_READ** - Call the callback when there is data to be read.
- **FL_WRITE** - Call the callback when data can be written without blocking.
- **FL_EXCEPT** - Call the callback if an exception occurs on the file.

16.12 Damage Masks

The following damage mask bits are used by the standard FLTK widgets:

- **FL_DAMAGE_CHILD** - A child needs to be redrawn.
- **FL_DAMAGE_EXPOSE** - The window was exposed.
- **FL_DAMAGE_SCROLL** - The Fl_Scroll widget was scrolled.
- **FL_DAMAGE_OVERLAY** - The overlay planes need to be redrawn.
- **FL_DAMAGE_USER1** - First user-defined damage bit.
- **FL_DAMAGE_USER2** - Second user-defined damage bit.
- **FL_DAMAGE_ALL** - Everything needs to be redrawn.
Chapter 17

GLUT Compatibility

This appendix describes the GLUT compatibility header file supplied with FLTK.

FLTK's GLUT compatibility is based on the original GLUT 3.7 and the follow-on FreeGLUT 2.4.0 libraries.

17.1 Using the GLUT Compatibility Header File

You should be able to compile existing GLUT source code by including <FL/glut.H> instead of <GL/glut.h>. This can be done by editing the source, by changing the -I switches to the compiler, or by providing a symbolic link from GL/glut.h to FL/glut.H.

All files calling GLUT procedures must be compiled with C++. You may have to alter them slightly to get them to compile without warnings, and you may have to rename them to get make to use the C++ compiler.

You must link with the FLTK library. Most of FL/glut.H is inline functions. You should take a look at it (and maybe at test/glpuzzle.cxx in the FLTK source) if you are having trouble porting your GLUT program.

This has been tested with most of the demo programs that come with the GLUT and FreeGLUT distributions.

17.2 Known Problems

The following functions and/or arguments to functions are missing, and you will have to replace them or comment them out for your code to compile:

- glutGet (GLUT_ELAPSED_TIME)
- glutGet (GLUT_SCREEN_HEIGHT_MM)
- glutGet (GLUT_SCREEN_WIDTH_MM)
- glutGet (GLUT_WINDOW_NUM_CHILDREN)
- glutInitDisplayMode (GLUT_LUMINANCE)
- glutKeyboardUpFunc (void (*callback)(unsigned char key, int x, int y))
- glutLayerGet (GLUT_HAS_OVERLAY)
• glutLayerGet(GLUT_LAYER_IN_USE)
• glutPushWindow()
• glutSetColor(), glutGetColor(), glutCopyColormap()
• glutVideoResize() missing.
• glutWarpPointer()
• glutWindowStatusFunc()
• Spaceball, buttonbox, dials, and tablet functions

Most of the symbols/enumerations have different values than GLUT uses. This will break code that relies on the actual values. The only symbols guaranteed to have the same values are true/false pairs like GLUT_DOWN and GLUT_UP, mouse buttons GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON, and GLUT_KEY_F1 thru GLUT_KEY_F12.

The strings passed as menu labels are not copied.

glutPostRedisplay() does not work if called from inside a display function. You must use glutIdleFunc() if you want your display to update continuously.

glutSwapBuffers() does not work from inside a display function. This is on purpose, because FLTK swaps the buffers for you.

glutUseLayer() does not work well, and should only be used to initialize transformations inside a resize callback. You should redraw overlays by using glutOverlayDisplayFunc().

Overlays are cleared before the overlay display function is called. glutLayerGet(GLUT_OVERLAY_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").

17.3 Mixing GLUT and FLTK Code

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

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

### Class Hierarchy
```
Fl_Gl_Window
   +----Fl_Glut_Window
```

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

### 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().
```

### 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>

Generated by Doxygen
17.4.5 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 18

Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

**Warning:** The Forms compatibility is deprecated and no longer maintained since FLTK 1.3.0 and is likely to be removed completely in FLTK 1.4 or 1.5

18.1 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>`.

18.2 Using the Compatibility Header File

You should be able to compile existing Forms or XForms source code by changing the include directory switch to your compiler so that the `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.
18.3 Problems You Will Encounter

Many parts of XForms use X-specific structures like XEvent in their interface. I did not emulate these! Unfortunately these features (such as the "canvas" widget) are needed by most large programs. You will need to rewrite these to use FLTK subclasses.

**Fl_Free** widgets emulate the old Forms "free" widget. It may be useful for porting programs that change the handle() function on widgets, but you will still need to rewrite things.

**Fl_Timer** widgets are provided to emulate the XForms timer. These work, but are quite inefficient and inaccurate compared to using Fl::add_timeout().

All instance variables are hidden. If you directly refer to the x, y, w, h, label, or other fields of your Forms widgets you will have to add empty parenthesis after each reference. The easiest way to do this is to globally replace ":x" with ":x()", etc. Replace ":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()
18.4 Additional Notes

These notes were written for porting programs written with the older IRISGL version of Forms. Most of these problems are the same ones encountered when going from old Forms to XForms:

**Does Not Run In Background**

The IRISGL library always forked when you created the first window, unless "foreground()" was called. FLTK acts like "foreground()" is called all the time. If you really want the fork behavior do "if (fork()) exit(0)" right at the start of your program.
You Cannot Use IRISGL Windows or fl_queue

If a Forms (not XForms) program if you wanted your own window for displaying things you would create a IRISGL window and draw in it, periodically calling Forms to check if the user hit buttons on the panels. If the user did things to the IRISGL window, you would find this out by having the value FL_EVENT returned from the call to Forms.

None of this works with FLTK. Nor will it compile, the necessary calls are not in the interface.

You have to make a subclass of Fl_Gl_Window and write a draw() method and handle() method. This may require anywhere from a trivial to a major rewrite.

If you draw into the overlay planes you will have to also write a draw_overlay() method and call redraw_overlay() on the OpenGL window.

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

You Must Use OpenGL to Draw Everything

The file <FL/gl.h> defines replacements for a lot of IRISGL calls, translating them to OpenGL. There are much better translators available that you might want to investigate.

You Cannot Make Forms Subclasses

Programs that call fl_make_object or directly setting the handle routine will not compile. You have to rewrite them to use a subclass of Fl_Widget. It is important to note that the handle() method is not exactly the same as the handle() function of Forms. Where a Forms handle() returned non-zero, your handle() must call do_callback(). And your handle() must return non-zero if it "understood" the event.

An attempt has been made to emulate the "free" widget. This appears to work quite well. It may be quicker to modify your subclass into a "free" widget, since the "handle" functions match.

If your subclass draws into the overlay you are in trouble and will have to rewrite things a lot.

You Cannot Use <device.h>

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

<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>
Anything else in `getvaluator` and you are on your own...

Font Numbers Are Different

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

```c
fl_font_name(3, "*courier-medium-r-no*");
fl_font_name(4, "*courier-bold-r-no*");
fl_font_name(5, "*courier-medium-o-no*");
fl_font_name(6, "*times-medium-r-no*");
fl_font_name(7, "*times-bold-r-no*");
fl_font_name(8, "*times-medium-i-no*");
fl_font_name(9, "*bookman-light-r-no*");
fl_font_name(10, "*bookman-demi-r-no*");
fl_font_name(11, "*bookman-light-i-no*");
```
Chapter 19

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
- The Wayland Interface

19.1 Accessing the OS Interfaces

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

```
#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.
19.2 The UNIX (X11) Interface

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

19.2.1 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().

Deprecated Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window ∗) with versions 1.4 and above.

Fl_Window ∗fl_find(ulong xid)
19.2 The UNIX (X11) Interface

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

**Deprecated** Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.

int fl_handle(const XEvent &)

This call allows you to supply the X events to FLTK, which may allow FLTK to cooperate with another toolkit or library. The return value is non-zero if FLTK understood the event. If the window does not belong to FLTK and the add_handler() functions all return 0, this function will return false.

Besides feeding events your code should call Fl::flush() periodically so that FLTK redraws its windows.

This function will call the callback functions. It will not return until they complete. In particular, if a callback pops up a modal window by calling fl_ask(), for instance, it will not return until the modal function returns.

19.2.2 Drawing using Xlib

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

```
extern Display *fl_display; // for compatibility with previous FLTK versions
extern Display *fl_x11_display(); // preferred access starting with FLTK 1.4
extern Window fl_window;
extern GC fl_gc; // for compatibility with previous FLTK versions
extern GC fl_x11_gc(); // preferred access starting with FLTK 1.4
extern int fl_screen;
extern XVisualInfo *fl_visual;
extern Colormap fl_colormap;
```

You must use them to produce Xlib calls. Don’t attempt to change them. A typical X drawing call is written like this:

```
XDrawSomething(fl_display, fl_window, fl_gc, ...);
```

Other information such as the position or size of the X window can be found by looking at Fl_Window::current(), which returns a pointer to the Fl_Window being drawn.

```
unsigned long fl_xpixel(Fl_Color i)
unsigned long fl_xpixel(uchar r, uchar g, uchar b)
```

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

```
int fl_parse_color(const char ∗p, uchar & r, uchar & g, uchar & b)
```

Convert a name into the red, green, and blue values of a color by parsing the X11 color names. On other systems, fl_parse_color() can only convert names in hexadecimal encoding, for example #ff8083.

```
extern XFontStruct *fl_xfont
```

Points to the font selected by the most recent fl_font(). This is not necessarily the current font of fl_gc, which is not set until fl_draw() is called. If FLTK was compiled with Xft support, fl_xfont will usually be 0 and fl_xftfont will contain a pointer to the XftFont structure instead.

```
extern void *fl_xftfont
```

If FLTK was compiled with Xft support enabled, fl_xftfont points to the xft font selected by the most recent fl_font(). Otherwise it will be 0. fl_xftfont should be cast to XftFont*.
19.2.3 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);
19.2.4 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.

```c
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.
Operating System Issues

The X region that is a combination of all damage() calls done so far is in Fl_X::i(this)->region. If NULL then you should redraw the entire window. The undocumented function fl_clip_region(←XRegion) will initialize the FLTK clip stack with a region or NULL for no clipping. You must set region to NULL afterwards as fl_clip_region() will own and delete it when done.

If damage() & FL_DAMAGE_EXPOSE then only X expose events have happened. This may be useful if you have an undamaged image (such as a backing buffer) around.

Here is a sample where an undamaged image is kept somewhere:
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:
void MyWindow::hide() {  
  if (mypixmap) {  
    XFreePixmap(fl_display,mypixmap);  
    mypixmap = 0;  
  }  
  Fl_Window::hide(); // you must call this
}

virtual void Fl_Window::~Fl_Window()

Because of the way C++ works, if you override hide() you must override the destructor as well (otherwise only the base class hide() is called):
MyWindow::MyWindow() {  
  hide();  
}

Note
Access to the Fl_X hidden class requires to #define FL_INTERNALS before compilation.

19.2.5 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: Fl_Window::icon(const Fl_RGB_Image *) and Fl_Window::icons(const Fl_RGB_Image *[], int). See also methods setting default window icons Fl_Window::default_icon(const Fl_RGB_Image *) and Fl_Window::default_icons(const Fl_RGB_Image *[], int).

FLTK on X11 also supports, for backward compatibility, use of the deprecated method Fl_Window::icon(const void *) as follows:
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:
#include "icon.xbm"
To use a multi-colored icon, the XPM format and library should be used as follows:

```
#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`).

Any window icon must be set with the above methods before the window is shown.

### 19.2.6 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`).
19.2.7 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.

19.3 The Windows Interface

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

19.3.1 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.

19.3.2 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.

19.3.3 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.
19.3.4 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:

```
extern HINSTANCE fl_display; // for compatibility with previous FLTK versions
extern HINSTANCE fl_win32_display(); // preferred access starting with FLTK 1.4
extern HWND fl_window;
extern HDC fl_gc; // for compatibility with previous FLTK versions
extern HDC fl_win32_gc(); // preferred access starting with FLTK 1.4
COLORREF fl_RGB();
HPEN fl_pen();
HBRUSH fl_brush();
```

These global variables are set before `Fl_Widget::draw()` is called, or by `Fl_Window::make_current()`. You can refer to them when needed to produce GDI calls, but don’t attempt to change them. The functions return GDI objects for the current color set by `fl_color()` and are created as needed and cached. A typical GDI drawing call is written like this:

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

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

19.3.5 HighDPI support

FLTK apps for the Windows platform are by default "Per-monitor DPI-aware V2". This means that any window automatically adjusts its physical size in relation to the scaling factor of the display where it maps. This also means that all drawings (e.g., text, lines, images) take advantage of the full resolution of the display in use. FLTK apps may also use the manifest mechanism to declare their level of DPI awareness. The FLTK library adapts to the DPI awareness level set in the app’s manifest, which can be lower than the default level if the manifest sets it so.

19.3.6 Display Scaling Factor

FLTK uses the value given by function `GetDpiForMonitor()` divided by 96. to initialize the scaling factor of each display in the system. This matches the value of "Change the size of text, apps and other items" found in section "System" subsection "Display" of Windows settings.

19.3.7 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: `Fl_Window::icon(const Fl_RGB_Image *)` and `Fl_Window::icons(const Fl_RGB_Image *[ ], int)`. See also methods setting default window icons `Fl_Window::default_icon(const Fl_RGB_Image *)` and `Fl_Window::default_icons(const Fl_RGB_Image *[ ], int)`.

FLTK on Windows also supports, for backward compatibility, use of the deprecated method `Fl_Window::icon(const void *)` as follows:

```
Set the icon for the window to the passed pointer. You will need to cast the HICON handle to a char * when calling this method. To set the icon using an icon resource compiled with your application use:
window->icon((const void *)LoadIcon(fl_display, MAKEINTRESOURCE(IDI_ICON)));
```

You can also use the `LoadImage()` and related functions to load specific resolutions or create the icon from bitmap data.

**NOTE:**
You must call `Fl_Window::show(int argc, char ** argv)` for the icon to be used. The `Fl_Window::show()` method does not bind the icon to the window.
Any window icon must be set with the above methods before the window is shown.

19.3.8 How to Not Get a MSDOS Console Window

Windows has a really stupid mode switch stored in the executables that controls whether or not to make a console window.

To always get a console window you simply create a console application (the "\SUBSYSTEM:CONSOLE" option for the linker). For a GUI-only application create a Windows application (the "\SUBSYSTEM:WINDOWS" option for the linker).

FLTK includes a WinMain() function that calls the ANSI standard main() entry point for you. This function creates a console window when you use the debug version of the library.

Windows applications without a console cannot write to stdout or stderr, even if they are run from a console window. Any output is silently thrown away. Additionally, Windows applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.

19.3.9 Known Windows Bugs and Problems

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

- If a program is deactivated, Fl::wait() does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in Windows and probably impossible to get around.

- Fl_Gl_Window::can_do_overlay() returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.

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

- Compilation with gcc 3.4.4 and -Os exposes an optimisation bug in gcc. The symptom is that when drawing filled circles only the perimeter is drawn. This can for instance be seen in the symbols demo. Other optimisation options such as -O2 and -O3 seem to work OK. More details can be found in STR#1656
19.4 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 Y en 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 \texttt{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, \texttt{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 \texttt{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 \texttt{fl_open_callback()} function.

\begin{verbatim}
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 \texttt{show()} of a window.
\end{verbatim}

\begin{verbatim}
Fl_Window *fl_xid(const Fl_Window *)

Returns the window reference for an \texttt{Fl_Window}, or \texttt{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.
\end{verbatim}

\begin{verbatim}
Fl_Window *fl_find(Window xid)

Returns the \texttt{Fl_Window} that corresponds to the given window reference, or \texttt{NULL} if not found.
\end{verbatim}

\begin{verbatim}
void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut)

Attaches the callback \texttt{cb} to the "About myprog" item of the system application menu. \texttt{cb} will be called with NULL first argument and \texttt{user_data} second argument. This MacOS-specific function is deprecated in FLTK 1.4 and replaced by \texttt{Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *data)} which is cross-platform.
\end{verbatim}

\begin{verbatim}
Fl_Sys_Menu_Bar class

The \texttt{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.
\end{verbatim}

19.4.1 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

\begin{verbatim}
<key>CFBundleIconFile</key>
<string>foo.icns</string>
\end{verbatim}

replacing \texttt{foo} by your application name. If you use Xcode, just add your .icns file to your application target.
19.4.2 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.

19.4.3 Internationalization

All FLTK programs contain an application menu with, e.g., the About xxx, Hide xxx, and Quit xxx items. This menu can be internationalized/localized by any of two means.

• using the `Fl_Mac_App_Menu` class.
• using the standard Mac OS X localization procedure. Create a language-specific .lproj directory (e.g., `German.lproj`) in the Resources subdirectory of the application bundle. Create therein a `Localizable.strings` file that translates all menu items to this language. The German `Localizable.strings` file, for example, contains:

  "About %@" = "Über %@";
  "Print Front Window"="Frontfenster drucken";
  "Services" = "Dienste";
  "Hide %@"="%@ 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.

19.4.4 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.
19.4.5 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.

19.4.6 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

19.5 The Wayland Interface

See file README.Wayland.txt for details about how to organize platform-specific source code for the Wayland platform.

extern struct wl_display *fl_wl_display();

After fl_open_display() has run, function fl_wl_display() returns a pointer to the struct wl_display representing the connection between the application and Wayland. For example, wl_display_get_fd(fl_wl_display()) gives the file descriptor one can use to communicate with the Wayland compositor according to the Wayland protocol.

struct wld_window *fl_wl_xid(const Fl_Window *)
Returns a pointer to an **FLTK-defined** structure holding Wayland-related data created when a window gets show()'n, or NULL if not show()'n.

**Fl_Window** ∗fl_wl_find(struct wld_window ∗wld_win)

Returns the **Fl_Window** that corresponds to the given Window, or NULL if not found.

**struct wl_surface** ∗fl_wl_surface(struct wld_window ∗wld_win)

Returns a pointer to the **struct wl_surface** corresponding to a show()'n top-level window or subwindow.

**cairo_t** ∗fl_wl_cairo(void)

Drawing natively to a Wayland window: Within an overridden **Fl_Widget::draw()** method, or after a call to **Fl_Window::make_current()**, it's possible to draw using the Cairo library. Function **fl_wl_cairo()** returns the adequate **cairo_t** value. Regular FLTK coordinates, with top-left origin, are to be used. All FLTK-defined drawing functions (e.g., **fl_rect()**, **fl_draw()**) can be used too.

**void fl_close_display()**

This closes the Wayland connection. You do not need to call this to exit. It may be useful to call this if you want your program to continue without the Wayland connection. You cannot open the display again, and cannot call any FLTK functions.

### 19.5.1 HiDPI display support

FLTK Wayland apps automatically scale according to the Wayland-defined, integer-valued scale factor. On a HiDPI display, it's enough to set this factor to 2 for any FLTK app to be drawn using twice as many pixels and thus to be as readable as it is on a regular display. With the gnome desktop, that is achieved in the "Displays" section of the "Settings" application, selecting 200 % for the "Scale" parameter. In addition to this, FLTK apps can also be scaled up or down typing **ctrl/+/-/0/** and with the **FLTK_SCALING_FACTOR** environment variable.

### 19.5.2 Window icons

Standard FLTK functions **Fl_Window::icon(const Fl_RGB_Imag**e** ∗), Fl_Window::icons(const Fl_RGB_Imag**e** ∗[], int), Fl_Window::default_icon(const Fl_RGB_Imag**e** ∗), and Fl_Window::default_icons(const Fl_RGB_Imag**e** ∗[], int) have no effect on the Wayland platform. The equivalent of a call to **Fl_Window::default_icon(const Fl_RGB_Imag**e** ∗) to set the application-specific window icon can be obtained as follows, using FLTK's editor app as an example:

- create a text file named **editor.desktop** containing:

  [Desktop Entry]
  Version=1.0
  Type=Application
  Name=Editor
  Name[fr]=Editeur
  Comment=FLTK editor
  Exec=editor %F
  Icon=/path/to/icon/file/editor.svg
  MimeType=text/plain

- The **Name** line therein determines the string displayed when the app runs.
- Optionally, one or more **Name[locale]** = lines can be used to set locale-specific app names.
- The **Icon** line accepts also .png files.
- Put this file in **/usr/local/share/applications/** so it's available to all system users or in **$HOME/.local/share/applications/** so it's available to a single user.
19.5.3 Window titlebars

Wayland supports both client-side window decoration (CSD), where client applications are responsible for drawing window titlebars, and server-side window decoration (SSD), where the Wayland compositor itself draws window titlebars. Among 3 tested Wayland compositors, Mutter (gnome's compositor) and Weston use CSD mode whereas the KDE compositor uses SSD mode. When running in CSD mode, FLTK uses a library called libdecor to draw titlebars. The libdecor library has been conceived to use various plug-in's to draw titlebars in various fashions intended to match any desktop's preferred titlebar style. FLTK supports drawing titlebars with any libdecor plug-in via an environment variable called LIBDECOR_PLUGIN_DIR which can be given the name of a directory containing the desired plug-in. When LIBDECOR_PLUGIN_DIR is not defined, or points to a directory that doesn't contain a libdecor plug-in, FLTK uses its built-in plug-in to draw titlebars. That is the most common situation, until libdecor plug-in's become available for popular UNIX desktops.
Chapter 20

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.

20.1 Migrating from FLTK 1.0 or 1.1 to 1.4

If you need 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.

20.2 Changes in Header Files

We strive to include only necessary header files in the public headers of the FLTK library to reduce dependencies and hence compile times.

We try to avoid including system header files as far as possible. Known exceptions are `<stdio.h>` where file system structures and functions are visible in the public API, for instance `FILE*`, and sometimes essential header files like `<stdlib.h>` and/or `<stddef.h>`. Some required platform headers may be included in platform specific header files like `<FL/platform.H>` or `<FL/platform_types.h>`.

In earlier versions (1.3.x) some of the public FLTK headers included some not strictly required system headers by accident.

The consequence for building user programs with FLTK 1.4 is that if you require a system or FLTK header in your user program that you don't #include explicitly but which has been included by FLTK 1.3.x your FLTK 1.3 program may issue compiler errors or warnings about missing header files or missing declarations when compiled with FLTK 1.4.

This is not a fault of FLTK 1.4 but a fault of the source code that did not include all required headers.

Suggested solution: include all FLTK and system header files your source code requires explicitly and don't rely on FLTK headers to include a particular header file.

The same applies to FLTK headers. The rule is to #include `<FL/Fl.H>` as the first FLTK header as described in the documentation elsewhere and to include FLTK headers for all classes you are using explicitly. You don't need to include headers of base classes - this is done by all FLTK headers as required. Besides that you need to include some support headers if you are using FLTK functions like `fl_choice()` and others. This is described in the function's documentation (if a required header is missing in the docs this is a bug).

If you follow these rules your program will be compatible with both FLTK 1.3.x and FLTK 1.4.x as long as you use only functions and classes defined in FLTK 1.3.
20.3 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 preference files has changed slightly. Please see `Fl_Preferences::Fl_Preferences(Root, const char*, const char*)` for details.

On Unix/Linux platforms new FLTK preference files are stored using the XDG Base Directory Specification which means in essence that user preference files are stored in the user's home directory under the subdirectory `.config`, i.e. in `$HOME/.config/fltk.org/` rather than `$HOME/.fltk/fltk.org/`. Existing preference files are still found and used, hence this new location is optional.

You may want to move the preference files from their old locations to their new locations as documented in `Fl_Preferences::Fl_Preferences(Root, const char*, const char*)`.

New Fl_Preferences types `Fl_Preferences::USER_L`, `Fl_Preferences::SYSTEM_L` and some more combinations with "_L" suffix have been defined to make preference files independent of the current locale. This is particularly important for floating point data which is stored in text form with varying decimal separator depending on the locale (either '.' or','). You may want to change your program to use these new constants instead of those without the "_L" suffix. For more information see the documentation of Fl_Preferences.

20.4 Fl::add_timeout and friends

Since FLTK 1.4.0 internal timeout handling has been unified across platforms. This ensures equal timeout handling, improved accuracy of Fl::repeat_timeout(), and easier maintenance (less potential for errors).

This will very likely not affect user code, however there is one subtle exception on macOS and Windows: in FLTK 1.3.x these platforms used system timers to schedule timeouts. Since FLTK 1.4.0 all platforms use the same internal timer management that was previously only used on Unix/Linux/X11. The consequence of this change is that the FLTK event loop needs to be executed to trigger timeout events, i.e. you must either call Fl::wait() repeatedly or start the event loop with Fl::run().

Code that did not execute the event loop and relied on the system timers has never been cross platform compatible, i.e. it wouldn't work on Unix/Linux. An example would be code that opened a splash window, scheduled a timeout with Fl::add_timeout(), and waited for the timer event w/o running the FLTK event loop. Such code must be modified to execute Fl::run() and/or use Fl::wait().

20.5 Fl_Image::copy() 'const'

Since FLTK 1.4.0 the virtual method Fl_Image::copy() has been declared 'const' so read-only ('const') images can be copied w/o casts.

This will very likely not affect user code. However, if you derived your own class from any of the Fl_+_image variants and you overrode 'Your'_Image::copy() then you must declare this 'const' as well, i.e. you must add the keyword 'const' to your declaration of copy() in your header file and the implementation.

Code example in header file:
```
class Your_Image {
    // ...
    copy() const;
    copy(int w, int h) const;
};
```
Chapter 21

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 \code{Fl_Clock} widget using the given boxtype, position, size, and label string.

\verbatim
\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
\endverbatim

\code{Fl_Clock::Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L)}

\:
\code{Fl_Clock_Output(X, Y, W, H, L)}

\verbatim

\verbatim
\type(t);
\box(t==FL_ROUND_CLOCK ? FL_NO_BOX : FL_UP_BOX);
\endverbatim

Note

From Duncan: (will be removed later, just for now as a reminder)

I’ve just added comments for the \code{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 \code{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 \code{functions.dox} file that defines a functions group and do that for all of the function documentation?)

Update: the trick is not to create duplicate entries in a new group, but to move the function information into the doxygen comments for the class, and use the navigation links provided. Simply using 'relatesalso' as the first doxygen command in the function’s comment puts it in the appropriate place. There is no need to have 'defgroup' and 'ingroup' as well, and indeed they don’t work. So, to summarize:

\code{Gizmo.H}

\verbatim
/** 
   \class Gizmo
   
   A gizmo that does everything
   */

class Gizmo {

    etc
};

extern int popup_gizmo(...);
\endverbatim

\code{Gizmo.cxx}

\verbatim
/** 
   \relatesalso Gizmo
   
   Pops up a gizmo dialog with a Gizmo in it
   */

int popup_gizmo(...);
\endverbatim

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.

\verbatim
<!-- *** This *** is *** invisible *** -->
\endverbatim

This will be visible again.

will be shown as:

The following text is a developer comment.

\verbatim
<!-- *** This *** is *** invisible *** -->
\endverbatim

This will be visible again.
21.1 Non-ASCII Characters

Different Headlines

You can use HTML tags `<H1> ... <H4>` for headlines with different sizes. As of doxygen 1.8.x there must not be more than three spaces at the beginning of the line for this to work. Currently (doxygen 1.8.6) there seems to be no difference in the font sizes of `<H3>` and `<H4>` in the pdf output, whereas the html output uses different font sizes.

```html
<H1>Headline in big text (H1)</H1>
<H2>Headline in big text (H2)</H2>
<H3>Headline in big text (H3)</H3>
<H4>Headline in big text (H4)</H4>
```

Headline in big text (H1)

Headline in big text (H2)

Headline in big text (H3)

Headline in big text (H4)

21.1 Non-ASCII Characters

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

This will appear in the document:

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

For further informations about HTML quoting characters see [http://www.doxygen.org/manual/htmlcmds.html](http://www.doxygen.org/manual/htmlcmds.html)

Alternatively you can use UTF-8 encoding within Doxygen comments.

21.2 Document Structure

- `\page` creates a named page
- `\section` creates a named section within that page
- `\subsection` creates a named subsection within the current section
- `\subsubsection` creates a named subsubsection within the current subsection
All these statements take a "name" as their first argument, and a title as their second argument. The title can contain spaces.

The page, section, and subsection titles are formatted in blue color and a size like "<H1>"", "<H2>"", and "<H3>"", and "<H4>"", respectively.

By FLTK documentation convention, a file like this one with a doxygen documentation chapter has the name "<chapter>.dox". The \page statement at the top of the page is "page <chapter> This is the title". Sections within a documentation page must be called "<chapter>_<section>", where "<chapter>" is the name part of the file, and "<section>" is a unique section name within the page that can be referenced in links. The same for subsections and subsubsections.

These doxygen page and section commands work only in special documentation chapters, not within normal source or header documentation blocks. However, links from normal (e.g. class) documentation to documentation sections do work.

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

## 21.3 Creating Links

Links to other documents and external links can be embedded with

- doxygen \ref links to other doxygen \page, \section, \subsection and \anchor locations
- HTML links without markup - doxygen creates "http://..." links automatically
- standard, non-Doxygen, HTML links
  - see chapter \ref unicode creates a link to the named chapter unicode that has been created with a \page statement.
  - For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
  - see <a href="https://www.fltk.org/">FLTK Library</a> creates a standard HTML link

appears as:

- see chapter Unicode and UTF-8 Support creates a link to the named chapter unicode that has been created with a \page statement.

- For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html

- see FLTK Library creates a standard HTML link

Generated by Doxygen
21.4  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.

```latex
\par Normal Paragraph with title
This paragraph will have a title, but because there is a blank line between the `\par` and the text, it will have the normal layout.

\par Indented Paragraph with title
This paragraph will also have a title, but because there is no blank line between the `\par` and the text, it will be indented.

\par
It is also possible to have an indented paragraph without title. This is how you indent subsequent paragraphs.

\par No link to Fl_Widget::draw()
Note that the paragraph title is treated as plain text. Doxygen type links will not work. HTML characters and tags may or may not work.

Fl_Widget::draw() links and &quot;html&quot; tags work

\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.
21.5 Navigation Elements

Each introduction (tutorial) page ends with navigation elements. These elements must only be included in the html documentation, therefore they must be separated with \htmlonly and \endhtmlonly.

The following code gives the navigation bar at the bottom of this page:

\htmlonly
<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 22

Software License

December 11, 2001

The FLTK library and included programs are provided under the terms of the GNU Library General Public License (LGPL) with the following exceptions:

1. Modifications to the FLTK configure script, config header file, and makefiles by themselves to support a specific platform do not constitute a modified or derivative work.

   The authors do request that such modifications be contributed to the FLTK project - send all contributions through the "Software Trouble Report" on the following page: https://www.fltk.org/bugs.php

2. Widgets that are subclassed from FLTK widgets do not constitute a derivative work.

3. Static linking of applications and widgets to the FLTK library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared FLTK libraries, or link their applications or widgets against a user-supplied version of FLTK.

   If you link the application or widget to a modified version of FLTK, then the changes to FLTK must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs that are linked to the FLTK library, nor do you have to identify the FLTK license in your program or documentation as required by section 6 of the LGPL.

   However, programs must still identify their use of FLTK. The following example statement can be included in user documentation to satisfy this requirement:

   [program/widget] is based in part on the work of the FLTK project ( https://www.fltk.org).
GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991
Copyright (C) 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary
GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the
GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make
sure the software is free for all its users.
This license, the Library General Public License, applies to some specially designated Free Software Foundation
software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed
to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish),
that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new
free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to
surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the
library, or if you modify it.
For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the
rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program
with the library, you must provide complete object files to the recipients so that they can relink them with the library,
after making changes to the library and recompiling it. And you must show them these terms so they know their
rights.
Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which
produces from the ordinary GPL.
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 propri-
etary 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.

Generated by Doxygen
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 apply only in those countries where the conditions described below are not considered to be incompatible with the requirement of a free software. In such cases, this section is intended to apply in whatever ways may be considered compatible.

This section can be added to any version of the License except those versions that are covered by section 13. When you add these sections to a version of the License, each of your works based on that version will be covered by the terms of the added section.

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.

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 23

Example Source Code

The FLTK distribution contains over 60 sample applications written in, or ported to, FLTK. If the FLTK archive you received does not contain either an 'examples' or 'test' directory, you can download the complete FLTK distribution from https://www.fltk.org/software.php. Most of the example programs were created while testing a group of widgets. They are not meant to be great achievements in clean C++ programming, but merely a test platform to verify the functionality of the FLTK library. Note that extra example programs are also available in an additional 'examples' directory, but these are NOT built automatically when you build FLTK, unlike those in the 'test' directory shown below.

23.1 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>boxtyp</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>gpuzzle</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>

23.1.1 adjuster

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

23.1.2 animated

animated shows a window with an animated square that shows drawing with transparency (alpha channel).

23.1.3 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.

### 23.1.4 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...

### 23.1.5 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.

### 23.1.6 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.

### 23.1.7 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.

### 23.1.8 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.

### 23.1.9 button

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

### 23.1.10 buttons

`buttons` shows a sample of FLTK button types.

### 23.1.11 cairo_test

`cairo_test` shows a sample of drawing with Cairo in an `Fl_Cairo_Window`. This program can only be built if FLTK was configured with Cairo support.

### 23.1.12 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.
23.1.13 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.

23.1.14 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.

23.1.15 color_chooser

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

23.1.16 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.

23.1.17 CubeView

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

23.1.18 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).

23.1.19 curve

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

23.1.20 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.

23.1.21 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().
23.1.22 doublebuffer

The doublebuffer demo shows the difference between a single buffered window, which may flicker during a slow redraw, and a double buffered window, which never flickers, but uses twice the amount of RAM. Some modern OS's double buffer all windows automatically to allow transparency and shadows on the desktop. FLTK is smart enough to not triple buffer a window in that case.

23.1.23 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.

23.1.24 fast_slow

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

23.1.25 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.

23.1.26 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.

23.1.27 forms

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

23.1.28 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.

23.1.29 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.

23.1.30 gl_overlay

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

23.1.31 glpuzzle
23.1 Example Applications: Overview

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

23.1.32 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.

23.1.33 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.

23.1.34 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).

23.1.35 iconize

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

23.1.36 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).

23.1.37 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.

23.1.38 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().

23.1.39 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.

23.1.40 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.
23.1.41 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`.

23.1.42 line_style

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

23.1.43 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.

23.1.44 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.

23.1.45 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.

23.1.46 message

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

23.1.47 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.

23.1.48 native-filechooser

The native-filechooser program invokes the platform specific file chooser, if available (see Fl_Native_File_Chooser widget).

23.1.49 navigation

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

23.1.50 offscreen

offscreen shows how to draw into an offscreen image and display the offscreen image in the program window.
23.1.51 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.

23.1.52 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.

23.1.53 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.

23.1.54 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.

23.1.55 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.

23.1.56 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.

23.1.57 radio

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

23.1.58 resizebox

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

23.1.59 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.

23.1.60 resize

The resize demo tests size and position functions with the given window manager.
23.1.61 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.

23.1.62 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.

23.1.63 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.

23.1.64 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.

23.1.65 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.).

23.1.66 table

The table demo shows the features of the Fl_Table widget.

23.1.67 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.

23.1.68 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.

23.1.69 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.
23.1.70  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.

23.1.71  tree

The tree demo shows the features of the Fl_Tree widget.

23.1.72  twowin

The twowin program tests focus transfer from one window to another window.

23.1.73  unittests

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

23.1.74  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.

23.1.75  valuators

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

23.1.76  windowfocus

windowfocus shows a very special case when a new window is shown while the focus stays in the original window.

23.1.77  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.

23.2  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.

23.2.1  cairo_test
The `cairo_test` demo program shows three shiny buttons drawn with Cairo in an `FL_Cairo_Window`.

![Figure 23.1 Buttons drawn with Cairo](image1)

### 23.2.2 icon

The `icon` program lets you set the program icon from an image (here an `FL_RGB_Image`).

![Figure 23.2 Green icon (Windows 10)](image2)

### 23.2.3 unitests
Select "drawing images" in the browser at the left side to see the image drawing example:
Chapter 24

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?

24.1 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.

24.2 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.

24.3 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).

### 24.4 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:

```c
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:

```c
void my_callback(Fl_Widget*, void*) {
    if (Fl::choice("Are you sure you want to quit?", "continue", "quit", NULL))
        exit(0);
}
```
Chapter 25
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 box types 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 (Fl_Color c)
  Remove Fl_File_Input::errorcolor(Fl_Color) in FLTK 1.5.0 or higher.

Member Fl_File_Input::errorcolor () const
  Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

Member fl_filename_list (const char *d, struct dirent ***, Fl_File_Sort_F *s=fl_numericsort)
  should support returning OS error messages

Member Fl_Group::delete_child (int n)
  Reimplementation of Fl_Group::delete_widget(int) in more FLTK subclasses. This is not yet complete.
Member `fl_height` (int font, int size)

In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

Member `Fl_Help_View::find` (const char *s, int ps=0)

complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and
do not (yet) match correctly ("<" matches "<" but "Œ" doesn't match "Œ", and "ü" doesn't match "ü")

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)

Raw integer shortcut needs examples. Dependent on responses to https://www.fltk.org/news.php?g=fltk.coredev+v:10086 and results of STR#2344

Member `fl_numericsort` (struct dirent **A, struct dirent **B)

Make comparison UTF-8 aware.

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 in-
tended. This would be possible if we could rely on all programs being coded with symbolic names and not hard
coded bit values.

Class `Fl_String`

Complete documentation of class `Fl_String`

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 (int width)
This method should eventually be removed

Member Fl_Text_Display::scrollbar_width () const
This method should eventually be removed.

Member Fl_Text_Display::shortcut (int s)
FIXME : get set methods pointing on shortcut__ have no effects as shortcut__ is unused in this class and derived!

Member Fl_Text_Display::shortcut () const
FIXME : get set methods pointing on shortcut__ have no effects as shortcut__ is unused in this class and derived!

Member Fl_Text_Display::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
doxxygen 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 FLTK Basics
This section needs a major rework. Add a chapter "Building FLTK with CMake".

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

Verify 16/24 bit Unicode limit for different character sets? OksiD’s code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in fl_utf8fromwc() and fl_ut8toUtf16().

Work through the code and this documentation to harmonize the [OksiD] and [fltk2] functions.

FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus (FL_Shortcut) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.
Chapter 26

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 (int width)
Use scrollbar_size() instead.

Member Fl_Browser___::scrollbar_width () const
Use scrollbar_size() instead.

Member Fl_clip (int x, int y, int w, int h)
Please use fl_push_clip(int x, int y, int w, int h) instead. fl_clip(int, int, int, int) will be removed in FLTK 1.5.

Member Fl_File_Input::errorcolor () const
Will be removed in FLTK 1.5.0 or higher.

Member Fl_File_Input::errorcolor (Fl_Color c)
Will be removed in FLTK 1.5.0 or higher.

Member fl_find (Window xid)
Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wl_window*), fl_win32_find(HWND) or fl_mac_find(FLWindow*) with FLTK 1.4.0 and above.

Member Fl_Flex::isSetSize (Fl_Widget ∗w) const
Please use set_size(Fl_Widget *) instead.

Member Fl_Flex::setSize (Fl_Widget ∗w, int size)
Please use set_size(Fl_Widget *, int) instead.

Member Fl_GIF_Image::Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data)
Please use Fl_GIF_Image(const char ∗imagename, const unsigned char ∗data, const size_t length) instead.

Member Fl_Group::focus (Fl_Widget ∗W)
This is for backwards compatibility only.

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::label (Fl_Menu_Item *m)`
Please use `Fl_Menu_Item::image()` instead.

Member `Fl_Image::label (Fl_Widget *w)`
Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Member `Fl_Image_Surface::highres_image()`
Use `image()` instead.

Member `fl_input (const char *fmt, const char *defstr,...)`
Please use `fl_input_str(int maxchar, const char *fmt, const char *defstr,...)` instead.

Member `Fl_Menu_Item::check()`
Please use `Fl_Menu_Item::set()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `Fl_Menu_Item::checked () const`
Please use `Fl_Menu_Item::value()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `Fl_Menu_Item::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()`
Please use `Fl_Menu_Item::clear()` instead. This method will be removed in FLTK 1.5.0 or later.

Member `fl_password (const char *fmt, const char *defstr,...)`
Please use `fl_password_str(int maxchar, const char *fmt, const char *defstr,...)` instead.

Member `Fl_Text_Display::scrollbar_width (int width)`
Use `scrollbar_size()` instead.

Member `Fl_Text_Display::scrollbar_width () const`
Use `scrollbar_size()` instead.

Member `Fl_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::next_displayed (Fl_Tree_Prefs &prefs)`
in 1.3.3 for confusing name, use `next_visible()` instead

Member `Fl_Tree::prev_displayed (Fl_Tree_Prefs &prefs)`
in 1.3.3 for confusing name, use `prev_visible()`

Member `FL_VERSION`
This double version number is retained for compatibility with existing program code. New code should use `int FL_API_VERSION` instead. `FL_VERSION` is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

Member `Fl_Widget::color2 (unsigned a)`
Use `selection_color(unsigned)` instead.
Member `Fl_Widget::color2` () const
Use `selection_color()` instead.

Member `Fl_Window::free_position` ()
please use `force_position(0)` instead

Member `Fl_Window::icon` () const
in 1.3.3

Member `Fl_Window::icon` (const void *ic)
in 1.3.3 in favor of platform-independent methods `Fl_Window::icon(const Fl_RGB_Image *icon)` and `Fl_Window::icons(const Fl_RGB_Image *icons[], int count)`.

Page Operating System Issues
Kept for compatibility with FLTK versions before 1.4. Use preferentially `fl_x11_xid(const Fl_Window *)` with versions 1.4 and above.
Kept for compatibility with FLTK versions before 1.4. Use preferentially `fl_x11_find(Window)` with versions 1.4 and above.
Chapter 27

Module Index

27.1 Modules

Here is a list of all modules:

- Callback Function Typedefs .................................................. 239
- Windows handling functions ............................................... 240
- Events handling functions .................................................. 242
- Selection & Clipboard functions ......................................... 256
- Screen functions ............................................................... 259
- Color & Font functions ....................................................... 264
- Drawing functions ............................................................. 274
- Multithreading support functions ........................................ 303
- Safe widget deletion support functions ................................. 305
- Cairo Support Functions and Classes ................................... 307
- Unicode and UTF-8 functions ............................................. 309
- Mac OS X-specific symbols .................................................. 325
- Common Dialog Classes and Functions ................................. 326
- File names and URI utility functions ..................................... 343
- Fl_string ............................................................................ 350
Chapter 28

Hierarchical Index

28.1 Class Hierarchy

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

`Fl_Preferences::Entry` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 351
`Fl` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 351
`Fl_Cairo_State` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 426
`FL_CHART_ENTRY` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 434
`Fl_End` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 468
`Fl_File_Chooser` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 474
`Fl_File_Icon` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 481
`Fl_Gl_Choice` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 509
`Fl_Glut_Bitmap_Font` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518
`Fl_Glut_StrokeChar` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518
`Fl_Glut_StrokeFont` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518
`Fl_Glut_StrokeStrip` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518
`Fl_Glut_StrokeVertex` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 519
`Fl_Help_Block` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
`Fl_Help_Dialog` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 533
`Fl_Help_Font_Stack` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 535
`Fl_Help_Font_Style` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 535
`Fl_Help_Link` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 536
`Fl_Help_Target` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 536
`Fl_Image` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 547
`Fl_Bitmap` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 379
`Fl_Bitmap::XBM` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1042
`Fl_Bitmap::Pixmap` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 651
`Fl_Bitmap::GIF` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 506
`Fl_Bitmap::XPM` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1043
`Fl_Bitmap::RGB` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 705
`Fl_Bitmap::BMP` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 381
`Fl_Bitmap::JPEG` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 589
`Fl_Bitmap::PNG` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 657
`Fl_Bitmap::PNM` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 659
`Fl_Bitmap::SVG` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 760
`Fl_Bitmap::Shared` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 726
`Fl_Bitmap::Tiled` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 872
`Fl_Bitmap::Reader` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 557
`Fl_Bitmap::Label` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 591
`Fl_Bitmap::Mac_App_Menu` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 595
`Fl_Bitmap::Menu_Item` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 616
`Fl_Bitmap::Multi_Label` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 630
`Fl_Bitmap::Native_File_Chooser` . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 633
`Fl_Bitmap::Native_File_Chooser_FLTK_Driver`
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Widget</td>
<td>977</td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1015</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_Copy_Surface</td>
<td>456</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_EPS_File_Surface</td>
<td>469</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_Image_Surface</td>
<td>558</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_Paged_Device</td>
<td>646</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_PostScript_File_Device</td>
<td>662</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_Printer</td>
<td>689</td>
</tr>
<tr>
<td>Fl_Widget_Surface:Fl_SVG_File_Surface</td>
<td>758</td>
</tr>
<tr>
<td>Fl_Widget_Device</td>
<td>977</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Box</td>
<td>383</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button</td>
<td>422</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Check_Button</td>
<td>439</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Radio_Light_Button</td>
<td>698</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Round_Button</td>
<td>711</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Round_Round_Button</td>
<td>698</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Radio_Button</td>
<td>697</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Repeat_Button</td>
<td>701</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Return_Button</td>
<td>703</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Button:Fl_Toggle_Button</td>
<td>883</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Clock</td>
<td>447</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Clock:Fl_Clock_Output</td>
<td>444</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Clock:Fl_Round_Clock</td>
<td>713</td>
</tr>
<tr>
<td>Fl_Widget:Fl_FormsBitmap</td>
<td>501</td>
</tr>
<tr>
<td>Fl_Widget:Fl_FormsPixmap</td>
<td>502</td>
</tr>
<tr>
<td>Fl_Widget:Fl_FormsText</td>
<td>504</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Free</td>
<td>504</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Group</td>
<td>521</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser</td>
<td>406</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_File_Browser</td>
<td>471</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Hold_Browser</td>
<td>471</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Hold_Browser</td>
<td>544</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Multi_Browser</td>
<td>629</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Select_Browser</td>
<td>725</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Check_Browser</td>
<td>434</td>
</tr>
<tr>
<td>Fl_Widget:Fl_Browser:Fl_Color_chooser</td>
<td>451</td>
</tr>
</tbody>
</table>
28.1 Class Hierarchy

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Flex</td>
<td>490</td>
</tr>
<tr>
<td>Fl_Help_View</td>
<td>537</td>
</tr>
<tr>
<td>Fl_Input_Choice</td>
<td>584</td>
</tr>
<tr>
<td>Fl_Pack</td>
<td>644</td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>714</td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>748</td>
</tr>
<tr>
<td>Fl_Table</td>
<td>772</td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>791</td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>793</td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>817</td>
</tr>
<tr>
<td>Fl_Simple_Terminal</td>
<td>735</td>
</tr>
<tr>
<td>Fl_Text_Editor</td>
<td>857</td>
</tr>
<tr>
<td>Fl_Tile</td>
<td>869</td>
</tr>
<tr>
<td>Fl_Tree</td>
<td>889</td>
</tr>
<tr>
<td>Fl_Window</td>
<td>1020</td>
</tr>
<tr>
<td>Fl_Double_Window</td>
<td>466</td>
</tr>
<tr>
<td>Fl_Cairo_Window</td>
<td>427</td>
</tr>
<tr>
<td>Fl_Overlay_Window</td>
<td>641</td>
</tr>
<tr>
<td>Fl_GL_Window</td>
<td>509</td>
</tr>
<tr>
<td>Fl_Glut_Window</td>
<td>519</td>
</tr>
<tr>
<td>Fl_Single_Window</td>
<td>744</td>
</tr>
<tr>
<td>Fl_Menu_Window</td>
<td>628</td>
</tr>
<tr>
<td>Fl_Wizard</td>
<td>1041</td>
</tr>
<tr>
<td>Fl_Input</td>
<td>565</td>
</tr>
<tr>
<td>Fl_File_Input</td>
<td>486</td>
</tr>
<tr>
<td>Fl_Float_Input</td>
<td>500</td>
</tr>
<tr>
<td>Fl_Int_Input</td>
<td>588</td>
</tr>
<tr>
<td>Fl_Multiline_Input</td>
<td>631</td>
</tr>
<tr>
<td>Fl_Output</td>
<td>640</td>
</tr>
<tr>
<td>Fl_Multiline_Output</td>
<td>632</td>
</tr>
<tr>
<td>Fl_Secret_Input</td>
<td>724</td>
</tr>
<tr>
<td>Fl_Spinner::Fl_Spinner_Input</td>
<td>753</td>
</tr>
<tr>
<td>Fl_Menu</td>
<td>596</td>
</tr>
<tr>
<td>Fl_Choice</td>
<td>440</td>
</tr>
<tr>
<td>Fl_Menu_Bar</td>
<td>611</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>765</td>
</tr>
<tr>
<td>Fl_Menu_Button</td>
<td>613</td>
</tr>
<tr>
<td>Fl_Positioner</td>
<td>660</td>
</tr>
<tr>
<td>Fl_Progress</td>
<td>695</td>
</tr>
<tr>
<td>Fl_Timer</td>
<td>880</td>
</tr>
<tr>
<td>Fl_Valuator</td>
<td>962</td>
</tr>
<tr>
<td>Fl_Adjuster</td>
<td>377</td>
</tr>
<tr>
<td>Fl_Counter</td>
<td>459</td>
</tr>
<tr>
<td>Fl_Simple_Counter</td>
<td>734</td>
</tr>
<tr>
<td>Fl_Dial</td>
<td>463</td>
</tr>
<tr>
<td>Fl_Fill_Dial</td>
<td>489</td>
</tr>
<tr>
<td>Fl_Line_Dial</td>
<td>595</td>
</tr>
<tr>
<td>Fl_Roller</td>
<td>710</td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>745</td>
</tr>
<tr>
<td>Fl_Fill_Slider</td>
<td>489</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider</td>
<td>545</td>
</tr>
<tr>
<td>Fl_HOR_Nice_Slider</td>
<td>546</td>
</tr>
<tr>
<td>Fl_Hor_Slider</td>
<td>546</td>
</tr>
<tr>
<td>Fl_Nice_Slider</td>
<td>639</td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>720</td>
</tr>
<tr>
<td>Fl_Value_Slider</td>
<td>974</td>
</tr>
</tbody>
</table>
Hierarchical Index

Fl_Hor_Value_Slider .......................................................... 547
Fl_Value_Input ................................................................. 967
Fl_Value_Output ............................................................... 971
Fl_Widget_Tracker .............................................................. 1018
Fl_XColor ....................................................................... 1043
Fl_Text_Editor::Key_Binding .................................................. 1044
Fl_Preferences::Name .......................................................... 1044
Fl_Preferences::Node .......................................................... 1045
Fl_Paged_Device::page_format ............................................... 1046
Fl_Preferences::RootNode ..................................................... 1046
Fl_Scroll::ScrollInfo .......................................................... 1046
Fl_Text_Display::Style_Table_Entry ........................................ 1047
Chapter 29

Class Index

29.1 Class List

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

- **Fl_Preferences::Entry** ................................................................. 351
- **Fl**
  The Fl is the FLTK global (static) class containing state information and global methods for the current application .................................................. 351
- **Fl_Adjuster**
  Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range .......................................................... 377
- **Fl_Bitmap**
  Supports caching and drawing of mono-color (bitmap) images ........................... 379
- **Fl_BMP_Image**
  Supports loading, caching, and drawing of Windows Bitmap (BMP) image files ... 381
- **Fl_Box**
  This widget simply draws its box, and possibly its label ................................ 383
- **Fl_Browser**
  Displays a scrolling list of text lines, and manages all the storage for the text .... 385
- **Fl_Browser_**
  This is the base class for browsers .................................................... 406
- **Fl_Button**
  Buttons generate callbacks when they are clicked by the user ....................... 422
- **Fl_Cairo_State**
  Contains all the necessary info on the current cairo context ....................... 426
- **Fl_Cairo_Window**
  This defines a FLTK window with cairo support .................................... 427
- **Fl_Chart**
  Fl_Chart displays simple charts ....................................................... 429
- **Fl_CHART_ENTRY**
  For internal use only ........................................................................... 434
- **Fl_Check_Browser**
  Displays a scrolling list of text lines that may be selected and/or checked by the user .................. 434
- **Fl_Check_Button**
  A button with a "checkmark" to show its status ...................................... 439
- **Fl_Choice**
  A button that is used to pop up a menu .................................................. 440
- **Fl_Clock**
  This widget provides a round analog clock display ..................................... 444
- **Fl_Clock_Output**
  This widget can be used to display a program-supplied time ....................... 447
- **Fl_Color_Chooser**
  Standard RGB color chooser .................................................................. 451
Fl_Copy_Surface
Supports copying of graphical data to the clipboard ........................................ 456
Fl_Counter
Controls a single floating point value with button (or keyboard) arrows .................. 459
Fl_Device_Plugin
This plugin socket allows the integration of new device drivers for special window or screen types .... 462
Fl.Dial
Circular dial to control a single floating point value ........................................... 463
Fl_Display_Device
A display to which the computer can draw ............................................................ 465
Fl_Double_Window
The Fl_Double_Window provides a double-buffered window .................................... 466
Fl_End
This is a dummy class that allows you to end a Fl_Group in a constructor list of a class: ........ 468
Fl_EPS_File_Surface
Encapsulated PostScript drawing surface ............................................................... 469
Fl_File_Browser
Displays a list of filenames, optionally with file-specific icons ................................ 471
Fl_File_Chooser
Displays a standard file selection dialog that supports various selection modes ............ 474
Fl_File_Icon
Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget .............................................................. 481
Fl_File_Input
This widget displays a pathname in a text input field ............................................. 486
Fl_Fill_Dial
Draws a dial with a filled arc .................................................................................. 489
Fl_Fill_Slider
Widget that draws a filled horizontal slider, useful as a progress or value meter ............ 489
Fl_Flex
Fl_Flex is a container (layout) widget for one row or one column of widgets ............ 490
Fl_Float_Input
Subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits) .......................................................... 500
Fl_FormsBitmap
Forms compatibility Bitmap Image Widget ............................................................. 501
Fl_FormsPixmap
Forms pixmap drawing routines .............................................................................. 502
Fl_FormsText
 ............................................................................................................................. 504
Fl_Free
Emulation of the Forms "free" widget ...................................................................... 504
Fl_GIF_Image
Supports loading, caching, and drawing of Compuserve GIF\textsuperscript{SM} images ........... 506
Fl_Gl_Choice
 ............................................................................................................................. 509
Fl_Gl_Window
Sets things up so OpenGL works ............................................................................ 509
Fl_Glut_Bitmap_Font
Fltk glut font/size attributes used in the glutXXX functions ................................... 518
Fl_Glut_Choice
 ............................................................................................................................. 518
Fl_Glut_Font
 ............................................................................................................................. 518
Fl_Glut_Paint Strip
 ............................................................................................................................. 518
Fl_Glut_Paint Vertex
 ............................................................................................................................. 519
Fl_Glut_Window
GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx): ......................................................... 519
Fl_Group
FLTK container widget ............................................................................................. 521
Fl_Help_Block
 ............................................................................................................................. 532
29.1 Class List

- **Fl_Help_Dialog**
  Displays a standard help dialog window using the `Fl_Help_View` widget
  
- **Fl_Help_Font_Stack**
  
- **Fl_Help_Font_Style**
  `Fl_Help_View` font stack element definition
  
- **Fl_Help_Link**
  Definition of a link for the html viewer
  
- **Fl_Help_Target**
  `Fl_Help_Target` structure
  
- **Fl_Help_Abstract**
  Displays HTML text
  
- **Fl_Hold_Browser**
  The `Fl_Hold_Browser` is a subclass of `Fl_Browser` which lets the user select a single item, or no items by clicking on the empty space
  
- **Fl_Hor_Fill_Slider**
  
- **Fl_Hor_Nice_Slider**
  
- **Fl_Hor_Slider**
  Horizontal Slider class
  
- **Fl_Hor_Value_Slider**
  
- **Fl_Image**
  Base class for image caching, scaling and drawing
  
- **Fl_Image_Reader**
  
- **Fl_Image_Surface**
  Directs all graphics requests to an `Fl_Image`
  
- **Fl_Input**
  This is the FLTK text input widget
  
- **Fl_Input_Abstract**
  This class provides a low-overhead text input field
  
- **Fl_Input_Choice**
  A combination of the input widget and a menu button
  
- **Fl_Int_Input**
  Subclass of `Fl_Input` that only allows the user to type decimal digits (or hex numbers of the form 0xaef)
  
- **Fl_JPEG_Image**
  Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images
  
- **Fl_Kdialog_Native_File_Chooser_Driver**
  
- **Fl_Label**
  This struct stores all information for a text or mixed graphics label
  
- **Fl_Light_Button**
  This subclass displays the "on" state by turning on a light, rather than drawing pushed in
  
- **Fl_Line_Dial**
  
- **Fl_Mac_App_Menu**
  
- **Fl_Menu**
  Base class of all widgets that have a menu in FLTK
  
- **Fl_Menu_Bar**
  This widget provides a standard menubar interface
  
- **Fl_Menu_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
  
- **Fl_Menu_Item**
  The `Fl_Menu_Item` structure defines a single menu item that is used by the `Fl_Menu_Item` class
  
- **Fl_Menu_Window**
  Window type used for menus
  
- **Fl_Multi_Browser**
  Subclass of `Fl_Browser` which lets the user select any set of the lines
  
- **Fl_Multi_Label**
  Allows a mixed text and/or graphics label to be applied to an `Fl_Menu_Item` or `Fl_Widget`
### Class Index

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fl_Multiline_Input</strong></td>
<td>This input field displays ‘\n’ characters as new lines rather than ‘^J’, and accepts the Return, Tab, and up and down arrow keys</td>
<td>631</td>
</tr>
<tr>
<td><strong>Fl_Multiline_Output</strong></td>
<td>This widget is a subclass of Fl_Output that displays multiple lines of text</td>
<td>632</td>
</tr>
<tr>
<td><strong>Fl_Native_File_Chooser</strong></td>
<td>This class lets an FLTK application easily and consistently access the operating system’s native file chooser</td>
<td>633</td>
</tr>
<tr>
<td><strong>Fl_Nice_Slider</strong></td>
<td></td>
<td>639</td>
</tr>
<tr>
<td><strong>Fl_Output</strong></td>
<td>This widget displays a piece of text</td>
<td>640</td>
</tr>
<tr>
<td><strong>Fl_Overlay_Window</strong></td>
<td>This window provides double buffering and also the ability to draw the “overlay” which is another picture placed on top of the main image</td>
<td>641</td>
</tr>
<tr>
<td><strong>Fl_Pack</strong></td>
<td>This widget was designed to add the functionality of compressing and aligning widgets</td>
<td>644</td>
</tr>
<tr>
<td><strong>Fl_Paged_Device</strong></td>
<td>Represents page-structured drawing surfaces</td>
<td>646</td>
</tr>
<tr>
<td><strong>Fl_Pixmap</strong></td>
<td>Supports caching and drawing of colormap (pixmap) images, including transparency</td>
<td>651</td>
</tr>
<tr>
<td><strong>Fl_Plugin</strong></td>
<td>This class is provided for Forms compatibility</td>
<td>660</td>
</tr>
<tr>
<td><strong>Fl_Plugin_Manager</strong></td>
<td>To send graphical output to a PostScript file</td>
<td>662</td>
</tr>
<tr>
<td><strong>Fl_PNG_Image</strong></td>
<td>Supports loading, caching, and drawing of Portable Network Graphics (PNG) image files</td>
<td>657</td>
</tr>
<tr>
<td><strong>Fl_PNM_Image</strong></td>
<td>Supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files</td>
<td>659</td>
</tr>
<tr>
<td><strong>Fl_Positioner</strong></td>
<td>This class is provided for Forms compatibility</td>
<td>669</td>
</tr>
<tr>
<td><strong>Fl_PostScript_File_Device</strong></td>
<td>OS-independent print support</td>
<td>689</td>
</tr>
<tr>
<td><strong>Fl_Progress</strong></td>
<td>Displays a progress bar for the user</td>
<td>695</td>
</tr>
<tr>
<td><strong>Fl_Radio_Button</strong></td>
<td>The Fl_Radio_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down</td>
<td>697</td>
</tr>
<tr>
<td><strong>Fl_Radio_Light_Button</strong></td>
<td>The Fl_Radio_Light_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key</td>
<td>698</td>
</tr>
<tr>
<td><strong>Fl_Radio_Round_Button</strong></td>
<td>Supports caching and drawing of full-color images with 1 to 4 channels of color information</td>
<td>705</td>
</tr>
<tr>
<td><strong>Fl_Rect</strong></td>
<td>“dolly” control commonly used to move 3D objects</td>
<td>710</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Fl_Round_Button</td>
<td>Buttons generate callbacks when they are clicked by the user.</td>
<td></td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>A clock widget of type FL_ROUND_CLOCK</td>
<td></td>
</tr>
<tr>
<td>Fl_Scroll</td>
<td>This container widget lets you maneuver around a set of widgets much larger than your window.</td>
<td></td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>Displays a slider with arrow buttons at the ends of the scrollbar.</td>
<td></td>
</tr>
<tr>
<td>Fl_Scrollbar_Data</td>
<td>A local struct to manage a scrollbar’s xywh region and tab values.</td>
<td></td>
</tr>
<tr>
<td>Fl_Secret_Input</td>
<td>Subclass of Fl_Input that displays its input as a string of placeholders.</td>
<td></td>
</tr>
<tr>
<td>Fl_Select_Browser</td>
<td>The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.</td>
<td></td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>This class supports caching, loading, and drawing of image files.</td>
<td></td>
</tr>
<tr>
<td>Fl_Simple_Counter</td>
<td>This widget creates a counter with only 2 arrow buttons.</td>
<td></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>
<td></td>
</tr>
<tr>
<td>Fl_Single_Window</td>
<td>This is the same as Fl_Window.</td>
<td></td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>Sliding knob inside a box.</td>
<td></td>
</tr>
<tr>
<td>Fl_Spinner</td>
<td>This widget is a combination of a numerical input widget and repeat buttons.</td>
<td></td>
</tr>
<tr>
<td>Fl_Spinner_Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fl_String</td>
<td>Fl_String is the basic string class for FLTK.</td>
<td></td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>A drawing surface that’s susceptible to receive graphical output.</td>
<td></td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>A drawing surface producing a Scalable Vector Graphics (SVG) file.</td>
<td></td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>Supports loading, caching and drawing of scalable vector graphics (SVG) images.</td>
<td></td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar</td>
<td>A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.</td>
<td></td>
</tr>
<tr>
<td>Fl_Table</td>
<td>A table of widgets or other content.</td>
<td></td>
</tr>
<tr>
<td>Fl_Table_Row</td>
<td>A table with row selection capabilities.</td>
<td></td>
</tr>
<tr>
<td>Fl_Tabs</td>
<td>“file card tabs” interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits.</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>This class manages Unicode text displayed in one or more Fl_Text_Display widgets.</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Display</td>
<td>Rich text display widget.</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Editor</td>
<td>This is the FLTK text editor widget.</td>
<td></td>
</tr>
<tr>
<td>Fl_Text_Selection</td>
<td>This is an internal class for Fl_Text_Buffer to manage text selections.</td>
<td></td>
</tr>
<tr>
<td>Fl_Tile</td>
<td>Lets you resize its children by dragging the border between them.</td>
<td></td>
</tr>
<tr>
<td>Fl_Tiled_Image</td>
<td>This class supports tiling of images over a specified area.</td>
<td></td>
</tr>
</tbody>
</table>
Class Index

FL_Timeout
Class FL_Timeout handles all timeout related functions ........................................... 875

FL_Timer
This is provided only to emulate the Forms Timer widget ........................................... 880

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 .................................................................................................................. 883

FL_Tooltip
Tooltip support for all FLTK widgets .............................................................................. 884

FL_Tree
Tree widget ..................................................................................................................... 889

FL_Tree_Item
Tree widget item ............................................................................................................. 931

FL_Tree_Item_Array
Manages an array of FL_Tree_Item pointers .................................................................... 953

FL_Tree_Prefs
Tree widget's preferences ............................................................................................... 956

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 .......................... 962

FL_Value_Inp ut
Displays a numeric value ................................................................................................. 967

FL_Value_Output
Displays a floating point value ....................................................................................... 971

FL_Value_Slider
FL_Slider widget with a box displaying the current value ................................................ 974

FL_Widget
FL_Widget is the base class for all widgets in FLTK ................................................................ 977

FL_Widget_Surface
A surface on which any FLTK widget can be drawn ......................................................... 1015

FL_Widget_Tracker
This class should be used to control safe widget deletion .................................................. 1018

FL_Window
This widget produces an actual window ........................................................................... 1020

FL_Wizard
This widget is based off the FL_Tabs widget, but instead of displaying tabs it only changes "tabs"
under program control ..................................................................................................... 1041

FL_XBM_Image
Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files .......................... 1042

FL_XColor .................................................................................................................. 1043

FL_XPM_Image
Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency .. 1043

FL_Text_Editor::Key_Binding
Simple linked list item associating a key/state to a function ............................................ 1044

FL_Preferences::Name
'Name' provides a simple method to create numerical or more complex procedural names for
entries and groups on the fly ......................................................................................... 1044

FL_Preferences::Node ................................................................................................... 1045

FL_Paged_Device::page_format
Width, height and name of a page format ........................................................................ 1046

FL_Preferences::RootNode ............................................................................................. 1046

FL_Scroll::ScrollInfo
Structure to manage scrollbar and widget interior sizes .................................................. 1046

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 .................................................................................................................. 1047

Generated by Doxygen
Chapter 30

File Index

30.1 File List

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

- armscii_8.h
- ascii.h
- big5.h
- big5_emacs.h
- case.h
- cgdebug.h
- cp1133.h
- cp1251.h
- cp1255.h
- cp1256.h
- cp936ext.h
- dingbats_.h

Enumerations.H

This file contains type definitions and general enumerations

fastarrow.h

filename.H

File names and URI utility functions

Fl.cxx

Implementation of the member functions of class Fl

Fl.H

Fl static class

Fl_Adjuster.H

Utility functions for drawing arcs and circles

fl_arc.cxx

Utility functions for common dialogs

fl_ask.cxx

fl_ask.H

API for common dialogs

fl_attr.h

Fl_Bitmap.H

Fl_BMPэтаж.Image.H

Fl_Box.H

Fl_Browser.H

Fl_Browser_.H

Fl_Button.H

Fl_Cairo.H

Fl_Cairo_Window.H

fl_boxtype.cxx

Drawing code for common box types

fl_casts.H

Generated by Doxygen
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 ............................................. 1081
Fl_Color_Chooser.H
  Fl_Color_Chooser widget ................................... 1082
fl-compose.cxx
  Utility functions to support text input .................. 1082
fl_config.h ................................................ ??
Fl_Copy_Surface.H ........................................ ??
Fl_Counter.H ............................................... ??
fl_curve.cxx
  Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path 1083
Fl_Device.H
  Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin .............. 1083
fl-double_window.cxx
  Fl_Double_Window implementation .......................... 1083
Fl_Double_Window.H .......................................... ??
fl-draw.h
  Utility header to pull drawing functions together .... 1084
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_Flex.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_Image.H
  Fl_Image, Fl_RGB_Image classes ............................. 1089
Fl_Image_Reader.h .......................................... ??
Fl_Image_Surface.H ......................................... ??
Fl_Input.H ................................................... ??
Fl_Input_.H ................................................... ??
Fl_Input_Choice.H .......................................... ??
Fl_Int_Input.H .............................................. ??
Fl_JPEG_Image.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 ..................................................................... 1090
Fl_Menu_Window.H ................................................................. ??
Fi_Message.h ....................................................................... ??
fi_message.H ........................................................................ ??
Fl_Multi_Browser.H ............................................................... ??
Fl_Multi_Label.H ................................................................... ??
Fl_Multiline_Input.H .............................................................. ??
Fl_Multiline_Output.H ............................................................. ??
Fl_Native_File_Selector.H ...................................................... ??
Fl_Native_File_Selector_Kdialog.H .......................................... ??
Fl_Nice_Slider.H ................................................................... ??
Fl_Object.H ........................................................................... ??
Fl_Output.H ............................................................................ ??
Fl_Overlay_Window.H ............................................................ ??
Fl_Pack.H .............................................................................. ??
Fl_Paged_Device.cxx .............................................................. 1091
Fl_Paged_Device.H ................................................................. 1091
Fl_Pixmap.H ......................................................................... ??
Fl_Plugin.H .......................................................................... ??
Fl_PNG_Image.H ................................................................... ??
Fl_PNM_Image.H .................................................................. ??
Fl_Positioner.H ................................................................... ??
Fl_PostScript.H ..................................................................... ??
Fl_PostScript_File_Device and Fl_EPS_File_Surface .................. 1092
Fl_Preferences.H ................................................................... ??
Fl_Printer.H ........................................................................... ??
Fl_Progress.H ...................................................................... ??
Fl_Radio_Button.H ................................................................. ??
Fl_Radio_Light_Button.H ........................................................ ??
Fl_Radio_Round_Button.H ...................................................... ??
fll_rect.cxx ......................................................................... 1093
Fl_Rect.H ............................................................................... ??
Fl_Repeat_Button.H ............................................................... ??
Fl_Return_Button.H ............................................................... ??
Fl_RGB_Image.H ................................................................... ??
Fl_Roller.H ............................................................................. ??
Fl_Round_Button.H ............................................................... ??
Fl_Round_Clock.H ................................................................. ??
Fl_Screen_Driver.H ............................................................... ??
Fl_Scroll.H .......................................................................... ??
Fl_Scrollbar.H ..................................................................... ??
Fl_Secret_Input.H .................................................................. ??
Fl_Select_Browser.H ............................................................. ??
Fl_Shared_Image.H .................................................................. 1093

Fl_Shared_Image class ................................................................ 1093

Fl_Shared_Image.H
The \texttt{fl\_show\_colormap()} function hides the implementation classes used to provide the popup window and color selection mechanism.

\texttt{Fl\_Simple\_Counter.H}

\texttt{Fl\_Simple\_Terminal.H}

\texttt{Fl\_Single\_Window.H}

\texttt{Fl\_Slider.H}

\texttt{Fl\_Spinner.H}

Basic \texttt{Fl\_String} class for FLTK.

\texttt{Fl\_Sys\_Menu\_Bar.H}

Definition of class \texttt{Fl\_Sys\_Menu\_Bar}.

\texttt{Fl\_Sys\_Menu\_Bar\_Driver.H}

\texttt{Fl\_System\_Driver.H}

\texttt{Fl\_Table.H}

\texttt{Fl\_Table\_Row.H}

\texttt{Fl\_Tabs.H}

\texttt{Fl\_Text\_Buffer.H}

\texttt{Fl\_Text\_Display.H}

\texttt{Fl\_Text\_Editor.H}

\texttt{Fl\_Tile.H}

\texttt{Fl\_Tiled\_Image.H}

\texttt{Fl\_Timer.H}

\texttt{Fl\_Timeout.H}

\texttt{Fl\_Timeout\_cxx}

\texttt{Fl\_Tooltip.H}

This file contains the definitions of the \texttt{Fl\_Tree} class.

\texttt{Fl\_Tree\_Item.H}

This file contains the definitions for \texttt{Fl\_Tree\_Item}.

\texttt{Fl\_Tree\_Item\_Array.H}

This file defines a class that manages an array of \texttt{Fl\_Tree\_Item} pointers.

\texttt{Fl\_Tree\_Prefs.H}

This file contains the definitions for \texttt{Fl\_Tree}'s preferences.

\texttt{fl\_types.h}

This file contains simple "C"-style type definitions.

\texttt{fl\_utf8.h}

Header for Unicode and UTF-8 character handling.

\texttt{Fl\_Valuator.H}

\texttt{Fl\_Value\_Input.H}

\texttt{Fl\_Value\_Output.H}

\texttt{Fl\_Value\_Slider.H}

\texttt{fl\_vertex.cxx}

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

\texttt{Fl\_Widget.H}

\texttt{Fl\_Widget, Fl\_Label} classes.

\texttt{Fl\_Widget\_Surface.H}
Definitions of platform-dependent types

print_button.h
print_panel.h
slowarrow.h
spacing.h
symbol_.h ................................................................. ??
tatar_cyr.h ............................................................... ??
tcvn.h ................................................................. ??
tis620.h ................................................................. ??
ucs2be.h ................................................................. ??
tuf8.h ................................................................. ??
tuf8_internal.h ....................................................... ??
viscii.h ................................................................. ??
vsnprintf.c  Portable vsnprintf() implementation .................. 1114
wayland.H Definitions of functions specific to the Wayland platform .................. 1115
win32.H Definitions of functions specific to the Windows platform .................... 1116
x.H ................................................................. ??
x11.H Definitions of functions specific to the X11 platform ....................... 1116
Ximint.h ............................................................... ??
Xlibint.h ............................................................... ??
Xutf8.h ............................................................... ??
31.1 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 `awake` 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 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_Timeout_Handler)(void *data)`
  
  Signature of some timeout callback functions passed as parameters.
31.1.1 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

31.1.2 Typedef Documentation

31.1.2.1 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)

31.2 Windows handling functions

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

Functions

- static void Fl::default_atclose (Fl_Window *, void *)
  Default callback for window widgets.
- static Fl_Window * Fl::first_window ()
  Returns the first top-level window in the list of shown() windows.
- static void Fl::first_window (Fl_Window *)
  Sets the window that is returned by first_window().
- static Fl_Window * Fl::grab ()
  Returns the window that currently receives all events.
- static void Fl::grab (Fl_Window *)
  Selects the window to grab.
- static Fl_Window * Fl::modal ()
  Returns the top-most modal() window currently shown.
- static Fl_Window * Fl::next_window (const Fl_Window *)
  Returns the next top-level window in the list of shown() windows.
- static void Fl::set_abort (Fl_Abort_Handler f)
  For back compatibility, sets the void Fl::fatal handler callback.
- static void Fl::set_atclose (Fl_Atclose_Handler f)
  For back compatibility, sets the Fl::atclose handler callback.

Variables

- static void(* Fl::atclose ) (Fl_Window *, void *)
  Back compatibility: default window callback handler.

31.2.1 Detailed Description

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

31.2.2 Function Documentation

31.2.2.1 default_atclose()

```c
void Fl::default_atclose ( 
    Fl_Window * window,
    void * v ) [static]
```

Default callback for window widgets.
It hides the window and then calls the default widget callback.

31.2.2.2 first_window() [1/2]

```c
Fl_Window * Fl::first_window ( ) [static]
```

Returns the first top-level window in the list of shown() windows.
If a modal() window is shown this is the top-most modal window, otherwise it is the most recent window to get an event.

31.2.2.3 first_window() [2/2]

```c
void Fl::first_window ( 
    Fl_Window * window ) [static]
```

Sets the window that is returned by first_window().
The window is removed from wherever it is in the list and inserted at the top. This is not done if Fl::modal() is on or if the window is not shown(). Because the first window is used to set the “parent” of modal windows, this is often useful.

31.2.2.4 grab() [1/2]

```c
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.

31.2.2.5 grab() [2/2]

```c
void Fl::grab ( 
    Fl_Window * win ) [static]
```

Selects the window to grab.
This is used when pop-up menu systems are active.
Send all events to the passed window no matter where the pointer or focus is (including in other programs). The window does not have to be shown(), this lets the handle() method of a "dummy" window override all event handling and allows you to map and unmap a complex set of windows (under both X and Windows some window must be mapped because the system interface needs a window id).
If grab() is on it will also affect show() of windows by doing system-specific operations (on X it turns on override-redirect). These are designed to make menus popup reliably and faster on the system.
To turn off grabbing do Fl::grab(0).
Be careful that your program does not enter an infinite loop while grab() is on. On X this will lock up your screen!
To avoid this potential lockup, all newer operating systems seem to limit mouse pointer grabbing to the time during which a mouse button is held down. Some OS's may not support grabbing at all.

31.2.2.6 modal()

```c
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).

31.2.2.7 next_window()

Fl_Window * Fl::next_window (const Fl_Window * window) [static]

Returns the next top-level window in the list of shown() windows.
You can use this call to iterate through all the windows that are shown().

Parameters

in window must be shown and not NULL

31.2.2.8 set_atclose()

static void Fl::set_atclose (Fl_Atclose_Handler f) [inline], [static]

For back compatibility, sets the Fl::atclose handler callback.
You can now simply change the callback for the window instead.

See also

Fl_Window::callback(Fl_Callback*)

31.2.3 Variable Documentation

31.2.3.1 atclose

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

Back compatibility: default window callback handler.

See also

Fl::set_atclose()

31.3 Events handling functions

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

Functions

- static void Fl::add_handler (Fl_Event_Handler h)
  Install a function to parse unrecognized events.
- static void Fl::add_system_handler (Fl_System_Handler h, void *data)
  Install a function to intercept system events.
- static Fl_Widget * Fl::belowmouse ()
  Gets the widget that is below the mouse.
- static void Fl::belowmouse (Fl_Widget *)
  Sets the widget that is below the mouse.
- static int Fl::compose (int &del)
  Any text editing widget should call this for each FL_KEYBOARD event.
- 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 ()**
  Returns a pointer to the pasted data.
- **static const char * Fl::*event_clipboard_type ()**
  Returns the type of the pasted data.
- **static int Fl::event_command ()**
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- **static int Fl::event_ctrl ()**
  Returns non-zero if the Control key is pressed.
- **static Fl::Event_Dispatch Fl::event_dispatch ()**
  Returns the current event dispatch function.
- **static void Fl::event_dispatch (Fl::Event_Dispatch d)**
  Sets a new event dispatch function.
- **static int Fl::event_dx ()**
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- **static int Fl::event_dy ()**
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- **static int Fl::event_inside (const Fl_Widget *)**
  Returns whether or not the mouse event is inside a given child widget.
- **static int Fl::event_inside (int, int, int, int)**
  Returns whether or not the mouse event is inside the given rectangle.
- **static int Fl::event_is_click ()**
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- **static void Fl::event_is_click (int i)**
  Clears the value returned by Fl::event_is_click().
- **static int Fl::event_key ()**
  Gets which key on the keyboard was last pushed.
- **static int Fl::event_key (int key)**
Returns true if the given key was held down (or pressed) during the last event.

- static int Fl::event_length ()
  Returns the length of the text in Fl::event_text().

- static int Fl::event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.

- static int Fl::event_shift ()
  Returns non-zero if the Shift key is pressed.

- static int Fl::event_state ()
  Returns the keyboard and mouse button states of the last event.

- static int Fl::event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.

- static const char ∗ Fl::event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.

- static int Fl::event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

- static int Fl::event_x_root ()
  Returns the mouse position on the screen of the event.

- static int Fl::event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

- static int Fl::event_y_root ()
  Returns the mouse position on the screen of the event.

- static Fl_Widget ∗ Fl::focus ()
  Gets the current Fl::focus() widget.

- static void Fl::focus (Fl_Widget ∗)
  Sets the widget that will receive FL_KEYBOARD events.

- static int Fl::get_key (int key)
  Returns true if the given key is held down now.

- static void Fl::get_mouse (int &, int &)
  Return where the mouse is on the screen by doing a round-trip query to the server.

- static int Fl::handle (int, Fl_Window ∗)
  Handle events from the window system.

- static int Fl::handle_ (int, Fl_Window ∗)
  Handle events from the window system.

- static Fl_Widget ∗ Fl::pushed ()
  Gets the widget that is being pushed.

- static void Fl::pushed (Fl_Widget ∗)
  Sets the widget that is being pushed.

- static void Fl::remove_handler (Fl_Event_Handler h)
  Removes a previously added event handler.

- static void Fl::remove_system_handler (Fl_System_Handler h)
  Removes a previously added system event handler.

- static int Fl::test_shortcut (Fl_Shortcut)
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

Variables

- const char ∗const fl_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.
31.3 Events handling functions

31.3.1 Detailed Description

Fl class events handling API declared in `<FL/Fl.H>`

31.3.2 Function Documentation

31.3.2.1 add_handler()

```c
void Fl::add_handler ( Fl_Event_Handler ha ) [static]
```

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

- **FL_SHORTCUT** events that are not recognized by any widget. This lets you provide global shortcut keys.
- **FL_SCREEN_CONFIGURATION_CHANGED** events. Under X11, this event requires the libXrandr.so shared library to be loadable at run-time and the X server to implement the RandR extension.
- **FL_ZOOM_EVENT** events.
- System events that FLTK does not recognize. See fl_xevent.
- Some other events when the widget FLTK selected returns zero from its handle() method. Exactly which ones may change in future versions, however.

See also
- Fl::remove_handler(Fl_Event_Handler)
- Fl::event_dispatch(Fl_Event_Dispatch d)
- Fl::handle(int, Fl_Window *)

31.3.2.2 add_system_handler()

```c
void Fl::add_system_handler ( Fl_System_Handler ha, void * data ) [static]
```

Install a function to intercept system events. FLTK calls each of these functions as soon as a new system event is received. The processing will stop at the first function to return non-zero. If all functions return zero then the event is passed on for normal handling by FLTK. Each function will be called with a pointer to the system event as the first argument and `data` as the second argument. The system event pointer will always be void *, but will point to different objects depending on the platform:

- X11: XEvent
- Windows: MSG
- OS X: NSEvent

Parameters

<table>
<thead>
<tr>
<th>ha</th>
<th>The event handler function to register</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>User data to include on each call</td>
</tr>
</tbody>
</table>
See also

```cpp
Fl::remove_system_handler(Fl_System_Handler)
```

### 31.3.2.3 `belowmouse()` [1/2]

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

Gets the widget that is below the mouse.

See also

```cpp
belowmouse(Fl_Widget*)
```

### 31.3.2.4 `belowmouse()` [2/2]

```cpp
void Fl::belowmouse (Fl_Widget * o) [static]
```

Sets the widget that is below the mouse.

This is for highlighting buttons. It is not used to send FL_PUSH or FL_MOVE directly, for several obscure reasons, but those events typically go to this widget. This is also the first widget tried for FL_SHORTCUT events.

If you change the belowmouse widget, the previous one and all parents (that don't contain the new widget) are sent FL_LEAVE events. Changing this does not send FL_ENTER to this or any widget, because sending FL_ENTER is supposed to test if the widget wants the mouse (by it returning non-zero from handle()).

### 31.3.2.5 `compose()`

```cpp
int Fl::compose (int & del) [static]
```

Any text editing widget should call this for each FL_KEYBOARD event.

Use of this function is very simple.

If `true` is returned, then it has modified the `Fl::event_text()` and `Fl::event_length()` to a set of bytes to insert (it may be of zero length!). It will also set the "del" parameter to the number of bytes to the left of the cursor to delete, this is used to delete the results of the previous call to `Fl::compose()`.

If `false` is returned, the keys should be treated as function keys, and `del` is set to zero. You could insert the text anyways, if you don’t know what else to do.

Text editing widgets can preferentially call `fl_set_spot()` to indicate the window coordinates of the bottom of the current insertion point and the line height. This way, auxiliary windows that help choosing among alternative characters with some text input methods appear just below or above the insertion point. If widgets don’t do that, such auxiliary windows appear at the widget’s bottom.

On some platforms, text input can involve marked text, that is, temporary text replaced by other text during the input process. This occurs, e.g., under Wayland or macOS when using dead keys or when entering CJK characters.

Text editing widgets should preferentially signal marked text, usually underlining it. Widgets can use `int Fl::compose_state` after having called `Fl::compose(int&)` to obtain the length in bytes of marked text that always finishes at the current insertion point. Widgets should also call `void fl_reset_spot()` when processing FL_UNFOCUS events. The `Fl_Input` and `Fl_Text_Editor` widgets underline marked text. If none of this is done by a user-defined text editing widget, text input will work, but will not signal to the user what text is marked.

Finally, text editing widgets should call `set_flag(MAC_USE_ACCENTS_MENU);` in their constructor if they want to use, on the macOS platform, the feature introduced with Mac OS 10.7 “Lion” where pressing and holding certain keys on the keyboard opens a diacritic marks popup window.

**Note**

For compatibility with FLTK 1.3, text editing widgets can call `Fl::insertion_point_location(int x, int y, int height)` and `Fl::reset_marked_text()` only under the macOS platform to indicate/reset the coordinates of the current insertion point. This is deprecated in version 1.4 because redundant with the platform-independent `fl_set_spot()` and `fl_reset_spot()` functions.
31.3 Events handling functions

31.3.2.6 compose_reset()

`void Fl::compose_reset() [static]`
If the user moves the cursor, be sure to call `Fl::compose_reset()`. The next call to `Fl::compose()` will start out in an initial state. In particular it will not set "del" to non-zero. This call is very fast so it is ok to call it many times and in many places.

31.3.2.7 disable_im()

`void Fl::disable_im() [static]`
Disables the system input methods facilities.

See also

`enable_im()`

31.3.2.8 enable_im()

`void Fl::enable_im() [static]`
Enables the system input methods facilities. This is the default.

See also

`disable_im()`

31.3.2.9 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.

31.3.2.10 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()`

31.3.2.11 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()`.

31.3.2.12 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().

31.3.2.13 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().

31.3.2.14 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] }

31.3.2.15 event_clicks() [1/2]

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

Return values

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

31.3.2.16 event_clicks() [2/2]

static void Fl::event_clicks ( __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()

31.3.2.17 event_clipboard()

static void* Fl::event_clipboard ( ) [inline], [static]
During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
The returned data is an Fl_RGB_Image * when the result of Fl::event_clipboard_type() is Fl::clipboard_image.

31.3.2.18 event_clipboard_type()

static const char* Fl::event_clipboard_type ( ) [inline], [static]
31.3.2.19 event_dispatch()

```cpp
class Fl {
public:
  void event_dispatch(Fl_Event_Dispatch d) [static];
};
```

Set a new event dispatch function.

The event dispatch function is called after native events are converted to FLTK events, but before they are handled by FLTK. If the dispatch function `Fl_Event_Dispatch d` is set, it is up to the dispatch function to call `Fl::handle_(int, Fl_Window*)` or to ignore the event.

The dispatch function itself must return 0 if it ignored the event, or non-zero if it used the event. If you call `Fl::handle_()`, then this will return the correct value.

The event dispatch can be used to handle exceptions in FLTK events and callbacks before they reach the native event handler:

```cpp
int myHandler(int e, Fl_Window *w) {
  try {
    return Fl::handle_(e, w);
  } catch (...) {
    ...
  }
}
```

```cpp
main() {
  Fl::event_dispatch(myHandler);
  ...
  Fl::run();
}
```

**Parameters**

- `d` new dispatch function, or NULL

**See also**

- `Fl::add_handler(Fl_Event_Handler)`
- `Fl::handle(int, Fl_Window*)`
- `Fl::handle_(int, Fl_Window*)`

31.3.2.20 event_dx()

```cpp
static int event_dx() [inline], [static];
```

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

31.3.2.21 event_dy()

```cpp
static int event_dy() [inline], [static];
```

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

31.3.2.22 event_inside() [1/2]

```cpp
int event_inside(const Fl_Widget *o) [static];
```

Returns whether or not the mouse event is inside a given child widget.

Returns non-zero if the current `Fl::event_x()` and `Fl::event_y()` put it inside the given child widget's bounding box.

This method can only be used to check whether the mouse event is inside a child widget of the window that handles the event, and there must not be an intermediate subwindow (i.e. the widget must not be inside a subwindow of the current window). However, it is valid if the widget is inside a nested `Fl_Group`.

You must not use it with the window itself as the `o` argument in a window's `handle()` method.
Note

The mentioned restrictions are necessary, because this method does not transform coordinates of child widgets, and thus the given widget o must be within the same window that is handling the current event. Otherwise the results are undefined.

You should always call this rather than doing your own comparison so you are consistent about edge effects.

See also

Fl::event_inside(int, int, int)

Parameters

in  o  child widget to be tested

Returns

non-zero, if mouse event is inside the widget

31.3.2.23  event_inside() [2/2]

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

xx,yy,ww,hh  bounding box

Returns

non-zero, if mouse event is inside

31.3.2.24  event_is_click() [1/2]

static int Fl::event_is_click () [inline], [static]

Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".

You can test this on FL_DRAG, FL_RELEASE, and FL_MOVE events.

31.3.2.25  event_is_click() [2/2]

static void Fl::event_is_click (  
  int i ) [inline], [static]

Clears the value returned by Fl::event_is_click().

Useful to prevent the next click from being counted as a double-click or to make a popup menu pick an item with a single click. Don’t pass non-zero to this.
31.3 Events handling functions

31.3.2.26 event_key() [1/2]

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

Gets which key on the keyboard was last pushed.

The returned integer 'key code' is not necessarily a text equivalent for the keystroke. For instance: if someone presses '5' on the numeric keypad with numlock on, Fl::event_key() may return the 'key code' for this key, and NOT the character '5'. To always get the '5', use Fl::event_text() instead.

Returns

an integer 'key code', or 0 if the last event was not a key press or release.

See also

int event_key(int), event_text(), compose(int&).

31.3.2.27 event_key() [2/2]

```c
int Fl::event_key ( FL::event_key ( int key ) [static]
```

Returns true if the given key was held down (or pressed) during the last event.

This is constant until the next event is read from the server.

Fl::get_key(int) returns true if the given key is held down now. Under X this requires a round-trip to the server and is much slower than Fl::event_key(int).

Keys are identified by the unshifted values. FLTK defines a set of symbols that should work on most modern machines for every key on the keyboard:

- All keys on the main keyboard producing a printable ASCII character use the value of that ASCII character (as though shift, ctrl, and caps lock were not on). The space bar is 32.
- All keys on the numeric keypad producing a printable ASCII character use the value of that ASCII character plus FL_KP (e.g., FL_KP + '4', FL_KP + '?'). The highest possible value is FL_KP_Last so you can range-check to see if something is on the keypad.
- All numbered function keys use the number on the function key plus FL_F. The highest possible number is FL_F_Last, so you can range-check a value.
- Buttons on the mouse are considered keys, and use the button number (where the left button is 1) plus FL_Button.
- All other keys on the keypad have a symbol: FL_Escape, FL_BackSpace, FL_Tab, FL_Enter, FL_Print, FL←_Scroll_Lock, FL_Pause, FL_Insert, FL_Home, FL_Page_Up, FL_Delete, FL_End, FL_Page_Down, FL_Left, FL_Up, FL_Right, FL_Down, FL_ISO_Key, FL_Shift_L, FL_Shift_R, FL_Control_L, FL_Control_R, FL_Caps←_Lock, FL_Alt_L, FL_Alt_R, FL_Meta_L, FL_Meta_R, FL_Menu, FL_Num_Lock, FL_KP_Enter. Be careful not to confuse these with the very similar, but all-caps, symbols used by Fl::event_state().

On X Fl::get_key(FL_Button+n) does not work.

On Windows Fl::get_key(FL_KP_Enter) and Fl::event_key(FL_KP_Enter) do not work.

31.3.2.28 event_length()

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

Returns the length of the text in Fl::event_text().

There will always be a nul at this position in the text. However there may be a nul before that if the keystroke translates to a nul character or you paste a nul character.

31.3.2.29 event_original_key()

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

Returns the keycode of the last key event, regardless of the NumLock state.

If NumLock is deactivated, FLTK translates events from the numeric keypad into the corresponding arrow key events. event_key() returns the translated key code, whereas event_original_key() returns the keycode before NumLock translation.
31.3.2.30  \texttt{event\_state() [1/2]}

\begin{verbatim}
static int Fl::event_state ( ) [inline], [static]
\end{verbatim}

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.

31.3.2.31  \texttt{event\_state() [2/2]}

\begin{verbatim}
static int Fl::event_state ( int \texttt{mask} ) [inline], [static]
\end{verbatim}

Returns non-zero if any of the passed event state bits are turned on.
Use \texttt{mask} to pass the event states you're interested in. The legal event state bits are defined in \texttt{Fl::event\_state()}.  

31.3.2.32  \texttt{event\_text()}

\begin{verbatim}
static const char * Fl::event_text ( ) [inline], [static]
\end{verbatim}

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 \texttt{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 \texttt{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.

31.3.2.33  \texttt{event\_x\_root()}

\begin{verbatim}
static int Fl::event\_x\_root ( ) [inline], [static]
\end{verbatim}

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 \texttt{event\_x\_root()},\texttt{event\_y\_root()}
and \texttt{event\_x()},\texttt{event\_y()}.  

}
31.3 Events handling functions

31.3.2.34  \texttt{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 \texttt{event\_x\_root()},\texttt{event\_y\_root()} and \texttt{event\_x()},\texttt{event\_y()}.

31.3.2.35  \texttt{focus()} [1/2]

static Fl\_Widget* Fl::focus () [inline], [static]

Gets the current Fl\_focus() widget.

See also

\texttt{Fl\_focus(Fl\_Widget*)}

31.3.2.36  \texttt{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 \texttt{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

\texttt{Fl\_Widget::take\_focus()}

31.3.2.37  \texttt{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

\texttt{event\_key(int)}

31.3.2.38  \texttt{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.

31.3.2.39  \texttt{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 \texttt{Fl::handle\_}(\texttt{int}, \texttt{Fl\_Window*}) unless the user sets a dispatch function. If a user dispatch function is set, the user must make sure that \texttt{Fl::handle\_()} is called, or the event will be ignored.

### Parameters

| \texttt{e} | the event type (\texttt{Fl::event\_number()} is not yet set) |
| \texttt{window} | the window that caused this event |

### Returns

0 if the event was not handled

### See also

\texttt{Fl::add\_handler(Fl\_Event\_Handler)}

\texttt{Fl::event\_dispatch(Fl\_Event\_Dispatch)}

---

#### 31.3.2.40 \texttt{handle\_()}

\texttt{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 \texttt{Fl::handle\_}(\texttt{int}, \texttt{Fl\_Window*})

### Parameters

| \texttt{e} | the event type (\texttt{Fl::event\_number()} is not yet set) |
| \texttt{window} | the window that caused this event |

### Returns

0 if the event was not handled

### See also

\texttt{Fl::event\_dispatch(Fl\_Event\_Dispatch)}

---

#### 31.3.2.41 \texttt{pushed()} [1/2]

\texttt{static Fl\_Widget* Fl::pushed ( ) [inline], [static]}

Gets the widget that is being pushed.

### See also

\texttt{void pushed(Fl\_Widget*)}

---

#### 31.3.2.42 \texttt{pushed()} [2/2]

\texttt{void Fl::pushed ( Fl\_Widget * o ) [static]}

Generated by Doxygen
31.3 Events handling functions

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()).

31.3.2.43 remove_handler()

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

See also
Fl::handle(int, Fl_Window*)

31.3.2.44 remove_system_handler()

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)

31.3.2.45 testShortcut()

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().

Returns
non-zero if there is a match.

31.3.3 Variable Documentation

31.3.3.1 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]"
    [..]
}
```
31.3.3.2 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:
#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)"
}

31.4 Selection & Clipboard functions

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

Functions

- static void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data=0)
  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
- static int Fl::clipboard_contains (const char *type)
  Returns non 0 if the clipboard contains data matching type.
- static void Fl::copy (const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text)
  Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).
- static int Fl::dnd ()
  Initiate a Drag And Drop operation.
- static void Fl::paste (Fl_Widget &receiver)
  Backward compatibility only.
- static void Fl::paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)
  Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.
- static void Fl::remove_clipboard_notify (Fl_Clipboard_Notify_Handler h)
  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
- static void Fl::selection (Fl_Widget &owner, const char *, int len)
  Changes the current selection.
- static Fl_Widget * Fl::selection_owner ()
  Back-compatibility only: Gets the widget owning the current selection
- static void Fl::selection_owner (Fl_Widget *)
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
31.4 Selection & Clipboard functions

Variables

- static char const * Fl::clipboard_image = "image"
  Denotes image data.
- static char const * Fl::clipboard_plain_text = "text/plain"
  Denotes plain textual data.

31.4.1 Detailed Description

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

31.4.2 Function Documentation

31.4.2.1 add_clipboard_notify()

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

FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard. The source argument indicates which of the two has changed. Only changes by other applications are reported.

Example:

```c
void clip_callback(int source, void *data) {
    if ( source == 0 ) printf("CLIP CALLBACK: selection buffer changed\n");  
    if ( source == 1 ) printf("CLIP CALLBACK: clipboard changed\n");
}
int main() {

..}  
    Fl::add_clipboard_notify(clip_callback);  
..}  
}
```

Note

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

31.4.2.2 clipboard_contains()

```c
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`.

31.4.2.3 copy()

```c
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.

### 31.4.2.4 dnd()

```cpp
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)`

### 31.4.2.5 paste() [1/2]

```cpp
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)`

### 31.4.2.6 paste() [2/2]

```cpp
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.

31.4.2.7 selection()

```cpp
void Fl::selection (  
    Fl_Widget & owner,  
    const char * text,  
    int len ) [static]
```

Changes the current selection.
The block of text is copied to an internal buffer by FLTK (be careful if doing this in response to an FL_PASTE as this may be the same buffer returned by event_text()). The selection_owner() widget is set to the passed owner.

31.4.2.8 selection_owner() [1/2]

```cpp
static Fl_Widget * Fl::selection_owner ( ) [inline], [static]
```

back-compatibility only: Gets the widget owning the current selection

See also

```cpp
Fl_Widget* selection_owner(Fl_Widget*)
```

31.4.2.9 selection_owner() [2/2]

```cpp
void Fl::selection_owner (  
    Fl_Widget * owner ) [static]
```

Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection. FL_SELECTIONCLEAR is sent to the previous selection owner, if any. Copying the buffer every time the selection is changed is obviously wasteful, especially for large selections. An interface will probably be added in a future version to allow the selection to be made by a callback function. The current interface will be emulated on top of this.

31.5 Screen functions

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

**Functions**

- static int Fl::h ()
  
  Returns the height in pixels of the main screen work area.
- static void Fl::keyboard_screen_scaling (int value)
  
  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- static int Fl::screen_count ()
  
  Gets the 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)
**Detailed Description**

FLTK supports high-DPI screens using a screen scaling factor. The scaling factor is initialized by the library to a value based on information obtained from the OS. If this initial value is not satisfactory, the FLTK_SCALING_FACTOR environment variable can be set to a value FLTK will multiply to the OS-given value. The 2 variants of functions Fl::screen_scale() allow to programmatically get and set scaling factor values. The scaling factor value can be further changed at runtime by typing ctrl/+/-/0/ (cmd/+/-/0/ under macOS). FLTK sends the FL_ZOOM_EVENT when the factor value is changed, to which a callback can be associated with Fl::add_handler(). By default, FLTK displays the new scaling factor value in a yellow, transient window. This can be changed with option Fl::OPTION_SHOW_SCALING.

**Function Documentation**

31.5.2.1 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.
31.5 Screen functions

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. |

31.5.2.2 screen_dpi()

```cpp
void Fl::screen_dpi (  
    float & h,  
    float & v,  
    int n = 0 ) [static]
```

Gets the screen resolution in dots-per-inch for the given screen.

Parameters

| out | h,v | horizontal and vertical resolution |
| in | n   | the screen number (0 to Fl::screen_count() - 1) |

See also

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

31.5.2.3 screen_num() [1/2]

```cpp
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 |

31.5.2.4 screen_num() [2/2]

```cpp
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 |
### 31.5.2.5 `screen_scale()`

```c
void Fl::screen_scale ( 
    int n, 
    float factor ) [static]
```

Sets the value of the GUI scaling factor for screen number `n`. Also sets the scale factor value of all windows mapped to screen number `n`, if any.

### 31.5.2.6 `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)`

### 31.5.2.7 `screen_work_area()` [1/3]

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

Gets the bounding box of the work area of the screen that contains the mouse pointer.

**Parameters**

- `out X, Y, W, H` the work area bounding box

See also

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

### 31.5.2.8 `screen_work_area()` [2/3]

```c
void Fl::screen_work_area ( 
    int & X, 
    int & Y, 
    int & W, 
    int & H, 
    int mx, 
    int my ) [static]
```

Gets the bounding box of the work area of a screen that contains the specified screen position `mx, my`.

**Parameters**

- `out X, Y, W, H` the work area bounding box
- `in mx, my` the absolute screen position
### 31.5.2.9 screen_work_area() [3/3]

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

Gets the bounding box of the work area of the given screen.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
<td>the work area bounding box</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the screen number (0 to Fl::screen_count() - 1)</td>
</tr>
</tbody>
</table>

**See also**

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

### 31.5.2.10 screen_xywh() [1/4]

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

Gets the bounding box of a screen that contains the mouse pointer.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
<td>the corresponding screen bounding box</td>
</tr>
</tbody>
</table>

**See also**

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

### 31.5.2.11 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**

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
<td>the corresponding screen bounding box</td>
</tr>
<tr>
<td>in</td>
<td>mx, my</td>
<td>the absolute screen position</td>
</tr>
</tbody>
</table>
31.5.2.12  screen_xywh() [3/4]

void Fl::screen_xywh (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int mx,  
    int my,  
    int mw,  
    int mh )  [static]

Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by \( mx, my, mw, mh \).

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mx, my, mw, mh</td>
<td>the rectangle to search for intersection with</td>
</tr>
</tbody>
</table>

See also

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

31.5.2.13  screen_xywh() [4/4]

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

Gets the screen bounding rect for the given screen.
Under Windows, Mac OS X, and the Gnome desktop, screen #0 contains the menubar/taskbar

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X, Y, W, H</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the screen number (0 to Fl::screen_count() - 1)</td>
</tr>
</tbody>
</table>

See also

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

31.6  Color & Font functions

fl global color, font functions.

Functions

- **FL_Color fl_color ()**
  
  Return the last fl_color() that was set.
- **void fl_color (FL_Color c)**
  
  Set the color for all subsequent drawing operations.
- **void fl_color (int c)**
  
  for back compatibility - use fl_color(FL_Color c) instead
31.6 Color & Font functions

- **void fl_color (uchar r, uchar g, uchar b)**
  Set the color for all subsequent drawing operations.

- **Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)**
  Returns the weighted average color between the two given colors.

- **Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg)**
  Returns a color that contrasts with the background color.

- **int fl_descent ()**
  Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.

- **Fl_Font fl_font ()**
  Return the face set by the most recent call to fl_font().

- **FL_EXPORT void fl_font (Fl_Font face, Fl_Fontsize fsize)**
  Sets the current font, which is then used in various drawing routines.

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

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

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

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

- **FL_EXPORT const char * fl_local_to_latin1 (const char * t, int n=-1)**
  Convert 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)**
  Convert text from local encoding to Mac Roman character set.

- **FL_EXPORT const char * fl_mac_roman_to_local (const char * t, int n=-1)**
  Convert 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 ()**
  Return 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)**
  Determine 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)**
  Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().

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

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

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

- **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 void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)**
  Returns the RGBA value(s) for the given FLTK color index.
• static const char * Fl::get_font_name (Fl_Font, int *attributes=0)
  Get a human-readable string describing the family of this face.
• static int Fl::get_font_sizes (Fl_Font, int *&sizep)
  Return an array of sizes in sizep.
• static void Fl::set_color (Fl_Color i, unsigned c)
  Sets an entry in the fl_color_index table.
• static void Fl::set_color (Fl_Color, uchar, uchar, uchar)
  Sets an entry in the fl_color_index table.
• static void Fl::set_color (Fl_Color, uchar, uchar, uchar, uchar)
  Sets an entry in the fl_color_index table.
• static void Fl::set_font (Fl_Font, const char *)
  Changes a face.
• static void Fl::set_font (Fl_Font, Fl_Font)
  Copies one face to another.
• static Fl_Font Fl::set_fonts (const char * =0)
  FLTK will open the display, and add every fonts on the server to the face table.

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

31.6.2 Function Documentation

31.6.2.1 fl_color() [1/3]
Fl_Color fl_color ( ) [inline]
Return the last fl_color() that was set.
This can be used for state save/restore.

31.6.2.2 fl_color() [2/3]
void fl_color (  
    Fl_Color c ) [inline]
Set the color for all subsequent drawing operations.
For colormapped displays, a color cell will be allocated out of fl_colormap the first time you use a color. If the
colormap fills up then a least-squares algorithm is used to find the closest color. If no valid graphical context (fl_gc)
is available, the foreground is not set for the current window.

Parameters

| in | c | color |

31.6.2.3 fl_color() [3/3]
void fl_color (  
    uchar r,  
    uchar g,  
    uchar b ) [inline]
Set the color for all subsequent drawing operations.
The closest possible match to the RGB color is used. The RGB color is used directly on TrueColor displays. For
colormap visuals the nearest index in the gray ramp or color cube is used. If no valid graphical context (fl_gc) is
available, the foreground is not set for the current window.

Parameters

| in  | r,g,b | color components |

### 31.6.2.4 fl_color_average()

```c
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:

\[
    \text{color} = \text{color1} \cdot \text{weight} + \text{color2} \cdot (1 - \text{weight})
\]

Thus, a weight value of 1.0 will return the first color, while a value of 0.0 will return the second color.

Parameters

| in  | color1,color2 | boundary colors |
| in  | weight         | weighting factor |

### 31.6.2.5 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

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

Returns

contrasting color

### 31.6.2.6 fl_font() [1/2]

```c
Fl_Font fl_font ( ) [inline]
```

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

### 31.6.2.7 fl_font() [2/2]

```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.

31.6.2.8 fl_height() [1/2]

int fl_height ( ) [inline]
Return the recommended minimum line spacing for the current font.
You can also use the value of size passed to fl_font().

31.6.2.9 fl_height() [2/2]

FL_EXPORT int fl_height ( 
  int font,
  int size )
This function returns the actual height of the specified font and size.
Normally the font height should always be 'size', but with the advent of XFT, there are (currently) complexities that seem to only be solved by asking the font what its actual font height is. (See STR#2115)
This function was originally undocumented in 1.1.x, and was used only by Fl_Text_Display. We're now documenting it in 1.3.x so that apps that need precise height info can get it with this function.

Returns
the height of the font in pixels.

Todo In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

31.6.2.10 fl_latin1_to_local()

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

Parameters

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

Returns
pointer to internal buffer containing converted characters

31.6.2.11 fl_local_to_latin1()

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

Parameters

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

pointer to internal buffer containing converted characters

31.6.2.12 fl_local_to_mac_roman()

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

Convert 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

31.6.2.13 fl_mac_roman_to_local()

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

Convert text from Mac Roman character set to local encoding.

Parameters

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

Returns

pointer to internal buffer containing converted characters

31.6.2.14 fl_show_colormap()

FL_EXPORT Fl_Color fl_show_colormap(
    Fl_Color oldcol)

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

\texttt{in oldcol} color to be highlighted when grid is shown.

Return values

\texttt{Fl\_Color} value of the chosen colormap entry.

See also

\texttt{Fl\_Color\_Chooser}

31.6.2.15 \texttt{fl\_size()}

\texttt{Fl\_Fontsize fl\_size()} [inline]

Return the size set by the most recent call to \texttt{fl\_font()}. This can be used to save/restore the font.

31.6.2.16 \texttt{fl\_text\_extents() [1/2]}

\texttt{FL\_EXPORT void fl\_text\_extents (}
\begin{verbatim}
    const char * c,
    int & dx,
    int & dy,
    int & w,
    int & h
\end{verbatim}

Determine the minimum pixel dimensions of a nul-terminated string using the current \texttt{fl\_font()}. Usage: given a string "txt" drawn using \texttt{fl\_draw(txt, x, y)} you would determine its pixel extents on the display using \texttt{fl\_text\_extents(txt, dx, dy, wo, ho)} such that a bounding box that exactly fits around the 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:

```cpp
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\n", W, H);
```

### 31.6.2.17 fl_text_extents() [2/2]

```cpp
void fl_text_extents ( const char * t, int n, int & dx, int & dy, int & w, int & h ) [inline]
```

Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().

**Note**

The string length is measured in bytes, not (UTF-8) characters.

**See also**

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

### 31.6.2.18 fl_width()

```cpp
double fl_width ( unsigned int c ) [inline]
```

Return the typographical width of a single character using the current font face and size.

**Note**

If a valid fl_gc is NOT found then it uses the first window gc, or the screen gc if no fltk window is available when called.

### 31.6.2.19 free_color()

```cpp
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.

### 31.6.2.20 get_color() [1/3]

```cpp
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.
31.6.21  get_color() [2/3]

void Fl::get_color (  
    Fl_Color i,  
    uchar & red,  
    uchar & green,  
    uchar & blue ) [static]

Returns the RGB value(s) for the given FLTK color index.  
This form returns the red, green, and blue values separately in referenced variables.

See also

unsigned get_color(Fl_Color c)

31.6.22  get_color() [3/3]

void Fl::get_color (  
    Fl_Color i,  
    uchar & red,  
    uchar & green,  
    uchar & blue,  
    uchar & alpha ) [static]

Returns the RGBA value(s) for the given FLTK color index.  
This form returns the red, green, blue, and alpha values separately in referenced variables.

See also

unsigned get_color(Fl_Color c)

31.6.23  get_font()

const char * Fl::get_font (  
    Fl_Font fnum ) [static]

Gets the string for this face.  
This string is different for each face. Under X this value is passed to XListFonts to get all the sizes of this face.

31.6.24  get_font_name()

const char * Fl::get_font_name (  
    Fl_Font fnum,  
    int * attributes = 0 ) [static]

Get a human-readable string describing the family of this face. 
This is useful if you are presenting a choice to the user. There is no guarantee that each face has a different name. The return value points to a static buffer that is overwritten each call. The integer pointed to by attributes (if the pointer is not zero) is set to zero, FL_BOLD or FL_ITALIC or FL_BOLD | FL_ITALIC. To locate a “family” of fonts, search forward and back for a set with non-zero attributes, these faces along with the face with a zero attribute before them constitute a family.

31.6.25  get_font_sizes()

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

Return an array of sizes in sizep. 
The return value is the length of this array. The sizes are sorted from smallest to largest and indicate what sizes can be given to fl_font() that will be matched exactly (fl_font() will pick the closest size for other sizes). A zero in the first location of the array indicates a scalable font, where any size works, although the array may list sizes that work “better” than others. Warning: the returned array points at a static buffer that is overwritten each call. Under X this will open the display.
31.6.2.26  set_color() [1/3]

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.

31.6.2.27  set_color() [2/3]

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.

31.6.2.28  set_color() [3/3]

void Fl::set_color (  
    Fl_Color i,  
    uchar red,  
    uchar green,  
    uchar blue,  
    uchar alpha ) [static]

Sets an entry in the fl_color index table.
You can set it to any 8-bit RGBA color.

Note

The color transparency is effective under the Wayland, hybrid Wayland/X11 and macOS platforms, whereas it has no effect under the X11 and Windows platforms. It’s also effective for widgets added to an Fl_Gl_Window.

Version

1.4

31.6.2.29  set_font()

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

Changes a face.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fnum</td>
<td>The font number to be assigned a new face</td>
</tr>
<tr>
<td>name</td>
<td>Name of the font to assign. The string pointer is simply stored, the string is not copied, so the string must be in static memory. The exact name to be used depends on the platform :</td>
</tr>
</tbody>
</table>

- Windows, X11, Xft: use the family name prefixed by one character to indicate the desired font variant. Characters ' ', 'I', 'B', 'P' denote plain, italic, bold, and bold-italic variants, respectively. For example, string "IGabriola" is to be used to denote the "Gabriola italic" font. The "Oblique" suffix, in whatever case, is to be treated as "italic", that is, prefix the family name with 'I'.
- Other platforms, i.e., X11 + Pango, Wayland, macOS: use the full font name as returned by function Fl::get_font_name() or as listed by applications test/fonts or test/utf8. No prefix is to be added.
31.6.2.30 set_fonts()

```c
Fl_Font Fl::set_fonts (const char ∗xfarnname = 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.

31.7 Drawing functions

FLTK global graphics and GUI drawing functions.

Enumerations

- enum {
  FL_SOLID = 0 , FL_DASH = 1 , FL_DOT = 2 , FL_DASHDOT = 3 ,
  FL_DASHDOTDOT = 4 , FL_CAP_FLAT = 0x100 , FL_CAP_ROUND = 0x200 , FL_CAP_SQUARE = 0x300 ,
  FL_JOIN_MITER = 0x1000 , FL_JOIN_ROUND = 0x2000 , FL_JOIN_BEVEL = 0x3000 }

Functions

- FL_EXPORT int fl_add_symbol (const char ∗name, void(∗drawit)(Fl_Color), int scalable)
  Adds a symbol to the system.
- int fl_antialias ()
  Return whether line drawings are currently antialiased.
- void fl_antialias (int state)
  Turn antialiased line drawings ON or OFF, if supported by platform.
- void fl_arc (double x, double y, double r, double start, double end)
  Add a series of points to the current path on the arc of a circle.
- void fl_arc (int x, int y, int w, int h, double a1, double a2)
  Draw ellipse sections using integer coordinates.
- void fl_begin_complex_polygon ()
  Start drawing a complex filled polygon.
- void fl_begin_line ()
  Start drawing a list of lines.
- void fl_begin_loop ()
  Start drawing a closed sequence of lines.
- FL_EXPORT void fl_begin_offscreen (Fl_Offscreen ctx)
  Send all subsequent drawing commands to this offscreen buffer.
- void fl_begin_points ()
  Start drawing a list of points.
- void fl_begin_polygon ()
  Start drawing a convex filled polygon.
- char fl_can_do_alpha_blending ()
  Check whether platform supports true alpha blending for RGBA images.
- FL_EXPORT Fl_RGB_Image ∗ fl_capture_window (Fl_Window ∗win, int x, int y, int w, int h)
  Captures the content of a rectangular zone of a mapped window.
• **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._

• **void fl_clip** (int x, int y, int w, int h)

  _Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated)._

• **int fl_clip_box** (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

  _Intersect a rectangle with the current clip region and return the bounding box of the result._

• **Fl_Region fl_clip_region**()

  _Return the current clipping region._

• **void fl_clip_region** (Fl_Region r)

  _Replace the top of the clipping stack with a clipping region of any shape._

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

  _Copy a rectangular area of the given offscreen buffer into the current drawing destination._

• **FL_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)

  _Add a series of points on a Bézier curve to the path._

• **FL_EXPORT void fl_delete_offscreen** (Fl_Offscreen ctx)

  _Deletion of an offscreen graphics buffer._

• **void fl_draw** (const char ∗str, int n, int x, int y)

  _Draws starting at the given x, y location a UTF-8 string of length n bytes._

• **FL_EXPORT void fl_draw** (const char ∗str, int x, int y)

  _Draw a nul-terminated UTF-8 string starting at the given x, y location._

• **FL_EXPORT void fl_draw** (const char ∗str, int x, int y, int w, int h, Fl_Align align, Fl_Image ∗img=0, int draw_symbols=1)

  _Fancy string drawing function which is used to draw all the labels._

• **FL_EXPORT void fl_draw** (const char ∗str, int x, int y, int w, int h, Fl_Align align, Fl_Image ∗img=0, int draw_∗symbols=1)

  _Fancy string drawing function which is used to draw all the labels._

• **void fl_draw** (int angle, const char ∗str, int n, int x, int y)

  _Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise._

• **FL_EXPORT void fl_draw** (int angle, const char ∗str, int x, int y)

  _Draw a nul-terminated UTF-8 string starting at the given x, y location rotating angle degrees counter-clockwise._

• **FL_EXPORT void fl_draw_box** (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)

  _Draws a box using given type, position, size and color._

• **void fl_draw_check** (Fl_Rect bb, Fl_Color col)

  _Draw a check mark inside the given bounding box._

• **void fl_draw_image** (const uchar ∗buf, int X, int Y, int W, int H, int D=3, int L=0)

  _Draw an 8-bit per color RGB or luminance image._

• **void fl_draw_image** (Fl_Drawing_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=3)

  _Draw an image using a callback function to generate image data._

• **void fl_draw_image_mono** (const uchar ∗buf, int X, int Y, int W, int H, int D=1, int L=0)

  _Draw a gray-scale (1 channel) image._

• **void fl_draw_image_mono** (Fl_Drawing_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=1)

  _Draw a gray-scale image using a callback function to generate image data._

• **int fl_draw_pixmap** (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_drawPixmap (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.

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

- void fl_end_complex_polygon ()
  End complex filled polygon, and draw.

- void fl_end_line ()
  End list of lines, and draw.

- void fl_end_loop ()
  End closed sequence of lines, and draw.

- FL_EXPORT void fl_end_offscreen ()
  Quit sending drawing commands to the current offscreen buffer.

- void fl_end_points ()
  End list of points, and draw.

- void fl_end_polygon ()
  End convex filled polygon, and draw.

- 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 ()
  Separate loops of the path.

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

- void fl_line (int x, int y, int x1, int y1, int x2, int y2)
  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)

- void fl_line_style (int style, int width=0, char *dashes=0)
  Set how to draw lines (the "pen").

- void fl_loop (int x, int y, int x1, int y1, int x2, int y2)
  Outline a 3-sided polygon with lines.

- void fl_loop (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Outline a 4-sided polygon with lines.

- 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_measurePixmap (char *const *data, int &w, int &h)
  Get the dimensions of a pixmap.

- FL_EXPORT int fl_measurePixmap (const char *const *cdata, int &w, int &h)
  Get the dimensions of a pixmap.

- void fl_mult_matrix (double a, double b, double c, double d, double x, double y)
  Concatenate another transformation onto the current one.

- int fl_not_clipped (int x, int y, int w, int h)
  Does the rectangle intersect the current clip region?

- 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.

- FL_EXPORT float fl_override_scale ()
  Removes any GUI scaling factor in subsequent drawing operations.

- void fl_pie (int x, int y, int w, int h, double a1, double a2)
  Draws 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)
  Draws a 3-sided polygon.

- void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Draws 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)
  Intersect the current clip region with a rectangle and push 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 (Fl_Rect r)
  Draws a 1-pixel border inside the given bounding box.

- 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 (Fl_Rect bb, uchar r, uchar g, uchar b)
  Colors a rectangle with "exactly" the passed r, g, b color.

- void fl_rectf (Fl_Rect r)
  Colors with current color a rectangle that exactly fills the given bounding box.

- void fl_rectf (Fl_Rect r, Fl_Color c)
  Colors with passed color a rectangle that exactly fills 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.

- 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)
  Resets marked text.

- void fl_restore_clip ()
  Undos any clobbering of the clip region done by your program.

- FL_EXPORT void fl_restore_scale (float s)
  Restores the GUI scaling factor and the clipping region in subsequent drawing operations.
• void fl_rotate (double d)
  Concatenate rotation transformation onto the current one.

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

• void fl_scale (double x)
  Concatenate scaling transformation onto the current one.

• void fl_scale (double x, double y)
  Concatenate scaling transformation onto the current one.

• FL_EXPORT void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void(*draw_area)(void *, int, int, int, void *data)
  Scroll a rectangle and draw the newly exposed portions.

• FL_EXPORT void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)
  Inform text input methods about the current text insertion cursor.

• 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.

• double fl_transform_dx (double x, double y)
  Transform distance using current transformation matrix.

• double fl_transform_dy (double x, double y)
  Transform distance using current transformation matrix.

• double fl_transform_x (double x, double y)
  Transform coordinate using the current transformation matrix.

• double fl_transform_y (double x, double y)
  Transform coordinate using the current transformation matrix.

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

• void fl_translate (double x, double y)
  Concatenate translation transformation onto the current one.

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

• void fl_xyline (int x, int y, int x1)
  Draw a horizontal line from (x,y) to (x1,y).

• void fl_xyline (int x, int y, int x1, int y2)
  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).

• void fl_xyline (int x, int y, int x1, int y2, int x3)
  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).

• void fl_yxline (int x, int y, int y1)
  Draw a vertical line from (x,y) to (x,y1).

• void fl_yxline (int x, int y, int y1, int x2)
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).

• void fl_yxline (int x, int y, int y1, int x2, int y3)
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

### 31.7.1 Detailed Description

FLTK global graphics and GUI drawing functions.
These functions are declared in `<FL/fl_draw.H>`, and in `<FL/platform.H>` for offscreen buffer-related ones.
31.7 Drawing functions 279

31.7.2 Enumeration Type Documentation

31.7.2.1 anonymous enum

Anonymous enum

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_SOLID</td>
<td>line style: _______________</td>
</tr>
<tr>
<td>FL_DASH</td>
<td>line style: _ _ _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_DOT</td>
<td>line style: . . . . . . .</td>
</tr>
<tr>
<td>FL_DASHDOT</td>
<td>line style: _ _ _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_DASHDOTDOT</td>
<td>line style: _ _ _ _ _ _ _ _</td>
</tr>
<tr>
<td>FL_CAP_FLAT</td>
<td>cap style: end is flat</td>
</tr>
<tr>
<td>FL_CAP_ROUND</td>
<td>cap style: end is round</td>
</tr>
<tr>
<td>FL_CAP_SQUARE</td>
<td>cap style: end wraps end point</td>
</tr>
<tr>
<td>FL_JOIN_MITER</td>
<td>join style: line join extends to a point</td>
</tr>
<tr>
<td>FL_JOIN_ROUND</td>
<td>join style: line join is rounded</td>
</tr>
<tr>
<td>FL_JOIN_BEVEL</td>
<td>join style: line join is tidied</td>
</tr>
</tbody>
</table>

31.7.3 Function Documentation

31.7.3.1 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of symbol (without the &quot;@&quot;)</td>
</tr>
<tr>
<td>drawit</td>
<td>function to draw symbol</td>
</tr>
<tr>
<td>scalable</td>
<td>set to 1 if drawit uses scalable vector drawing</td>
</tr>
</tbody>
</table>

Returns

1 on success, 0 on failure

31.7.3.2 fl_antialias()

```c
void fl_antialias (int state ) [inline]
```

Turn antialiased line drawings ON or OFF, if supported by platform. Currently, only the Windows platform allows to change whether line drawings are antialiased. Turning it OFF may accelerate heavy drawing operations.
31.7.3.3 fl_arc() [1/2]

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

Add a series of points to the current path on the arc of a circle.
You can get elliptical paths by using scale and rotate before calling fl_arc().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,r</th>
<th>center and radius of circular arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>start,end</td>
<td>angles of start and end of arc measured in degrees counter-clockwise from 3 o’clock. If end is less than start then it draws the arc in a clockwise direction.</td>
</tr>
</tbody>
</table>

![Figure 31.2 fl_arc(x,y,r,a1,a2)](image)

Examples:

    // Draw an arc of points
    fl_begin_points();
    fl_arc(100.0, 100.0, 50.0, 0.0, 180.0);
    fl_end_points();
    // Draw arc with a line
    fl_begin_line();
    fl_arc(200.0, 100.0, 50.0, 0.0, 180.0);
    fl_end_line();
    // Draw filled arc
    fl_begin_polygon();
    fl_arc(300.0, 100.0, 50.0, 0.0, 180.0);
    fl_end_polygon();

31.7.3.4 fl_arc() [2/2]

    void fl_arc {
        int x,
        int y,
        int w,
        int h,
        double a1,
        double a2 ) [inline]

Draw ellipse sections using integer coordinates.
These functions match the rather limited circle drawing code provided by X and Windows. The advantage over using 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>
<tr>
<td>(a_1, a_2)</td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. (a_2) must be greater than or equal to (a_1).</td>
</tr>
</tbody>
</table>

![Figure 31.3 fl\_pie() and fl\_arc()](image)

### 31.7.3.5 \texttt{fl\_begin\_complex\_polygon()}

```c
void fl_begin_complex_polygon ( ) [inline]
```

Start drawing a complex filled polygon.

The polygon may be concave, may have holes in it, or may be several disconnected pieces. Call \texttt{fl\_gap()} to separate loops of the path.

To outline the polygon, use \texttt{fl\_begin\_loop()} and replace each \texttt{fl\_gap()} with \texttt{fl\_end\_loop();fl\_begin\_loop()} pairs.

#### Note

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

### 31.7.3.6 \texttt{fl\_begin\_offscreen()}

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

Send all subsequent drawing commands to this offscreen buffer.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ctx)</td>
<td>the offscreen buffer.</td>
</tr>
</tbody>
</table>

#### Note

The \(ctx\) argument must have been created by \texttt{fl\_create\_offscreen()}. 

### 31.7.3.7 \texttt{fl\_begin\_points()}

```c
void fl_begin_points ( ) [inline]
```
Start drawing a list of points. Points are added to the list with `fl_vertex()`.

### 31.7.3.8 `fl_can_do_alpha_blending()`

```c
char fl_can_do_alpha_blending () [inline]
```

Check whether platform supports true alpha blending for RGBA images.

**Returns**

- 1 if true alpha blending supported by platform
- 0 not supported so FLTK will use screen door transparency

### 31.7.3.9 `fl_capture_window()`

```c
FL_EXPORT Fl_RGB_Image * fl_capture_window (  
    Fl_Window * win,  
    int x,  
    int y,  
    int w,  
    int h )
```

Captures the content of a rectangular zone of a mapped window.

**Parameters**

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>a mapped <code>Fl_Window</code> (derived types including <code>Fl_Gl_Window</code> are also possible)</td>
</tr>
<tr>
<td><code>x,y,w,h</code></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

### 31.7.3.10 `fl_circle()`

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

`fl_circle(x,y,r)` is equivalent to `fl_arc(x,y,r,0,360)`, but may be faster.

**Parameters**

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in x,y,r</code></td>
<td>center and radius of circle</td>
</tr>
</tbody>
</table>

**Note**

`fl_circle()` is best used as part of the Drawing Complex Shapes API, that is, flanked by `fl_begin_XXX()` and `fl_end_XXX()` calls where XXX can be 'loop' or 'polygon' to draw, respectively a circle or a disk. Transformation functions (e.g., `fl_scale(double, double)`) can be also used for `fl_circle()` to draw empty of filled ellipses. It must be the only thing in the path: if you want a circle as part of a complex polygon you must use `fl_arc()`.
Nevertheless, `fl_circle()` can also be used by itself to draw circles.

### 31.7.3.11 fl_clip()

```c
void fl_clip ( int x, int y, int w, int h ) [inline]
```

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

**Parameters**

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

**Deprecated** Please use `fl_push_clip(int x, int y, int w, int h)` instead. `fl_clip(int, int, int, int)` will be removed in FLTK 1.5.

### 31.7.3.12 fl_clip_box()

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

Intersect a rectangle with the current clip region and return the bounding box of the result. Returns non-zero if the resulting rectangle is different to the original. The given rectangle \((x, y, w, h)\) should be entirely inside its window, otherwise the result may be unexpected, i.e. this function may not clip the rectangle to the window coordinates and size. In particular \(x\) and \(y\) should not be negative.

The resulting bounding box can be used to limit the necessary drawing to this rectangle.

**Example:**

```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**

| in | x, y, w, h | position and size of rectangle |
| out | X, Y, W, H | position and size of resulting bounding box |
Returns

Non-zero if the resulting rectangle is different to the original.

See also

fl_not_clipped()

31.7.3.13  fl_clip_region() [1/2]

Fl_Region fl_clip_region ( ) [inline]
Return the current clipping region.

Note

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its return value can be always NULL if the current drawing surface is not the display.

31.7.3.14  fl_clip_region() [2/2]

void fl_clip_region ( Fl_Region r ) [inline]
Replace 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

<table>
<thead>
<tr>
<th>in</th>
<th>r</th>
<th>clipping region</th>
</tr>
</thead>
</table>

31.7.3.15  fl_copy_offscreen()

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

<table>
<thead>
<tr>
<th>x,y</th>
<th>position where to draw the copied rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>w,h</td>
<td>size of the copied rectangle</td>
</tr>
<tr>
<td>pixmap</td>
<td>offscreen buffer containing the rectangle to copy</td>
</tr>
<tr>
<td>srcx.srcy</td>
<td>origin in offscreen buffer of rectangle to copy</td>
</tr>
</tbody>
</table>
31.7 Drawing functions

31.7.3.16 fl_create_offscreen()

`FL_Offscreen fl_create_offscreen (int w, int h)`

Creation of an offscreen graphics buffer.

Parameters

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

Returns

the created graphics buffer.

The pixel size of the created graphics buffer is equal to the number of pixels in an area of the screen containing the current window sized at \( w, h \) FLTK units. This pixel size varies with the value of the scale factor of this screen.

Note

Work with the \( \text{fl}_\text{XXX}_\text{offscreen}() \) functions is equivalent to work with an \( \text{Fl}_\text{Image}_\text{Surface} \) object, as follows:

<table>
<thead>
<tr>
<th>Fl_Offscreen-based approach</th>
<th>Fl_Image_Surface-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Offscreen off = fl_create_offscreen(w, h)</td>
<td>Fl_Image_Surface *surface = new Fl_Image_Surface(w, h, 1)</td>
</tr>
<tr>
<td>fl_begin_offscreen(off)</td>
<td>Fl_Surface_Device::push_current(surface)</td>
</tr>
<tr>
<td>fl_end_offscreen()</td>
<td>Fl_Surface_Device::pop_current()</td>
</tr>
<tr>
<td>fl_copy_offscreen(x,y,w,h, off, sx,sy)</td>
<td>fl_copy_offscreen(x,y,w,h, surface-&gt;offscreen(), sx,sy)</td>
</tr>
<tr>
<td>fl_rescale_offscreen(off)</td>
<td>surface-&gt;rescale()</td>
</tr>
<tr>
<td>fl_delete_offscreen(off)</td>
<td>delete surface</td>
</tr>
</tbody>
</table>

31.7.3.17 fl_cursor()

`FL_EXPORT void fl_cursor ( Fl_Cursor c)`

Sets the cursor for the current window to the specified shape and colors. The cursors are defined in the `<FL/Enumerations.H>` header file.

31.7.3.18 fl_curve()

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

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

Parameters

| in | X0,Y0 | curve start point |
| in | X1,Y1 | curve control point |
Parameters

| in | X2,Y2 | curve control point |
| in | X3,Y3 | curve end point |

### 31.7.3.19 fl_delete_offscreen()

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

Deletion of an offscreen graphics buffer.

**Parameters**

- **ctx**: the buffer to be deleted.

**Note**

The `ctx` argument must have been created by `fl_create_offscreen()`.

### 31.7.3.20 fl_draw() [1/4]

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

Draw a null-terminated UTF-8 string starting at the given x, y location.

Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract `fl_descent()` from y. To align to the top, subtract `fl_descent()` and add `fl_height()`. This version of `fl_draw` provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

### 31.7.3.21 fl_draw() [2/4]

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

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

The string is formatted and aligned inside the passed box. Handles ‘\t’ and ‘\n’, expands all other control characters to ‘^X’, and aligns inside or against the edges of the box. See `Fl_Widget::align()` for values of `align`. The value `FL_ALIGN_INSIDE` is ignored, as this function always prints inside the box. If `img` is provided and is not NULL, the image is drawn above or below the text as specified by the `align` value. The `draw_symbols` argument specifies whether or not to look for symbol names starting with the ‘@’ character.

### 31.7.3.22 fl_draw() [3/4]

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

Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

Note

When using X11 (Unix, Linux, Cygwin et al.) this needs Xft to work. Under plain X11 (w/o Xft) rotated text is not supported by FLTK. A warning will be issued to stderr at runtime (only once) if you use this method with an angle other than 0.

31.7.3.23 fl_draw() [4/4]

FL_EXPORT void fl_draw (  
  int angle,
  const char * str,
  int x,
  int y)

Draw a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

This version of fl_draw provides direct access to the text drawing function of the underlying OS and is supported by all fltk platforms except X11 without Xft.

31.7.3.24 fl_draw_box()

FL_EXPORT void fl_draw_box (  
  Fl_Boxtype t,
  int x,
  int y,
  int w,
  int h,
  Fl_Color c)

Draws a box using given type, position, size and color.

Parameters

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

31.7.3.25 fl_draw_check()

void fl_draw_check (  
  Fl_Rect bb,
  Fl_Color col)

Draw a check mark inside the given bounding box.

The check mark is allowed to fill the entire box but the algorithm used makes sure that a 1-pixel border is kept free if the box is large enough. You need to calculate margins for box borders etc. yourself.

The check mark size is limited (minimum and maximum size) and the check mark is always centered in the given box.

Note

If the box is too small (bad GUI design) the check mark will be drawn over the box borders. This is intentional for better user experience. Otherwise users might not be able to recognize if a box is checked.

The size limits are implementation details and may be changed at any time.
Parameters

- **in** `bb` rectangle that defines the bounding box
- **in** `col` Fl_Color to draw the check mark

Since

1.4.0

### 31.7.3.26 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]
```

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

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>buf</th>
<th>points at the &quot;r&quot; data of the top-left pixel. Color data must be in r,g,b order. Luminance data is only one gray byte.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y</td>
<td>position where to put top-left corner of image</td>
</tr>
<tr>
<td>in</td>
<td>W,H</td>
<td>size of the image</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>delta to add to the pointer between pixels. It may be any value greater than or equal to 1, or it can be negative to flip the image horizontally</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>delta to add to the pointer between lines (if 0 is passed it uses W * D), and may be larger than W * D to crop data, or negative to flip the image vertically</td>
</tr>
</tbody>
</table>

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

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

Gray scale (1-channel) images may be drawn. This is done if `abs(D)` is less than 3, or by calling `fl_draw_image_mono()`. Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting `D` greater than 1 will let you display one channel of a color image.

**Note:**

The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.

### 31.7.3.27 fl_draw_image() [2/2]

```c
void fl_draw_image ( 
    Fl_Draw_Image_Cb cb, 
    void * data, 
    int X, 
    int Y, 
    int W, 
    int H, 
    int D = 3 ) [inline]
```
Draw an image using a callback function to generate image data.
You can generate the image as it is being drawn, or do arbitrary decompression of stored data, provided it can be
decompressed to individual scan lines.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cb</code></td>
<td>callback function to generate scan line data</td>
</tr>
<tr>
<td><code>data</code></td>
<td>user data passed to callback function</td>
</tr>
<tr>
<td><code>X,Y</code></td>
<td>screen position of top left pixel</td>
</tr>
<tr>
<td><code>W,H</code></td>
<td>image width and height</td>
</tr>
<tr>
<td><code>D</code></td>
<td>data size per pixel 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.

31.7.3.28 `fl_draw_image_mono()` [1/2]

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

Draw a gray-scale (1 channel) image.

See also

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

31.7.3.29 `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) [inline]
```

Draw a gray-scale image using a callback function to generate image data.

See also

`fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D).`
31.7.30  **fl_draw_pixmap()** [1/2]

```c
int fl_draw_pixmap (  
    char *const * data,  
    int x,  
    int y,  
    Fl_Color bg = FL_GRAY )  [inline]
```

Draw XPM image data, with the top-left corner at the given position.

See also

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

31.7.31  **fl_draw_pixmap()** [2/2]

```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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
</tr>
<tr>
<td>in</td>
<td>x, y</td>
</tr>
<tr>
<td>in</td>
<td>bg</td>
</tr>
</tbody>
</table>

**Returns**

0 if there was any error decoding the XPM data.

31.7.32  **fl_draw_symbol()**

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

Draw the named symbol in the given rectangle using the given color.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>label</td>
</tr>
<tr>
<td>in</td>
<td>x, y</td>
</tr>
<tr>
<td>in</td>
<td>w, h</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
</tr>
</tbody>
</table>
31.7 Drawing functions

Returns

1 on success, 0 on failure

31.7.3.33 fl_expand_text()

FL_EXPORT const char * fl_expand_text (  
    const char * from,  
    char * buf,  
    int maxbuf,  
    double maxw,  
    int & n,  
    double & width,  
    int wrap,  
    int draw_symbols )  

Copy from to buf, replacing control characters with ^X.  
Stop at a newline or if maxbuf characters written to buffer. Also word-wrap if width exceeds maxw. Returns a  
pointer to the start of the next line of characters. Sets n to the number of characters put into the buffer. Sets width  
to the width of the string in the current font.

31.7.3.34 fl_focus_rect()

void fl_focus_rect (  
    int x,  
    int y,  
    int w,  
    int h ) [inline]  

Draw a dotted rectangle, used to indicate keyboard focus on a widget.  
This method draws the rectangle in the current color and independent of the Fl::visible_focus() option. You may  
need to set the current color with fl_color() before you call this.

31.7.3.35 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

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>sets of 4 grayscale values in top, left, bottom, right order</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
</tbody>
</table>

31.7.3.36 fl_frame2()

FL_EXPORT void fl_frame2 (  
    const char * s,  
    int x,  

Generated by Doxygen
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 \( \text{fl\_frame2}() \) with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and \( \text{fl\_frame}() \) is the order of the line segments.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s )</td>
<td>sets of 4 grayscale values in bottom, right, top, left order</td>
</tr>
<tr>
<td>( x, y, w, h )</td>
<td>position and size</td>
</tr>
</tbody>
</table>

31.7.3.37 \( \text{fl\_gap()} \)

void \( \text{fl\_gap}() \) [inline]

Separate loops of the path.

It is unnecessary but harmless to call \( \text{fl\_gap}() \) before the first vertex, after the last vertex, or several times in a row.

31.7.3.38 \( \text{fl\_line\_style()} \)

void \( \text{fl\_line\_style}() \)

If you change this it is your responsibility to set it back to the default using \( \text{fl\_line\_style}(0) \).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{style} )</td>
<td>A bitmask which is a bitwise-OR of a line style, a cap style, and a join style. If you don’t specify a dash type you will get a solid line. If you don’t specify a cap or join type you will get a system-defined default of whatever value is fastest.</td>
</tr>
<tr>
<td>( \text{width} )</td>
<td>The thickness of the lines in pixels. Zero results in the system defined default, which on both X and Windows is somewhat different and nicer than 1.</td>
</tr>
<tr>
<td>( \text{dashes} )</td>
<td>A pointer to an array of dash lengths, measured in pixels. The first location is how long to draw a solid portion, the next is how long to draw the gap, then the solid, etc. It is terminated with a zero-length entry. A NULL pointer or a zero-length array results in a solid line. Odd array sizes are not supported and result in undefined behavior.</td>
</tr>
</tbody>
</table>

Note

Because of how line styles are implemented on Win32 systems, you must set the line style after setting the drawing color. If you set the color after the line style you will lose the line style settings.

The \( \text{dashes} \) array does not work under the (unsupported!) operating systems Windows 95, 98 or Me, since those operating systems do not support complex line styles.

31.7.3.39 \( \text{fl\_measure()} \)

\[ \text{FL\_EXPORT \ void \ \text{fl\_measure}()} \]

- \( \text{const char * str, } \)
- \( \text{int & w, } \)
Measure how wide and tall the string will be when printed by the \texttt{fl\_draw()} function with \texttt{align} parameter. If the incoming \texttt{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 \texttt{fl\_measure()} is called, it is advised to first set the current font with \texttt{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 \texttt{fl\_font()} explicitly in your own code, the width and height may be set to unexpected values, even zero!

\textbf{Note:} In the general use case, it's a common error to forget to set \texttt{w} to 0 before calling \texttt{fl\_measure()} when wrap behavior isn't needed.

\begin{verbatim}
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
\end{verbatim}

\subsection{31.7.3.40 fl\_measure\_ pixmap() [1/2]}

\texttt{FL\_EXPORT int fl\_measure\_ pixmap (}
\begin{verbatim}
    char *const * data,
    int & w,
    int & h )
\end{verbatim}

Get the dimensions of a pixmap.

An XPM image contains the dimensions in its data. This function returns the width and height.

\begin{verbatim}
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

\end{verbatim}

\subsection{31.7.3.41 fl\_measure\_ pixmap() [2/2]}

\begin{verbatim}
FL\_EXPORT int fl\_measure\_ pixmap (}
    const char *const * cdata,
    int & w,
    int & h )
\end{verbatim}

Get the dimensions of a pixmap.

See also

\texttt{fl\_measure\_ pixmap(char\* const\* data, int \&w, int \&h)}
31.7.3.42  fl_mult_matrix()

void fl_mult_matrix (  
    double a,  
    double b,  
    double c,  
    double d,  
    double x,  
    double y ) [inline]

Concatenate another transformation onto the current one.

Parameters

| in  | a,b,c,d,x,y   | transformation matrix elements such that \( X' = ax + cy + x \) and \( Y' = bx + dy + y \) |

31.7.3.43  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()

31.7.3.44  fl_oldShortcut()

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_ALT|FL_CTRL|'c'
"^c" -- Same as FL_CTRL|FL_META|'c'
"c" -- Same as FL_COMMAND|'c' (see FL_COMMAND for platform specific behavior)
"0x63" -- Same as "c" (hex 63=='c')
"99" -- Same as "c" (dec 99=='c')
"0143" -- Same as "c" (octal 0143=='c')
"0x63" -- Same as (FL_CTRL|'c'), or (FL_CTRL|0x63)
"99" -- Same as (FL_CTRL|'c'), or (FL_CTRL|99)
"0143" -- Same as (FL_CTRL|'c'), or (FL_CTRL|0143)
```

Remarks
Due to XForms legacy, there are some odd things to consider when using the modifier characters.
(1) You can use the special modifier keys for chords only if the modifiers are provided in this order: #, +, ^, !, @. Other ordering can yield undefined results.
So for instance, Ctrl-Alt-c must be specified as "#^c" (and not "^c"), due to the above ordering rule.
(2) If you want to make a shortcut that uses one of the special modifier characters (as the character being modified), then to avoid confusion, specify the numeric equivalent, e.g.

| If you want... | Then use...
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'#' as the shortcut..</td>
<td>&quot;0x23&quot; (instead of just &quot;)&quot;.</td>
</tr>
<tr>
<td>'+' as the shortcut..</td>
<td>&quot;0x2b&quot; (instead of just &quot;+&quot;.</td>
</tr>
<tr>
<td>^ as the shortcut..</td>
<td>&quot;0x5e&quot; (instead of just &quot;^&quot;.</td>
</tr>
<tr>
<td>Alt-+ as the shortcut..</td>
<td>&quot;0x2b&quot; (instead of &quot;+&quot;.</td>
</tr>
<tr>
<td>Alt-^ as the shortcut..</td>
<td>&quot;0x5e&quot; (instead of &quot;^&quot;.</td>
</tr>
</tbody>
</table>

As a general rule that's easy to remember, unless the shortcut key to be modified is a single alpha-numeric character [A-Z,a-z,0-9], it's probably best to use the numeric equivalents.

Don't fix these silly legacy issues in a future release. Nobody is using this anymore.

31.7.3.45 fl_override_scale()
FL_EXPORT float fl_override_scale ( )
Removes any GUI scaling factor in subsequent drawing operations.
This must be matched by a later call to fl_restore_scale(). This function can be used to transiently perform drawing operations that are not rescaled by the current value of the GUI scaling factor. The resulting drawing context has no clipping region.
Returns
The GUI scaling factor value that was in place when the function started.

31.7.3.46 fl_pie()

void fl_pie ( int x, int y, int w, int h, double a1, double a2 ) [inline]
Draw filled ellipse sections using integer coordinates.
Like fl_arc(), but fl_pie() draws a filled-in pie slice. This slice may extend outside the line drawn by fl_arc(); to avoid this use w - 1 and h - 1.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x,y,w,h</code></td>
<td>bounding box of complete circle</td>
</tr>
<tr>
<td><code>a1,a2</code></td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. <code>a2</code> must be greater than or equal to <code>a1</code>.</td>
</tr>
</tbody>
</table>

**Figure 31.4 fl_pie() and fl_arc()**

#### 31.7.3.47 fl_polygon() [1/2]

```c
void fl_polygon (  
    int x,  
    int y,  
    int x1,  
    int y1,  
    int x2,  
    int y2 } [inline]
```

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

#### 31.7.3.48 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]
```

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

#### 31.7.3.49 fl_pop_clip()

```c
void fl_pop_clip ( ) [inline]
```

Restore the previous clip region.
You must call `fl_pop_clip()` once for every time you call `fl_push_clip()`. Unpredictable results may occur if the clip stack is not empty when you return to FLTK.

#### 31.7.3.50 fl_push_clip()

```c
void fl_push_clip (  
    int x,  
    int y,  
```
31.7 Drawing functions

```c
int w,
int h ) [inline]
```

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

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
</tr>
</tbody>
</table>

31.7.3.51 fl_push_matrix()

```c
void fl_push_matrix ( ) [inline]
```

Save the current transformation matrix on the stack.

The maximum depth of the stack is 32.

31.7.3.52 fl_read_image()

```c
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
</tr>
<tr>
<td>in</td>
<td>X,Y</td>
</tr>
<tr>
<td>in</td>
<td>w,h</td>
</tr>
<tr>
<td>in</td>
<td>alpha</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 \( w \times h \times 3 \) bytes when reading RGB images, or \( w \times h \times 4 \) 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.

See also

`fl_capture_window()`

31.7.3.53 fl_rect() [1/2]

```c
void fl_rect ( 
    Fl_Rect r ) [inline]
```

Draw a 1-pixel border inside the given bounding box.

This is the same as `fl_rect(int x, int y, int w, int h)` but with `Fl_Rect r` as input argument.

31.7.3.54 fl_rect() [2/2]

```c
void fl_rect ( 
    int x,
```
int y, int w, int h) [inline]
Draw 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.

31.7.3.55 fl_rectf() [1/2]
void fl_rectf (  
  Fl_Rect bb,  
  uchar r,  
  uchar g,  
  uchar b) [inline]
Color a rectangle with "exactly" the passed r,g,b color.
This is the same as fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b) but with Fl_Rect bb (bounding box) as argument instead of (x, y, w, h).
See also
  fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)

31.7.3.56 fl_rectf() [2/2]
void fl_rectf (  
  int x,  
  int y,  
  int w,  
  int h,  
  uchar r,  
  uchar g,  
  uchar b) [inline]
Color a rectangle with "exactly" the passed r,g,b color.
On screens with less than 24 bits of color this is done by drawing a solid-colored block using fl_draw_image() so that the correct color shade is produced.

31.7.3.57 fl_rescale_offscreen()
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

31.7.3.58 fl_reset_spot()
FL_EXPORT void fl_reset_spot (  
  void)
Resets marked text.
In many languages, typing a character can involve multiple keystrokes. For example, the Ä can be composed of two dots (¨) on top of the character, followed by the letter A (on a Mac with U.S. keyboard, you’d type Alt-U, Shift-A. To inform the user that the dots may be followed by another character, the ¨ is underlined).
Call this function if character composition needs to be aborted for some reason. One such example would be the text input widget losing focus.
31.7 Drawing functions

31.7.3.59 fl_restore_scale()

FL_EXPORT void fl_restore_scale (  
   float s  )  
Restores the GUI scaling factor and the clipping region in subsequent drawing operations.

Parameters

| s | Value returned by a previous call to fl_override_scale(). |

31.7.3.60 fl_rotate()

void fl_rotate (  
   double d  )  [inline]  
Concatenate rotation transformation onto the current one.

Parameters

| in d | rotation angle, counter-clockwise in degrees (not radians) |

31.7.3.61 fl_scale() [1/2]

void fl_scale (  
   double x  )  [inline]  
Concatenate scaling transformation onto the current one.

Parameters

| in x | scale factor in both x-direction and y-direction |

31.7.3.62 fl_scale() [2/2]

void fl_scale (  
   double x,  
   double y  )  [inline]  
Concatenate scaling transformation onto the current one.

Parameters

| in x,y | scale factors in x-direction and y-direction |

31.7.3.63 fl_scroll()

FL_EXPORT void fl_scroll (  
   int X,  
   int Y,  
   int W,  
   int H,  
   int dx,  
   int dy,  
   
Generated by Doxygen
void (*)(void *, int, int, int, int) draw_area,
void * data)

Scroll a rectangle and draw the newly exposed portions.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y</th>
<th>position of top-left of rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W, H</td>
<td>size of rectangle</td>
</tr>
<tr>
<td>in</td>
<td>dx, dy</td>
<td>pixel offsets for shifting rectangle</td>
</tr>
<tr>
<td>in</td>
<td>draw_area</td>
<td>callback function to draw rectangular areas</td>
</tr>
<tr>
<td>in</td>
<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>

31.7.3.64 fl_set_spot()

FL_EXPORT void fl_set_spot (  
  int font,
  int size,
  int X,  
  int Y,  
  int W,  
  int H,
  Fl_Window * win = 0 )

Inform text input methods about the current text insertion cursor.

Parameters

| font | Font currently in use in text input. |
| size | Size of the current font. |
| X, Y | Position of the bottom of the current text insertion cursor. |
| W, H | Width and height of the current text insertion cursor. |
| win | Points to the Fl_Window object containing the current text widget, or NULL. |

31.7.3.65 fl_set_status()

FL_EXPORT void fl_set_status (  
  int X,
  int Y,
  int W,
  int H )

Todo provide user documentation for fl_set_status function

31.7.3.66 flShortcut_label() [1/2]

FL_EXPORT const char* flShortcut_label (  
  unsigned int shortcut )

Get a human-readable string from a shortcut value.
Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero then an empty string is returned. The return value points at a static buffer that is overwritten with each call.
Since FLTK 1.3.4 modifier key names can be localized, but key names can not yet be localized. This may be added to a future FLTK version.

Modifier key names (human-readable shortcut names) can be defined with the following global const char * pointer variables:

- `fl_local_ctrl` => name of FL_CTRL
- `fl_local_alt` => name of FL_ALT
- `fl_local_shift` => name of FL_SHIFT
- `fl_local_meta` => name of FL_META

```c
fl_local_ctrl = "Strg"; // German for "Ctrl"
fl_local_shift = "Umschalt"; // German for "Shift"
```

Note

Due to random static initialization order this should always be done from code in main() or called by main() as opposed to static initialization since the default strings in the FLTK library are set by static initializers. Otherwise this might result in the wrong order so FLTK’s internal initialization overwrites your strings.

The shortcut name will be constructed by adding all modifier names in the order defined above plus the name of the key. A ‘+’ character is added to each modifier name unless it has a trailing ‘\’ or a trailing ‘+’.

Example:

Ctrl+Alt+Shift+Meta+F12

The default values for modifier key names are as given above for all platforms except macOS. macOS uses graphical characters that represent the typical macOS modifier names in menus, e.g. cloverleaf, saucepan, etc. You may, however, redefine macOS modifier names as well.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>shortcut</th>
<th>the integer value containing the ASCII character or extended keystroke plus modifiers</th>
</tr>
</thead>
</table>

Returns

a pointer to a static buffer containing human readable text for the shortcut

31.7.3.67 flShortcutLabel() [2/2]

```c
FL_EXPORT const char* flShortcutLabel ( 
    unsigned int shortcut, 
    const char ** eom 
)
```

Get a human-readable string from a shortcut value.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>shortcut</th>
<th>the integer value containing the ASCII character or extended keystroke plus modifiers</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>eom</th>
<th>if this pointer is set, it will receive a pointer to the end of the modifier text</th>
</tr>
</thead>
</table>

Returns

a pointer to a static buffer containing human readable text for the shortcut

See also

`flShortcutLabel(unsigned int shortcut)`
31.7.3.68  fl_transform_dx()

double fl_transform_dx (  
    double x,  
    double y ) [inline]

Transform distance using current transformation matrix.

Parameters

**in** x, y coordinate

31.7.3.69  fl_transform_dy()

double fl_transform_dy (  
    double x,  
    double y ) [inline]

Transform distance using current transformation matrix.

Parameters

**in** x, y coordinate

31.7.3.70  fl_transform_x()

double fl_transform_x (  
    double x,  
    double y ) [inline]

Transform coordinate using the current transformation matrix.

Parameters

**in** x, y coordinate

31.7.3.71  fl_transform_y()

double fl_transform_y (  
    double x,  
    double y ) [inline]

Transform coordinate using the current transformation matrix.

Parameters

**in** x, y coordinate

31.7.3.72  fl_transformed_vertex()

void fl_transformed_vertex (  
    double xf,  
    double yf ) [inline]

Add coordinate pair to the vertex list without further transformations.
31.7.3.73  

**fl_translate()**

```c
void fl_translate ( 
    double x, 
    double y ) [inline]
```

Concatenate translation transformation onto the current one.

**Parameters**

- **x, y**: translation factor in x-direction and y-direction

31.7.3.74  

**fl_vertex()**

```c
void fl_vertex ( 
    double x, 
    double y ) [inline]
```

Add a single vertex to the current path.

**Parameters**

- **x, y**: coordinate

31.8  

**Multithreading support functions**

**fl** multithreading support functions declared in `<FL/Fl.H>`

**Functions**

- **static int Fl::awake (Fl::Awake_Handler cb, void *message=0)**
  
  See `void awake(void *message=0)`.
- **static void Fl::awake (void *message=0)**
  
  Sends a message pointer to the main thread, causing any pending `Fl::wait()` call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.
- **static int Fl::lock ()**
  
  The `lock()` method blocks the current thread until it can safely access FLTK widgets and data.
- **static void Fl::thread_message ()**
  
  The `thread_message()` method returns the last message that was sent from a child by the `awake()` method.
- **static void Fl::unlock ()**
  
  The `unlock()` method releases the lock that was set using the `lock()` method.

31.8.1  

**Detailed Description**

**fl** multithreading support functions declared in `<FL/Fl.H>`

31.8.2  

**Function Documentation**
### 31.8.2.1 awake() [1/2]

```c
int Fl::awake (Fl_Awake_Handler func, void * data = 0) [static]
```

See void awake(void* message=0).

Let the main thread know an update is pending and have it call a specific function.

Registers a function that will be called by the main thread during the next message handling cycle. Returns 0 if the callback function was registered, and -1 if registration failed. Over a thousand awake callbacks can be registered simultaneously.

See also

```c
Fl::awake(void* message=0)
```

### 31.8.2.2 awake() [2/2]

```c
void Fl::awake (void * msg = 0) [static]
```

Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

Multiple calls to Fl::awake() will queue multiple pointers for the main thread to process, up to a system-defined (typically several thousand) depth. The default message handler saves the last message which can be accessed using the Fl::thread_message() function.

In the context of a threaded application, a call to Fl::awake() with no argument will trigger event loop handling in the main thread. Since it is not possible to call Fl::flush() from a subsidiary thread, Fl::awake() is the best (and only, really) substitute.

See also: Multithreading

### 31.8.2.3 lock()

```c
int Fl::lock () [static]
```

The lock() method blocks the current thread until it can safely access FLTK widgets and data.

Child threads should call this method prior to updating any widgets or accessing data. The main thread must call lock() to initialize the threading support in FLTK. lock() will return non-zero if threading is not available on the platform.

Child threads must call unlock() when they are done accessing FLTK.

When the wait() method is waiting for input or timeouts, child threads are given access to FLTK. Similarly, when the main thread needs to do processing, it will wait until all child threads have called unlock() before processing additional data.

Returns

0 if threading is available on the platform; non-zero otherwise.

See also: Multithreading

### 31.8.2.4 thread_message()

```c
void * Fl::thread_message () [static]
```

The thread_message() method returns the last message that was sent from a child by the awake() method.

See also: Multithreading

### 31.8.2.5 unlock()

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

The unlock() method releases the lock that was set using the lock() method.

Child threads should call this method as soon as they are finished accessing FLTK.

See also: Multithreading
31.9 Safe widget deletion support functions

These functions, declared in `<FL/Fl.H>`, support deletion of widgets inside callbacks.

**Functions**

- static void Fl::clear_widget_pointer (Fl_Widget const *w)
  
  Clears a widget pointer in the watch list.

- static void Fl::delete_widget (Fl_Widget *w)
  
  Schedules a widget for deletion at the next call to the event loop.

- static void Fl::do_widget_deletion ()
  
  Deletes widgets previously scheduled for deletion.

- static void Fl::release_widget_pointer (Fl_Widget *&w)
  
  Releases a widget pointer from the watch list.

- static void Fl::watch_widget_pointer (Fl_Widget *&w)
  
  Adds a widget pointer to the widget watch list.

### 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.

### Function Documentation

#### 31.9.2.1 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

31.9.2.2 delete_widget()

```c
void Fl::delete_widget ( Fl_Widget * wi ) [static]
```

Schedules a widget for deletion at the next call to the event loop.

Use this method to delete a widget inside a callback function.
To avoid early deletion of widgets, this function should be called toward the end of a callback and only after any call to the event loop (Fl::wait(), Fl::flush(), Fl::check(), fl_ask(), etc.).
When deleting groups or windows, you must only delete the group or window widget and not the individual child widgets.

Since

FLTK 1.3.4 the widget will be hidden immediately, but the actual destruction will be delayed until the event loop is finished. Up to FLTK 1.3.3 windows wouldn't be hidden before the event loop was done, hence you had to hide() a window in your window close callback if you called Fl::delete_widget() to destroy (and hide) the window.
FLTK 1.3.0 it is not necessary to remove widgets from their parent groups or windows before calling this, because it will be done in the widget's destructor, but it is not a failure to do this nevertheless.

Note
In FLTK 1.1 you must remove widgets from their parent group (or window) before deleting them.

See also
- Fl_Widget::~Fl_Widget()

31.9.2.3 do_widget_deletion()

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

Deletes widgets previously scheduled for deletion.
This is for internal use only. You should never call this directly.
Fl::do_widget_deletion() is called from the FLTK event loop or whenever you call Fl::wait(). The previously scheduled widgets are deleted in the same order they were scheduled by calling Fl::delete_widget().

See also
- Fl::delete_widget(Fl_Widget *wi)
31.9.2.4 release_widget_pointer()

```c
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()

31.9.2.5 watch_widget_pointer()

```c
void Fl::watch_widget_pointer ( 
    Fl_Widget * & w ) [static]
```

Adds a widget pointer to the widget watch list.

**Note**

Internal use only, please use class Fl_Widget_Tracker instead.

This can be used, if it is possible that a widget might be deleted during a callback or similar function. The widget pointer must be added to the watch list before calling the callback. After the callback the widget pointer can be queried, if it is NULL. If it is NULL, then the widget has been deleted during the callback and must not be accessed anymore. If the widget pointer is not NULL, then the widget has not been deleted and can be accessed safely. After accessing the widget, the widget pointer must be released from the watch list by calling Fl::release_widget_pointer().

Example for a button that is clicked (from its handle() method):

```c
Fl_Widget * wp = this; // save 'this' in a pointer variable
Fl::watch_widget_pointer(wp); // add the pointer to the watch list
set_changed(); // set the changed flag
do_callback(); // call the callback
if (!wp) { // the widget has been deleted
    // DO NOT ACCESS THE DELETED WIDGET !
} else { // the widget still exists
    clear_changed(); // reset the changed flag
}
Fl::release_widget_pointer(wp); // remove the pointer from the watch list
```

This works, because all widgets call Fl::clear_widget_pointer() in their destructors.

See also

Fl::release_widget_pointer()

Fl::clear_widget_pointer()

An easier and more convenient method to control widget deletion during callbacks is to use the class Fl_Widget_Tracker with a local (automatic) variable.

See also

class Fl_Widget_Tracker

31.10 Cairo Support Functions and Classes

**Classes**

- class Fl_Cairo_State
  
  Contains all the necessary info on the current cairo context.

- class Fl_Cairo_Window
  
  This defines a FLTK window with cairo support.
Functions

- static bool Fl::cairo_autolink_context ()
  
  Gets the current autolink mode for cairo support.

- static void Fl::cairo_autolink_context (bool alink)
  
  when FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current cairo context.

- static cairo_t * Fl::cairo_cc ()
  
  Gets the current cairo context linked with a fltk window.

- static void Fl::cairo_cc (cairo_t *c, bool own=false)
  
  Sets the current cairo context to c.

- static cairo_t * Fl::cairo_make_current (Fl_Window *w)
  
  Provides a corresponding cairo context for window w.

31.10.1 Detailed Description

31.10.2 Function Documentation

31.10.2.1 cairo_autolink_context() [1/2]

static bool Fl::cairo_autolink_context ( ) [inline], [static]

Gets the current autolink mode for cairo support.

Return values

| false | if no cairo context autolink is made for each window. |
| true  | if any fltk window is attached a cairo context when it is current. |

See also

void cairo_autolink_context(bool alink)

Note

Only available when configure has the –enable-cairo option

31.10.2.2 cairo_autolink_context() [2/2]

static void Fl::cairo_autolink_context ( bool alink ) [inline], [static]

when FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current cairo context.

This is not the default, because it may not be necessary to add cairo support to all fltk supported windows. When you wish to associate a cairo context in this mode, you need to call explicitly in your draw() overridden method, Fl::cairo_make_current(Fl_Window*). This will create a cairo context but only for this Window. Still in custom cairo application it is possible to handle completely this process automatically by setting alink to true. In this last case, you don’t need anymore to call Fl::cairo_make_current(). You can use Fl::cairo_cc() to get the current cairo context anytime.

Note

Only available when configure has the –enable-cairo option
31.10.2.3  cairo_cc()

```c
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

31.10.2.4  cairo_make_current()

```c
cairo_t * Fl::cairo_make_current (  
    Fl_Window * wi ) [static]
```
Provides a corresponding cairo context for window wi.
This is needed in a draw() override if Fl::cairo_autolink_context() returns false, which is the default. The cairo←
context() does not need to be freed as it is freed every time a new cairo context is created. When the program terminates, a call to Fl::cairo_make_current(0) will destroy any residual context.

Note

A new cairo context is not always re-created when this method is used. In particular, if the current graphi-
cal 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.

31.11  Unicode and UTF-8 functions

fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

Macros

- `#define ERRORS_TO_CP1252 1`
  Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
- `#define ERRORS_TO_ISO8859_1 1`
  Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
- `#define NBC 0xFFFF + 1`
- `#define STRICT_RFC3629 0`
  A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.

Functions

- `FL_EXPORT int fl_access (const char *, 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 *)`
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- `FL_EXPORT int fl_chmod (const char *, 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 *, char **argv)`
• FL_EXPORT FILE * 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.
• 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 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_utf8from_mb (const wchar_t * src, unsigned srclen)
  Convert from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.
• FL_EXPORT 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.
• FL_EXPORT unsigned fl_utf8fromwc (char * dst, unsigned dstlen, const wchar_t * src, unsigned srclen)
  Convert a single Unicode codepoint into an array of 16-bit characters.
• FL_EXPORT int fl_utf8fromUTF16 (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_utf8frommbcs (const char * src, unsigned srclen)
  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8.
• FL_EXPORT int fl_utf8fromom_UTF8 (char * dst, unsigned dstlen, const wchar_t * src, unsigned srclen)
  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
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_utf8len** (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.

- **FL_EXPORT int fl_utf8len1** (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.

- **FL_EXPORT int fl_utf8locale** ()
  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_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_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.

### 31.11.1 Detailed Description

**FL** global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

### 31.11.2 Macro Definition Documentation

#### 31.11.2.1 ERRORS_TO_CP1252

```c
#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.
31.11.2.2 ERRORS_TO_ISO8859_1

#define ERRORS_TO_ISO8859_1 1
Set to 1 to turn bad UTF-8 bytes into ISO-8859-1. If this is zero they are
instead turned into the Unicode REPLACEMENT CHARACTER, of value 0xfffd. If this is on
`fl_ut8decode()` 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.

31.11.2.3 STRICT_RFC3629

#define STRICT_RFC3629 0
A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.
Turn this on will replace the bytes in those encodings with errors. If you do this then converting arbitrary 16-bit data
to UTF-8 and then back is not an identity, which will probably break a lot of software.

31.11.3 Function Documentation

31.11.3.1 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.

31.11.3.2 fl_chdir()

```c
int fl_chdir (const char ∗ path )
```

Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
This function is especially useful on the Windows platform where the standard _wchdir() function needs a path in
UTF-16 encoding. The path is converted to a system specific encoding if necessary and the system specific chdir(converted←_path) function is called.
The function returns 0 on success and -1 on error. Depending on the platform, `errno` may be set if an error occurs.

Note

The possible `errno` values are platform specific. Refer to the documentation of the platform specific chdir() function.

If the function is not implemented on a particular platform the default implementation returns -1 and `errno` is not
set.
If the `path` is NULL the function returns -1, but `errno` is not changed. This is a convenience feature of fl_chdir()
as opposed to chdir().
31.11 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)

31.11.3.3 fl_chmod()

int fl_chmod (const char * f, int mode)

Cross-platform function to set a files mode() with a UTF-8 encoded name or value. This function is especially useful 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.

31.11.3.4 fl_fopen()

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().

31.11.3.5 fl_getcwd()

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><code>buf</code></th>
<th>the buffer to populate (may be NULL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>len</code></td>
<td>the length of the buffer</td>
</tr>
</tbody>
</table>

Returns:

the CWD encoded as UTF-8

31.11.3.6  `fl_getenv()`

`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.

31.11.3.7  `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 |

Generated by Doxygen
31.11 Unicode and UTF-8 functions

Returns

1 if the path was created, 0 if creating the path failed at some point

31.11.3.8 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.

31.11.3.9 fl_mkdir()

int fl_mkdir (const char ∗ f, int mode )

Cross-platform function to create a directory with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard _wmkdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode of the directory</td>
</tr>
</tbody>
</table>

Returns

the return value of _wmkdir() on Windows or mkdir() on other platforms.

31.11.3.10 fl_nonspacing()

unsigned int fl_nonspacing (unsigned int ucs )

Returns true if the Unicode character ucs is non-spacing.
Non-spacing characters in Unicode are typically combining marks like tilde (~), diaeresis (¨), or other marks that are added to a base character, for instance 'a' (base character) + "¨" (combining mark) = 'ä' (German Umlaut).

• http://unicode.org/glossary/#base_character
• http://unicode.org/glossary/#nonspacing_mark
• http://unicode.org/glossary/#combining_character

31.11.3.11 fl_open()

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.
**31.11.3.12 fl_open_ext()**

```c
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><strong>fname</strong></th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>binary</strong></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><strong>oflags,...</strong></td>
<td>these arguments are as in the standard open() function. Setting <strong>oflags</strong> to zero opens the file for reading.</td>
</tr>
</tbody>
</table>

**Returns**

a file descriptor upon successful completion, or -1 in case of error.

**See also**

`fl_fopen()`, `fl_open_ext()`.

---

**31.11.3.13 fl_putenv()**

```c
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.
31.11 Unicode and UTF-8 functions

Parameters

| in | var | the UTF-8 encoded environment variable 'name=value' |

Returns

0 on success, non-zero in case of error.

31.11.3.14 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.

31.11.3.15 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.

31.11.3.16 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 |

Generated by Doxygen
Parameters

\[ b \] the stat struct to populate

Returns

the return value of \_wstat() on Windows or \texttt{stat}() on other platforms.

31.11.3.17 \texttt{fl\_system()}

\begin{verbatim}
int fl_system (  
    const char ∗ cmd )  
\end{verbatim}

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 \texttt{system}() directly.

Parameters

\begin{itemize}
    \item \texttt{cmd} the UTF-8 encoded command string
\end{itemize}

Returns

the return value of \_wsystem() on Windows or \texttt{system}() on other platforms.

31.11.3.18 \texttt{fl\_ucs\_to\_Utf16()}

\begin{verbatim}
unsigned fl_ucs_to_Utf16 (  
    const unsigned ucs,  
    unsigned short ∗ dst,  
    const unsigned dstlen )  
\end{verbatim}

Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.  
These are used by some system calls, especially on Windows.  
\texttt{ucs} is the value to convert.  
\texttt{dst} points at an array to write, and \texttt{dstlen} is the number of locations in this array.  
At most \texttt{dstlen} words will be written, and a 0 terminating word will be added if \texttt{dstlen} is large enough.  
Thus this function will never overwrite the buffer and will attempt return a zero-terminated string if space permits.  
If \texttt{dstlen} is zero then \texttt{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 \texttt{would} be written to \texttt{dst} if it is large enough, not counting any terminating zero.  
If the return value is greater than \texttt{dstlen} it indicates truncation, you should then allocate a new array of size \texttt{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 \texttt{dstlen} to 2 will ensure that any valid Unicode value can be converted,  
and setting \texttt{dstlen} to 3 or more will allow a NULL terminated sequence to be returned.

31.11.3.19 \texttt{fl\_unlink()}

\begin{verbatim}
int fl_unlink (  
    const char ∗ fname )  
\end{verbatim}

Cross-platform function to \texttt{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.
31.11 Unicode and UTF-8 functions

Parameters

- **name** the filename to unlink

Returns

the return value of _wunlink() on Windows or unlink() on other platforms.

31.11.3.20  **fl_utf8back()**

```c
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.

31.11.3.21  **fl_utf8bytes()**

```c
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** UCS4 encoded character

Returns

number of bytes required

31.11.3.22  **fl_utf8decode()**

```c
unsigned fl_utf8decode {
    const char * p,
    const char * end,
    int * len )
```

Decode a single UTF-8 encoded character starting at \( p \).
The resulting Unicode value (in the range 0-0x10ffff) is returned, and \( len \) is set to the number of bytes in the UTF-8 encoding (adding \( len \) to \( p \) will point at the next character).
If \( p \) points at an illegal UTF-8 encoding, including one that would go past \( end \), or where a code uses more bytes than necessary, then \( *(unsigned char *)p \) is translated as though it is in the Microsoft CP1252 character set and \( len \) is set to 1. Treating errors this way allows this to decode almost any ISO-8859-1 or CP1252 text that has been mistakenly placed where UTF-8 is expected, and has proven very useful.
If you want errors to be converted to error characters (as the standards recommend), adding a test to see if the length is unexpectedly 1 will work:

```c
if (*p & 0x80) {
    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;
}
```

Generated by Doxygen
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.

31.11.3.23 fl_utf8encode()

```c
int f1_utf8encode (unsigned ucs, char * buf )
```

Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
Up to 4 bytes may be written. If you know that ucs is less than 0x10000 then at most 3 bytes will be written. If you wish to speed this up, remember that anything less than 0x80 is written as a single byte.
If ucs is greater than 0x10ffff this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER).
RFC 3629 also says many other values for ucs are illegal (in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff). However I encode these as though they are legal, so that utf8encode/fl_utf8decode will be the identity for all codes between 0 and 0x10ffff.

31.11.3.24 fl_utf8from_mb()

```c
unsigned f1_utf8from_mb (char * dst, unsigned dstlen, const char * src, unsigned srclen )
```

Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.
Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.
On Unix or on Windows when a UTF-8 locale is in effect, this does not change the data. You may also want to check if fl_utf8test() returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.

31.11.3.25 fl_utf8froma()

```c
unsigned f1_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.

31.11.3.26 fl_utf8fromwc()

```c
unsigned f1_utf8fromwc (char * dst, unsigned dstlen, const wchar_t * src, unsigned srclen )
```

Generated by Doxygen
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.

31.11.3.27 fl_utf8fwd()

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.

31.11.3.28 fl_utf8len()

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

31.11.3.29 fl_utf8len1()

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

31.11.3.30 fl_utf8locale()

int fl_utf8locale (void)

Return true if the "locale" seems to indicate that UTF-8 encoding is used.
If true the fl_utf8to_mb and fl_utf8from_mb don't do anything useful. It is highly recommended that you change your system so this does return true. On Windows this is done by setting the "codepage" to CP_UTF8. On Unix this is done by setting $LC_CTYPE to a string containing the letters "utf" or "UTF" in it, or by deleting all $LC_* and $LANG environment variables. In the future it is likely that all non-Asian Unix systems will return true, due to the compatibility of UTF-8 with ISO-8859-1.

31.11.3.31 fl_utf8test()

```c
int fl_utf8test (const char * src, unsigned srclen )
```

Examines the first srclen bytes in src and returns a verdict on whether it is UTF-8 or not.

- Returns 0 if there is any illegal UTF-8 sequences, using the same rules as fl_utf8decode(). Note that some UCS values considered illegal by RFC 3629, such as 0xffff, are considered legal by this.
- Returns 1 if there are only single-byte characters (ie no bytes have the high bit set). This is legal UTF-8, but also indicates plain ASCII. It also returns 1 if srclen is zero.
- Returns 2 if there are only characters less than 0x800.
- Returns 3 if there are only characters less than 0x10000.
- Returns 4 if there are characters in the 0x10000 to 0x10ffff range.

Because there are many illegal sequences in UTF-8, it is almost impossible for a string in another encoding to be confused with UTF-8. This is very useful for transitioning Unix to UTF-8 filenames, you can simply test each filename with this to decide if it is UTF-8 or in the locale encoding. My hope is that if this is done we will be able to cleanly transition to a locale-less encoding.

31.11.3.32 fl_utf8to_mb()

```c
unsigned fl_utf8to_mb (const char * src, unsigned srclen, char * dst, unsigned dstlen )
```

Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

Unfortunately due to stupid design you will have to do this as needed for filenames. This is a bug on both Unix and Windows.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed. If fl_utf8locale() returns true then this does not change the data.

31.11.3.33 fl_utf8toa()

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

Convert a UTF-8 sequence into an array of 1-byte characters.

If the UTF-8 decodes to a character greater than 0xff then it is replaced with "."

Errors in the UTF-8 sequence are converted as individual bytes, same as fl_utf8decode() does. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly (and possibly CP1252 on Windows).

src points at the UTF-8 sequence, and srclen is the number of bytes to convert.

Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of
size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.

31.11.3.34  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).

31.11.3.35  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.

31.11.3.36  fl_utf_strcasecmp()

int fl_utf_strcasecmp (  
    const char * s1,  
    const char * s2 )  

UTF-8 aware strcasecmp - converts to Unicode and tests.

Returns

dependent on the result of comparison

Return values

0  if the strings are equal
Return values

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if s1 is greater than s2</td>
</tr>
<tr>
<td>-1</td>
<td>if s1 is less than s2</td>
</tr>
</tbody>
</table>

31.11.37 fl_utf_strncasecmp()

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

UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, s2</td>
<td>the UTF-8 strings to compare</td>
</tr>
<tr>
<td>n</td>
<td>the maximum number of UTF-8 characters to compare</td>
</tr>
</tbody>
</table>

Returns

result of comparison

Return values

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the strings are equal</td>
</tr>
<tr>
<td>&gt;0</td>
<td>if s1 is greater than s2</td>
</tr>
<tr>
<td>&lt;0</td>
<td>if s1 is less than s2</td>
</tr>
</tbody>
</table>

31.11.38 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 * len [for 16-bit Unicode]

31.11.39 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 * len [for 16-bit Unicode]

31.11.40 fl_wcwidth()

```c
int fl_wcwidth (const char * src )
```

extended wrapper around fl_wcwidth_(unsigned int ucs) function.
31.12 Mac OS X-specific symbols

Mac OS X-specific symbols declared in `<FL/platform.H>`

Classes

- class Fl_Mac_App_Menu

Functions

- **FL_Window * fl_mac_find (FLWindow *)**
  
  Returns the FL_Window corresponding to the given macOS-specific window reference.

- CGContextRef fl_mac_gc ()
  
  Returns the macOS-specific graphics context for the current window.

- void fl_mac_set_about (FL_Callback *cb, void *user_data, int shortcut=0)
  
  Attaches a callback to the “About myprog” item of the system application menu.

- FLWindow * fl_mac_xid (const FL_Window *win)
  
  Returns the macOS-specific window reference corresponding to the given FL_Window object.

- void fl_open_callback (void(*cb)(const char *))
  
  Register a function called for each file dropped onto an application icon.

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.

31.11.3.41 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](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.
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).

31.12.1 Detailed Description

Mac OS X-specific symbols declared in `<FL/platform.H>

See also

The Apple OS X Interface

31.12.2 Function Documentation

31.12.2.1 fl_mac_set_about()

```c
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)`.

31.12.2.2 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.

31.12.3 Variable Documentation

31.12.3.1 fl_mac_os_version

```c
int fl_mac_os_version [extern]
```

The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13).

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.

31.13 Common Dialog Classes and Functions

Common dialog functions for file selection, message output, and more.

Files

- file fl_ask.cxx
  
  Utility functions for common dialogs.
31.13 Common Dialog 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_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style fmt message.

- int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

- int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

- char * fl_dir_chooser (const char *message, const char *fname, int relative)
  Shows a file chooser dialog and gets a directory.

- char * fl_file_chooser (const char *message, const char *pat, const char *fname, int relative)
  Shows a file chooser dialog and gets a filename.

- void fl_file_chooser_callback (void (cb)(const char *))(const char *)
  Set the file chooser callback.

- void fl_file_chooser_ok_label (const char *l)
  Set the "OK" button label.

- const char * fl_input (const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message with variable arguments.

- Fl_String fl_input_str (int maxchar, const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the fmt message with variable arguments.

- void fl_message (const char *fmt,...)
  Shows an information message dialog box.

- int fl_message_hotspot ()
  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

- void fl_message_hotspot (int enable)
  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

- Fl_Widget * fl_message_icon ()
  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- void fl_message_icon_label (const char *str)
  Sets the icon label of the dialog window used in many common dialogs.

- void fl_message_position (const int x, const int y, const int center)
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- void fl_message_position (Fl_Widget *widget)
Sets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `int fl_message_position (int *x, int *y)`
  Gets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

- `void fl_message_title (const char *title)`
  Sets the title of the dialog window used in many common dialogs.

- `void fl_message_title_default (const char *title)`
  Sets the default title of the dialog window used in many common dialogs.

- `const char * fl_password (const char *fmt, const char *defstr,...)`
  Shows an input dialog displaying the `fmt` message with variable arguments.

- `Fl_String fl_password_str (int maxchar, const char *fmt, const char *defstr,...)`
  Shows an input dialog displaying the `fmt` message with variable arguments.

**Variables**

- static void(* Fl::error )(const char *,...) = Fl_System_Driver::error
  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

- Fl_Font fl_message_font_ = FL_HELVETICA

- Fl_Fontsize fl_message_size_ = -1

- const char * fl_no = "No"
  String pointer used in common dialogs, you can change it to another language

- const char * fl_ok = "OK"
  String pointer used in common dialogs, you can change it to another language

- const char * fl_yes = "Yes"
  String pointer used in common dialogs, you can change it to another language

- static void(* Fl::warning )(const char *,...) = Fl_System_Driver::warning
  FLTK calls Fl::warning() to output a warning message.

31.13.1 Detailed Description

Common dialog functions for file selection, message output, and more.

31.13.2 Function Documentation

31.13.2.1 fl_alert()

```c
void fl_alert (const char * fmt, ...)
```

Shows an alert message dialog box.

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

**Parameters**

| in | fmt | can be used as an snprintf-like format and variables for the message text |
### fl_ask()

```c
int fl_ask (const char ∗fmt,
            ... )
```

Shows a dialog displaying the `fmt` message, this dialog features 2 yes/no buttons.

#### Parameters

- **fmt**: can be used as an sprintf-like format and variables for the message text

#### Return values

- **0**: if the no button is selected
- **1**: if yes is selected

**Deprecated**: `fl_ask()` is deprecated since it uses "Yes" and "No" for the buttons which does not conform to the current FLTK Human Interface Guidelines. Use `fl_choice()` with the appropriate verbs instead.

### fl_beep()

```c
void fl_beep (int type )
```

Emits a system beep message.

#### Parameters

- **type**: The beep type from the `Fl_Beep` enumeration.

### 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 31.5 fl_choice() button ordering](image)

**Figure 31.5 fl_choice() button ordering**

Generated by Doxygen
Three choices with printf() style formatting:

```c
int num_msgs = GetNumberOfMessages();
switch ( fl_choice("What to do with %d messages?", "Send", "Save", "Delete", num_msgs) ) {
  case 0: .. // Send
  case 1: .. // Save (default)
  case 2: .. // Delete
  ..
}
```

Three choice example:

```c
switch ( fl_choice("How many bedrooms?", "Zero", "One", "Two") ) {
  case 0: .. // "Zero"
  case 1: .. // "One" (default)
  case 2: .. // "Two"
}
```

Two choice example:

```c
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
  case 0: .. // Yes
  case 1: .. // No (default)
}
```

One choice example:

```c
fl_choice("All hope is lost.", "OK", 0, 0);  // "OK" default
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>can be used as a sprintf-like format and variables for the message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>b0</td>
<td>text label for right button 0</td>
</tr>
<tr>
<td>in</td>
<td>b1</td>
<td>text label for middle button 1 (can be 0)</td>
</tr>
<tr>
<td>in</td>
<td>b2</td>
<td>text label for left button 2 (can be 0)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th></th>
<th>if the button with b0 text is pushed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></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>
31.13 Common Dialog Classes and Functions

31.13.2.5 fl_choice_n()

```c
int fl_choice_n ( 
    const char * fmt, 
    const char * b0, 
    const char * b1, 
    const char * b2, 
    ... )
```

Shows a dialog displaying the printf style `fmt` message.
This function is like `fl_choice()` but returns `-1` if the dialog window was closed by pressing the `Escape` key or the window close button rather than pushing one of the dialog buttons.

See also

`fl_choice()`

Parameters

- `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

- `-3` reserved, FLTK 1.3 only: another dialog is still open (not possible in 1.4)
- `-2` if the dialog was closed by pushing the window close button
- `-1` if the dialog was closed by hitting `Escape`
- `0` if the button with `b0` text is pushed
- `1` if the button with `b1` text is pushed
- `2` if the button with `b2` text is pushed

31.13.2.6 fl_color_chooser() [1/2]

```c
int fl_color_chooser ( 
    const char * name, 
    double & r, 
    double & g, 
    double & b, 
    int cmode ) [related]
```

Pops up a window to let the user pick an arbitrary RGB color.
Note

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

![Figure 31.10 fl_color_chooser](image)

Figure 31.10 fl_color_chooser

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>name</strong></td>
<td>Title label for the window</td>
</tr>
<tr>
<td><strong>in, out</strong></td>
<td><strong>r,g,b</strong></td>
<td>Color components in the range 0.0 to 1.0.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>cmode</strong></td>
<td>Optional mode for color chooser. See <code>mode(int)</code>. Default: -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if user confirms the selection</td>
</tr>
<tr>
<td>0</td>
<td>if user cancels the dialog</td>
</tr>
</tbody>
</table>

### 31.13.2.7 fl_color_chooser() [2/2]

```c
int fl_color_chooser ( 
    const char * name,
    uchar & r,
    uchar & g,
    uchar & b,
    int cmode ) [related]
```

Pops up a window to let the user pick an arbitrary RGB color.
31.13 Common Dialog Classes and Functions

Note

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

![Figure 31.11 fl_color_chooser](image)

**Figure 31.11 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 to 255.</td>
</tr>
<tr>
<td>in</td>
<td>cmode</td>
<td>Optional mode for color chooser. See <code>mode(int)</code>. Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

### Return values

<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>

#### 31.13.2.8 fl_dir_chooser()

```c
char * fl_dir_chooser (const char * message, const char * fname, int relative) [related]
```

Shows a file chooser dialog and gets a directory.

Note

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

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>message</td>
<td>Title bar text</td>
</tr>
<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>

Generated by Doxygen
Returns
the directory path string chosen by the user or NULL if user cancels

31.13.2.9 fl_file_chooser()

```
char * fl_file_chooser (  
    const char * message,  
    const char * pat,  
    const char * fname,  
    int relative ) [related]
```

Shows a file chooser dialog and gets a filename.

Note
`#include <FL/Fl_File_Chooser.H>`

![Figure 31.12 Fl_File_Chooser](image)

Figure 31.12 Fl_File_Chooser

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>message</code></td>
<td>text in title bar</td>
</tr>
<tr>
<td><code>pat</code></td>
<td>filename pattern filter</td>
</tr>
<tr>
<td><code>fname</code></td>
<td>initial/default filename selection</td>
</tr>
<tr>
<td><code>relative</code></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
31.13.2.10 fl_file_chooser_callback()

void fl_file_chooser_callback (  
  void(*) (const char *) cb) [related]
Set the file chooser callback.

Note
  
  #include <FL/Fl_File_Chooser.H>

31.13.2.11 fl_file_chooser_ok_label()

void fl_file_chooser_ok_label (  
  const char * l) [related]
Set the "OK" button label.

Note
  
  #include <FL/Fl_File_Chooser.H>

31.13.2.12 fl_input()

const char* fl_input (  
  const char * fmt,  
  const char * defstr,  
  ...)
Shows an input dialog displaying the fmt message with variable arguments.
This version of fl_input() is deprecated. The return value points to an internal allocated string that may be changed
later. You must copy the string immediately after return from this method - at least before the next execution of the
event loop.

Deprecated Please use fl_input_str(int maxchar, const char *fmt, const char *defstr, ...) instead.

#include <FL/fl_ask.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>can be used as an sprintf-like format and variables for the message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>defstr</td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

Returns

the user string input if OK was pushed

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if Cancel was pushed or the window was closed by the user</td>
</tr>
</tbody>
</table>

31.13.2.13 fl_input_str()

Fl_String fl_input_str (  
  int maxchar,  
  const char * fmt,  
  const char * defstr,  
  ...)

Generated by Doxygen
Shows an input dialog displaying the fmt message with variable arguments. Like `fl_input()`, but this method has an additional (first) argument `maxchar` that limits the number of characters that can be input. Since the string is encoded in UTF-8 it is possible that the number of bytes in the string is larger than `maxchar`.

Other than the deprecated `fl_input()` method w/o the `maxchar` argument, this one returns the string in an `Fl_String` object that must be released after use. This can be a local/automatic variable.

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

Example:
( Fl_String str = fl_input_str(0, "Enter text: ", "") ;
  printf("Text is: '%s' \n", str.value() ? str.value() : "<cancelled>");
  // (str goes out of scope)
```

If the user hits Escape or closes the window `str.value()` returns NULL.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>maxchar</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>fmt</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>defstr</code></td>
</tr>
</tbody>
</table>

### Returns

the user string input if OK was pushed or NULL in `Fl_String::value()`

### Return values

`Fl_String::value()` == NULL if Cancel was pushed or the window was closed by the user

Since

1.4.0

### 31.13.2.14 fl_message()

```c
void fl_message ( const char * fmt, ... )
```

Shows an information message dialog box.

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

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>fmt</code></td>
</tr>
</tbody>
</table>

### 31.13.2.15 fl_message_hotspot() [1/2]

```c
int fl_message_hotspot { void }
```

Gets whether or not to move the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()` to follow the mouse pointer.

This is a permanent setting. It remains active and affects the window position unless overridden by an explicit positioning request by means of one of the `fl_message_position()` variants.

```c
#include <FL/fl_ask.H>
```
Returns

0 if disabled, non-zero otherwise

See also

void fl_message_hotspot(int)
int fl_message_position(int +x, int +y)
void fl_message_position(Fl_Widget +)
fl_message_position()

31.13.2.16 fl_message_hotspot() [2/2]

void fl_message_hotspot (  
    int enable  
)  
Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer. The default is enabled, so that the default button is the hotspot and appears at the mouse position.

#include <FL/fl_ask.H>

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>enable</td>
<td>non-zero enables hotspot behavior, 0 disables hotspot</td>
</tr>
</tbody>
</table>

31.13.2.17 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(). The return value cannot be Null. The object pointed to is an Fl_Box widget. The returned pointer (Fl_Widget *) can be safely cast to an Fl_Box pointer.

Note

You can set some attributes of this default icon box. These attributes are sticky, i.e. they will be used in all subsequent common dialogs unless overridden by specific "one shot" variables. Setting any attribute except those mentioned below causes undefined behavior.

Supported icon attributes:

- box()
- labelfont()
- labelsize()
- color()
- labelcolor()
- image()
- align()

The icon size can not be changed. If you set an image() you should scale it to the available size, i.e. w() and h() of the icon box.

#include <FL/fl_ask.H>
31.13.2.18 fl_message_icon_label()

void fl_message_icon_label ( const char * str )

Sets the icon label of the dialog window used in many common dialogs.
This icon label will be used in the next call of one of the common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The label str is stored internally as a reference, it must be in scope until the dialog function (e.g. fl_choice) is called.
It applies only to the next call of one of the common dialogs and will be reset after that call so the next dialog will use its default label unless set again.

Note
This label string must be short, usually only one character so it fits in the icon box. You can use any valid UTF-8 character, e.g. the Euro sign ("€") which is three bytes in UTF-8 encoding.

# include <FL/fl_ask.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>icon label</th>
</tr>
</thead>
</table>

31.13.2.19 fl_message_position() [1/3]

void fl_message_position ( const int x, const int y, const int center )

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by fl_message_hotspot(int).
The preferred position set by any of the fl_message_position() variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.
If the optional argument center is non-zero (true) the message box will be centered at the given coordinates rather than using the X/Y position as the window position (top left corner).

# include <FL/fl_ask.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>Preferred X position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>y</td>
<td>Preferred Y position</td>
</tr>
<tr>
<td>in</td>
<td>center</td>
<td>1 = centered, 0 = absolute</td>
</tr>
</tbody>
</table>

See also
int fl_message_position(int *x, int *y)

31.13.2.20 fl_message_position() [2/3]

void fl_message_position ( Fl_Widget * widget )

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The message box will be centered over the given widget or window extensions.
Everything else is like `fl_message_position(int, int, int)` with argument 'center' set to 1.

```c
#include <FL/fl_ask.H>
```
Parameters

```
in | widget | Widget or window to position the message box over.
```

See also

```
int fl_message_position(int x, int y, int center)
```

### 31.13.2.21 fl_message_position() [3/3]

```c
int fl_message_position (  
  int * x,  
  int * y)
```

Gets the preferred position for the message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.

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

The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by `fl_message_hotspot(int)`.

The preferred position set by any of the `fl_message_position()` variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.

Parameters

```
out | x | Preferred X position, returns -1 if not set
out | y | Preferred Y position, returns -1 if not set
```

Returns

whether position is currently set or not

Return values

```
0 | position is not set (hotspot may be enabled or not)
1 | position is set (window position)
2 | position is set (message box centered)
```

See also

```
fl_message_hotspot()  
fl_message_hotspot(int)
```

```
fl_message_position(int, int)  
fl_message_position(const int x, const int y, const int center)  
fl_message_position(Fl_Widget *)
```

### 31.13.2.22 fl_message_title()

```c
void fl_message_title (  
  const char * title )
```

Sets the title of the dialog window used in many common dialogs.

This window title will be used in the next call of one of the common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.
The `title` string is copied internally, so that you can use a local variable or free the string immediately after this call. It applies only to the next call of one of the common dialogs and will be reset to an empty title (the default for all dialogs) after that call.

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

### Parameters

**`title`**
- `in` window label, string copied internally

#### 31.13.2.23 `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.

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

### Parameters

**`title`**
- `in` default window label, string copied internally

#### 31.13.2.24 `fl_password()`

```c
const char* fl_password ( const char * fmt, const char * defstr, ... )
```

Shows an input dialog displaying the `fmt` message with variable arguments. Like `fl_input()` except the input text is not shown, '*' or similar replacement characters are displayed instead.

**Deprecated** Please use `fl_password_str(int maxchar, const char *fmt, const char *defstr, ...)` instead.

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

### Parameters

**`fmt`**
- `in` can be used as an sprintf-like format and variables for the message text

**`defstr`**
- `in` defines the default returned string if no text is entered

### Returns

- the user string input if OK was pushed

### Return values

- `NULL` if Cancel was pushed or the window was closed by the user
31.13.25  fl_password_str()

fl_password_str (int maxchar, const char * fmt, const char * defstr, ...) Shows an input dialog displaying the fmt message with variable arguments. Like fl_input_str() except the input text is not shown, ’*’ or similar replacement characters are displayed instead. Other than the deprecated fl_password() method w/o the maxchar argument, this one returns the string in an Fl_String object that must be released after use. This can be a local/automatic variable.
For an example see fl_input_str()
#include <FL/fl_ask.h>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>maxchar</th>
<th>input size limit in characters (not bytes); use 0 for no limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</td>
<td>can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td>in</td>
<td>defstr</td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

Returns

the user string input if OK was pushed or NULL in Fl_String::value()

Return values

| Fl_String::value() | == NULL if Cancel was pushed or the window was closed by the user |

Since

1.4.0

31.13.3  Variable Documentation

31.13.3.1  error

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

#include <FL/Fl.H>

31.13.3.2  fatal

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.
31.14 File names and URI utility functions

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>
```

31.13.3.3 warning

void(* Fl::warning)(const char *format, ...) = Fl_System_Driver::warning [static]

FTK 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>
```

### 31.14 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 ***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)`

Generated by Doxygen
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.
- FLEXPORT 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).

### 31.14.1 Detailed Description

File names and URI functions defined in `<FL/filename.H>`

### 31.14.2 Typedef Documentation

#### 31.14.2.1 Fl_File_Sort_F

typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

File sorting function.

See also

fl_filename_list()
31.14 File names and URI utility functions

Returns

0 if no change, non zero otherwise

31.14.3.3  fl_filename_expand()

FL_EXPORT int fl_filename_expand (  
    char * to,  
    int tolen,  
    const char * from )

Expands a filename containing shell variables and tilde (~).
Currently handles these variants:

"~username"  // if 'username' does not exist, result will be unchanged
"~/file"
"$VARNAME"  // does NOT handle ${VARNAME}

Examples:
#include <FL/filename.H>
[..]
putenv("TMPDIR=/var/tmp");
fl_filename_expand(out, sizeof(out), "~fred/.cshrc");  // out="/usr/fred/.cshrc"
fl_filename_expand(out, sizeof(out), "~/file");  // out="/usr/<yourname>/cshrc"
fl_filename_expand(out, sizeof(out), "$TMPDIR/foo.txt");  // out="/var/tmp/foo.txt"

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting expanded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the expanded filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>filename containing shell variables</td>
</tr>
</tbody>
</table>

Returns

0 if no change, non zero otherwise

31.14.3.4  fl_filename_ext()

FL_EXPORT const char * fl_filename_ext (  
    const char * buf )

Gets the extension of a filename.
#include <FL/filename.H>
[..]
const char *out;
out = fl_filename_ext("/some/path/foo.txt");  // result: ".txt"
out = fl_filename_ext("/some/path/foo");  // result: NULL

Parameters

| in    | buf     | the filename to be parsed |

Returns

a pointer to the extension (including '.') if any or NULL otherwise

31.14.3.5  fl_filename_free_list()

FL_EXPORT void fl_filename_free_list (  
    struct dirent *** list,  
    int n )

Free the list of filenames that is generated by fl_filename_list().

Generated by Doxygen
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><code>in, out</code></td>
<td><code>list</code></td>
<td>table containing the resulting directory listing</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code></td>
<td>number of entries in the list</td>
</tr>
</tbody>
</table>

### 31.14.3.6 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><code>in</code></td>
<td><code>n</code></td>
<td>the filename to parse</td>
</tr>
</tbody>
</table>

**Returns**

non zero if file exists and is a directory, zero otherwise

### 31.14.3.7 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>d</code></td>
<td>the name of the directory to list. It does not matter if it has a trailing slash.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>list</code></td>
<td>table containing the resulting directory listing</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sort</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<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>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
<tr>
<td></td>
<td>fl_casenumeric_sort: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared equally case is not significant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fl_numeric_sort: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared according to their ASCII ordering - uppercase before lowercase.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

the number of entries if no error, a negative value otherwise.

**Todo** should support returning OS error messages

### 31.14.3.8 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 are matched "exactly" on a case-insensitive basis.

Notes:

- `s` and `p` are matched on a char/byte basis, not as UCS codepoints or UTF-8 sequences.
- `[set]` ranges must run from low to high, i.e. `[a-z]` and not `[z-a]`
- `[set]` comparison is case-sensitive, i.e. `[a-z]` won't match "A".
- `\x` only applies to the fl_filename_match special characters `*` `?` `{` `[^`
- `\x` needs a double `\` or the compiler will complain about non-standard escape sequences.

**Include:**

```c
#include <FL/filename.H>
```
31.14.3.9  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"    
(out = fl_filename_name("/usr/");  // out="usr"    
(out = fl_filename_name("/usr/");  // out="usr"    
(out = fl_filename_name("."");  // out=".."    
(out = fl_filename_name("..");  // out=".."

Returns
a pointer to the char after the last slash, or to filename if there is none.

31.14.3.10  fl_filename_relative()

FL_EXPORT int fl_filename_relative (   
    char * to,
    int tolen,
    const char * from )

Makes a filename relative to the current working directory.
#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/somedir/foo.txt");  // out="foo.txt", return=1
fl_filename_relative(out, sizeof(out), "/foo.txt");  // out="foo.txt", return=0 (no change)
fl_filename_relative(out, sizeof(out), "/foo.txt");  // out="foo.txt", return=0 (no change)
fl_filename_relative(out, sizeof(out), "/../foo.txt");  // out="../foo.txt", return=0 (no change)
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>
31.14.3.11 fl_filename_setext()

FL_EXPORT char * fl_filename_setext ( char * buf, int buflen, const char * ext )

Replaces the extension in buf of max. size buflen with the extension in ext.
If there’s no ‘.’ in buf, ext is appended.
If ext is NULL, behaves as if it were an empty string ("").

Example

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

char buf[FL_PATH_MAX] = "/path/myfile.cxx";
fl_filename_setext(buf, sizeof(buf), ".txt"); // buf[] becomes "/path/myfile.txt"
```

Returns

buf itself for calling convenience.

31.14.3.12 fl_open_uri()

int fl_open_uri ( const char * uri, char * msg, int msglen )

Opens the specified Uniform Resource Identifier (URI).

Uses an operating-system dependent program or interface. For URIs using the "ftp", "http", or "https" schemes, the system default web browser is used to open the URI, while "mailto" and "news" URIs are typically opened using the system default mail reader and "file" URIs are opened using the file system navigator.

On success, the (optional) msg buffer is filled with the command that was run to open the URI; on Windows, this will always be "open uri".
On failure, the msg buffer is filled with an English error message.

Note

Platform Specific Issues: Windows
With "file:" based URIs on Windows, you may encounter issues with anchors being ignored. Example: "file://c:/some/index.html#anchor" may open in the browser without the "#anchor" suffix. The behavior seems to vary across different Windows versions. Workaround: open a link to a separate html file that redirects to the desired "file:" URI.

Example

```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>uri</td>
<td>The URI to open</td>
</tr>
<tr>
<td>msg</td>
<td>Optional buffer which contains the command or error message</td>
</tr>
<tr>
<td>msglen</td>
<td>Length of optional buffer</td>
</tr>
</tbody>
</table>
Returns
1 on success, 0 on failure

31.15 Fl_string

Functions

• FL_EXPORT char * fl_strdup (const char *s)
  
  Cross platform interface to POSIX function strdup().

31.15.1 Detailed Description

31.15.2 Function Documentation

31.15.2.1 fl_strdup()

FL_EXPORT char* fl_strdup {
    const char * s }

Cross platform interface to POSIX function strdup(). The fl_strdup() function returns a pointer to a new string which is a duplicate of the string 's'. Memory for the new string is obtained with malloc(3), and can be freed with free(3).
Implementation:

• POSIX: strdup()

• WinAPI: _strdup()
Chapter 32

Class Documentation

32.1 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

32.2 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_LAST
} Enumerator for global FLTK options.

Static Public Member Functions

- `static int abi_check (const int val=FL_ABI_VERSION)`
  `Returns whether the runtime library ABI version is correct.`
- `static int abi_version ()`  
  `Returns the compiled-in value of the FL_ABI_VERSION constant.`
- `static int add_awake_handler_ (Fl_Awake_Handler, void *)`  
  `Adds an awake handler for use in awake().`
- `static void add_check (Fl_Timeout_Handler, void **)`  
  `FLTK will call this callback just before it flushes the display and waits for events.`
- `static void add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data=0)`  
  `FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.`
- `static void add_fd (int fd, Fl_FD_Handler cb, void **)`  
  `Adds file descriptor fd to listen to.`
- `static void add_fd (int fd, int when, Fl_FD_Handler cb, void **)`  
  `Adds file descriptor fd to listen to.`
• static void add_handler (Fl_Event_Handler h)
  Install a function to parse unrecognized events.

• static void add_idle (Fl_Idle_Handler cb, void *data=0)
  Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).

• static void add_system_handler (Fl_System_Handler h, void *data)
  Install a function to intercept system events.

• static void add_timeout (double t, Fl_Timeout_Handler, void *data=0)
  Adds a one-shot timeout callback.

• static int api_version ()
  Returns the compiled-in value of the FL_API_VERSION constant.

• static int arg (int argc, char **argv, int &i)
  Parse a single switch from argv, starting at word i.

• static void args (int argc, char **argv)
  Parse all command line switches matching standard FLTK options only.

• static int args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)
  Parse command line switches using the cb argument handler.

• static int awake (Fl_Awake_Handler cb, void *message=0)
  See void awake(void *message=0).

• static void awake (void *message=0)
  Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

• static void background (uchar, uchar, uchar)
  Changes fl_color(FL_BACKGROUND_COLOR) to the given color, and changes the gray ramp from 32 to 56 to black to white.

• static void background2 (uchar, uchar, uchar)
  Changes the alternative background color.

• static Fl_Widget * belowmouse ()
  Gets the widget that is below the mouse.

• static void belowmouse (Fl_Widget *)
  Sets the widget that is below the mouse.

• static int box_border_radius_max ()
  Get the maximum border radius of all "rounded" boxtypes in pixels.

• static void box_border_radius_max (int R)
  Set the maximum border radius of all "rounded" boxtypes in pixels.

• static Fl_Color box_color (Fl_Color)
  Gets the drawing color to be used for the background of a box.

• static int box_dh (Fl_Boxtype)
  Returns the height offset for the given boxtype.

• static int box Dw (Fl_Boxtype)
  Returns the width offset for the given boxtype.

• static int box_dx (Fl_Boxtype)
  Returns the X offset for the given boxtype.

• static int box_dy (Fl_Boxtype)
  Returns the Y offset for the given boxtype.

• static int box_shadow_width ()
  Get the box shadow width of all "shadow" boxtypes in pixels.

• static void box_shadow_width (int W)
  Set the box shadow width of all "shadow" boxtypes in pixels.

• static bool cairo_autolink_context ()

Generated by Doxygen
32.2 Fl Class Reference

**Gets the current autolink mode for cairo support.**

- static void **cairo_autolink_context** (bool alink)
  
  when FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current cairo context.

- static cairo_t **cairo_cc** ()
  
  Gets the current cairo context linked with a fltk window.

- static void **cairo_cc** (cairo_t *c, bool own=false)
  
  Sets the current cairo context to c.

- static cairo_t **cairo_make_current** (Fl_Window *w)
  
  Provides a corresponding cairo context for window w.

- static int **check** ()
  
  Same as Fl::wait(0).

- static void **clear_widget_pointer** (Fl_Widget const *w)
  
  Clears a widget pointer in the watch list.

- static int **clipboard_contains** (const char *type)
  
  Returns non 0 if the clipboard contains data matching type.

- static int **compose** (int &del)
  
  Any text editing widget should call this for each FL_KEYBOARD event.

- static void **compose_reset** ()
  
  If the user moves the cursor, be sure to call Fl::compose_reset().

- static void **copy** (const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text)
  
  Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).

- static int **damage** ()
  
  If true then flush() will do something.

- static void **damage** (int d)
  
  If true then flush() will do something.

- static void **default_atclose** (Fl_Window *, void *)
  
  Default callback for window widgets.

- static void **delete_widget** (Fl_Widget *w)
  
  Schedules a widget for deletion at the next call to the event loop.

- static void **disable_im** ()
  
  Disables the system input methods facilities.

- static void **display** (const char *)
  
  Sets the X display to use for all windows.

- static int **dnd** ()
  
  Initiate a Drag And Drop operation.

- static int **dnd_text_ops** ()
  
  Gets whether drag and drop text operations are supported.

- static void **dnd_text_ops** (int v)
  
  Sets whether drag and drop text operations are supported.

- static void **do_widget_deletion** ()
  
  Deletes widgets previously scheduled for deletion.

- static int **draw_box_active** ()
  
  Determines if the currently drawn box is active or inactive.

- static int **draw_GL_text_with_textures** ()
  
  Returns whether whether OpenGL uses textures to draw all text.

- static void **draw_GL_text_with_textures** (int val)
  
  Sets whether OpenGL uses textures to draw all text.

- static void **enable_im** ()
  
  Enables the system input methods facilities.
• static int event()
  Returns the last event that was processed.
• static int event_alt()
  Returns non-zero if the Alt key is pressed.
• static int event_button()
  Gets which particular mouse button caused the current event.
• static int event_button1()
  Returns non-zero if mouse button 1 is currently held down.
• static int event_button2()
  Returns non-zero if button 2 is currently held down.
• static int event_button3()
  Returns non-zero if button 3 is currently held down.
• static int event_buttons()
  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
• static int event_clicks()
  Returns non-zero if we had a double click event.
• static void event_clicks(int i)
  Manually sets the number returned by Fl::event_clicks().
• static void∗ event_clipboard()
  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
• static const char∗ event_clipboard_type()
  Returns the type of the pasted data during an FL_PASTE event.
• static int event_command()
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
• static int event_ctrl()
  Returns non-zero if the Control key is pressed.
• static Fl_Event_Dispatch event_dispatch()
  Return the current event dispatch function.
• static void event_dispatch(Fl_Event_Dispatch d)
  Set a new event dispatch function.
• static int event_dx()
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
• static int event_dy()
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
• static int event_is_click()
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH
  or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
• static void event_is_click(int i)
  Clears the value returned by Fl::event_is_click().
• static int event_key()
  Gets which key on the keyboard was last pushed.
• static int event_key(int key)
  Returns true if the given key was held down (or pressed) during the last event.
• static int event_length()
  Returns the length of the text in Fl::event_text().
• static int event_original_key()
  Returns the keycode of the last key event, regardless of the NumLock state.
• static int event_shift ()
  Returns non-zero if the Shift key is pressed.
• static int event_state ()
  Returns the keyboard and mouse button states of the last event.
• static int event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.
• static const char * event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
• static int event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
• static int event_x_root ()
  Returns the mouse position on the screen of the event.
• static int event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
• static int event_y_root ()
  Returns the mouse position on the screen of the event.
• static Fl_Window * first_window ()
  Returns the first top-level window in the list of shown() windows.
• static void first_window (Fl_Window *)
  Sets the window that is returned by first_window().
• static void flush ()
  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
• static Fl_Widget * focus ()
  Gets the current Fl::focus() widget.
• static void focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.
• static void foreground (uchar, uchar, uchar)
  Changes fl_color(FL_FOREGROUND_COLOR).
• static void free_color (Fl_Color i, int overlay=0)
  Frees the specified color from the colormap, if applicable.
• static int get_awake_handler_ (Fl_Awake_Handler &, void * &)
  Gets the last stored awake handler for use in awake().
• static Fl_Box_Draw_F * get_boxtype (Fl_Boxtype)
  Gets the current box drawing function for the specified box type.
• static unsigned get_color (Fl_Color i)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)
  Returns the RGBA value(s) for the given FLTK color index.
• static const char * get_font (Fl_Font)
  Gets the string for this face.
• static const char * get_font_name (Fl_Font, int *attributes=0)
  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 ()

Generated by Doxygen
Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().

- static int gl_visual (int, int *alist=0)
  
  This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.

- static Fl_Window * grab ()
  
  Returns the window that currently receives all events.

- static void grab (Fl_Window &win)
  
  See grab(Fl_Window+*)

- static void grab (Fl_Window +)
  
  Selects the window to grab.

- static int h ()
  
  Returns the height in pixels of the main screen work area.

- static int handle (int, Fl_Window *)
  
  Handle events from the window system.

- static int handle_ (int, Fl_Window *)
  
  Handle events from the window system.

- static int has_check (Fl_Timeout_Handler, void *data=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 *data=0)
  
  Returns true if the timeout exists and has not been called yet.

- static int is_scheme (const char *name)
  
  Returns whether the current scheme is the given name.

- static void keyboard_screen_scale (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)
  
  Backward compatibility only.

- static void paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)
  
  Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.

- static int program_should_quit ()
  
  Indicates to the FLTK library whether a program termination request was received and accepted.

- static void program_should_quit (int should_i)
  
  Indicates to the FLTK library whether a program termination request was received and accepted.

- static Fl_Widget * pushed ()
  
  Generated by Doxygen
• 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)
  Removes a file descriptor handler.

• static void remove_fd (int, int when)
  Removes a file descriptor handler.

• static void remove_handler (Fl_Event_Handler h)
  Removes a previously added event handler.

• static void remove_idle (Fl_Idle_Handler cb, void **data=0)
  Removes the specified idle callback, if it is installed.

• static void remove_system_handler (Fl_System_Handler h)
  Removes a previously added system event handler.

• static void remove_timeout (Fl_Timeout_Handler, 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 int run ()
  Calls Fl::wait() repeatedly as long as any windows are displayed.

• static void run_checks ()

• static void run_idle ()

• static const char * scheme ()
  See void scheme(const char *name)

• static int scheme (const char *name)
  Sets the current widget scheme.

• static int screen_count ()
  Gets the 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)
  Sets 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 &\( H \))
  Gets the bounding box of the work area of the screen that contains the mouse pointer.
- **static void** `screen_work_area` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \), int \( mx \), int \( my \))
  Gets the bounding box of the work area of a screen that contains the specified screen position \( mx, my \).
- **static void** `screen_work_area` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \), int \( n \))
  Gets the bounding box of the work area of the given screen.
- **static void** `screen_xywh` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \))
  Gets the bounding box of a screen that contains the mouse pointer.
- **static void** `screen_xywh` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \), int \( mx \), int \( my \))
  Gets the bounding box of a screen that contains the specified screen position \( mx, my \).
- **static void** `screen_xywh` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \), int \( mx \), int \( my \), int \( mw \), int \( mh \))
  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by \( mx, my, mw, mh \).
- **static void** `screen_xywh` (int &\( X \), int &\( Y \), int &\( W \), int &\( H \), int \( n \))
  Gets the screen bounding rect for the given screen.
- **static int** `scrollbar_size` ()
  Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
- **static void** `scrollbar_size` (int \( W \))
  Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
- **static void** `selection` (Fl_Widget &owner, const char *\( \), int \( len \))
  Changes the current selection.
- **static Fl_Widget *** `selection_owner` ()
  back-compatibility only: Gets the widget owning the current selection
- **static void** `selection_owner` (Fl_Widget *)
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
- **static void** `set_abort` (Fl_Abort_Handler \( f \))
  For back compatibility, sets the void Fl::fatal handler callback.
- **static void** `set_atclose` (Fl_Atclose_Handler \( f \))
  For back compatibility, sets the Fl::atclose handler callback.
- **static void** `set_box_color` (Fl_Color)
  Sets the drawing color for the box that is currently drawn.
- **static void** `set_boxtype` (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar)
  Sets the function to call to draw a specific boxtype.
- **static void** `set_boxtype` (Fl_Boxtype, Fl_Boxtype from)
  Copies the from boxtype.
- **static void** `set_color` (Fl_Color \( i \), unsigned \( c \))
  Sets an entry in the fl_color index table.
- **static void** `set_color` (Fl_Color, uchar, uchar, uchar)
  Sets an entry in the fl_color index table.
- **static void** `set_color` (Fl_Color, uchar, uchar, uchar, uchar)
  Sets an entry in the fl_color index table.
- **static void** `set_font` (Fl_Font, const char *)
  Changes a face.
- **static void** `set_font` (Fl_Font, Fl_Font)
  Generated by Doxygen
Copies one face to another.

- **static Fl_Font set_fonts (const char *=0)**
  
  FLTK will open the display, and add every fonts on the server to the face table.

- **static void set_idle (Fl_Old_Idle_Handler cb)**
  
  Sets an idle callback.

- **static void set_labeltype (Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *)**
  
  Sets the functions to call to draw and measure a specific labeltype.

- **static void set_labeltype (Fl_Labeltype, Fl_Labeltype from)**
  
  Sets the functions to call to draw and measure a specific labeltype.

- **static int system (const char *=command)**
  
  Run a command line on the computer.

- **static Fl_System_Driver * system_driver ()**
  
  Returns a pointer to the unique Fl_System_Driver object of the platform.

- **static int test_shortcut (Fl_Shortcut)**
  
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

- **static void thread_message ()**
  
  The thread_message() method returns the last message that was sent from a child by the awake() method.

- **static void unlock ()**
  
  The unlock() method releases the lock that was set using the lock() method.

- **static int use_high_res_GL ()**
  
  Returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.

- **static void use_high_res_GL (int val)**
  
  Sets whether GL windows should be drawn at high resolution on Apple computers with retina displays.

- **static double version ()**
  
  Returns the compiled-in value of the FL_VERSION constant.

- **static int visible_focus ()**
  
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- **static void visible_focus (int v)**
  
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.

- **static int visual (int)**
  
  Selects a visual so that your graphics are drawn correctly.

- **static int w ()**
  
  Returns the width in pixels of the main screen work area.

- **static int wait ()**
  
  Waits until "something happens" and then returns.

- **static double wait (double time)**
  
  Waits a maximum of time_to_wait seconds or until "something happens".

- **static void watch_widget_pointer (Fl_Widget * &w)**
  
  Adds a widget pointer to the widget watch list.

- **static int x ()**
  
  Returns the leftmost x coordinate of the main screen work area.

- **static int y ()**
  
  Returns the topmost y coordinate of the main screen work area.
Static Public Attributes

- static void(atclose)(Fl_Window*, void *)
  Back compatibility: default window callback handler.
- static char const*clipboard_image = "image"
  Denotes image data.
- static char const*clipboard_plain_text = "text/plain"
  Denotes plain textual data.
- static void(*error)(const char*,...) = Fl_System_Driver::error
  FLTK calls Fl::error() to output a normal error message.
- static void(*fatal)(const char*,...) = Fl_System_Driver::fatal
  FLTK calls Fl::fatal() to output a fatal error message.
- static const char*const help = helpmsg+13
  Usage string displayed if Fl::args() detects an invalid argument.
- static void(*idle)()
  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
- static void(*warning)(const char*,...) = Fl_System_Driver::warning
  FLTK calls Fl::warning() to output a warning message.

32.2.1 Detailed Description

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

32.2.2 Member Enumeration Documentation

32.2.2.1 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</td>
</tr>
<tr>
<td></td>
<td>in a text widget will change focus to the next text widget. (This is</td>
</tr>
<tr>
<td></td>
<td>considered 'old' behavior) When switched off (default), the cursor will</td>
</tr>
<tr>
<td></td>
<td>stop at the end of the text. Pressing Tab or Ctrl-Tab will advance the</td>
</tr>
<tr>
<td></td>
<td>keyboard focus. See also: Fl::input::tab_nav()</td>
</tr>
<tr>
<td>OPTION_VISIBLE_FOCUS</td>
<td>If visible focus is switched on (default), FLTK will draw a dotted</td>
</tr>
<tr>
<td></td>
<td>rectangle inside the widget that will receive the next keystroke. If</td>
</tr>
<tr>
<td></td>
<td>switched off, no such indicator will be drawn and keyboard navigation</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>text from any text widget. If disabled, no dragging is possible, however</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>tooltip text will open a little tooltip window until the mouse leaves the</td>
</tr>
<tr>
<td></td>
<td>widget. If disabled, no tooltip is shown.</td>
</tr>
</tbody>
</table>
### Enumerator

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_FNFC_USES_GTK</td>
<td>When switched on (default), Fl_Native_File_Chooser runs GTK file dialogs if the GTK library is available on the platform (linux/unix only). When switched off, GTK file dialogs aren't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_PRINTER_USES_GTK</td>
<td>When switched on (default), Fl_Printer runs the GTK printer dialog if the GTK library is available on the platform (linux/unix only). When switched off, the GTK printer dialog isn't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_SHOW_SCALING</td>
<td>When switched on (default), the library shows in a transient yellow window the zoom factor value. When switched off, no such window gets displayed.</td>
</tr>
<tr>
<td>OPTION_LAST</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

### 32.2.3 Member Function Documentation

#### 32.2.3.1 abi_check()

```c
static int Fl::abi_check ( const int val = FL_ABI_VERSION ) [inline], [static]
```

Returns whether the runtime library ABI version is correct. This enables you to check the ABI version of the linked FLTK library at runtime. Returns 1 (true) if the compiled ABI version (in the header files) and the linked library ABI version (used at runtime) are the same, 0 (false) otherwise. Argument `val` can be used to query a particular library ABI version. Use for instance 10303 to query if the runtime library is compatible with FLTK ABI version 1.3.3. This is rarely useful. The default `val` argument is `FL_ABI_VERSION`, which checks the version defined at configure time (i.e. in the header files at program compilation time) against the linked library version used at runtime. This is particularly useful if you linked with a shared object library, but it also concerns static linking.

See also

- Fl::abi_version()

#### 32.2.3.2 abi_version()

```c
int Fl::abi_version ( ) [static]
```

Returns the compiled-in value of the `FL_ABI_VERSION` constant. This is useful for checking the version of a shared library.

#### 32.2.3.3 add_check()

```c
void Fl::add_check ( Fl_Timeout_Handler cb, void * argp = 0 ) [static]
```

FLTK will call this callback just before it flushes the display and waits for events. This is different than an idle callback because it is only called once, then FLTK calls the system and tells it not to return until an event happens. This can be used by code that wants to monitor the application's state, such as to keep a display up to date. The advantage of using a check callback is that it is called only when no events are pending. If events are coming in quickly, whole blocks of them will be processed before this is called once. This can save significant time and avoid the application falling behind the events.

Sample code:

```c
bool state_changed; // anything that changes the display turns this on
void callback(void* arg) { 
  if (state_changed) return;
```

Generated by Doxygen
state_changed = false;
do_expensive_calculation();
widget->redraw();
}
main() {
Fl::add_check(callback);
return Fl::run();
}

32.2.3.4 add_fd() [1/2]

```cpp
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

32.2.3.5 add_fd() [2/2]

```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.

32.2.3.6 add_idle()

```cpp
void Fl::add_idle (  
    FL_Idle_Handler cb,  
    void ∗ data = 0  )  [static]
```

Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).
The idle function can be used to get background processing done.
You can have multiple idle callbacks. To remove an idle callback use Fl::remove_idle().
Fl::wait() and Fl::check() call idle callbacks, but Fl::ready() does not.
The idle callback can call any FLTK functions, including Fl::wait(), Fl::check(), and Fl::ready().
FLTK will not recursively call the idle callback.

32.2.3.7 add_timeout()

```cpp
void Fl::add_timeout (  
    double t,  
    FL_Timeout_Handler cb,  
    void ∗ argp = 0  )  [static]
```

Adds a one-shot timeout callback.
The function will be called by Fl::wait() at t seconds after this function is called. The optional void* argument is passed to the callback.
You can have multiple timeout callbacks. To remove a timeout callback use Fl::remove_timeout().
If you need more accurate, repeated timeouts, use Fl::repeat_timeout() to reschedule the subsequent timeouts.
The following code will print “TICK” each second on stdout with a fair degree of accuracy:

```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();
}
```

### 32.2.3.8 api_version()

```c
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.

### 32.2.3.9 arg()

```c
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().

Generated by Doxygen
- 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.

### 32.2.3.10 args() [1/2]

```cpp
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.

### 32.2.3.11 args() [2/2]

```cpp
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.

### 32.2.3.12 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.

32.2.3.13 background2()

```c
void Fl::background2 (uchar r, uchar g, uchar b) {static}
```

Changes the alternative background color.
This color is used as a background by Fl_Input and other text widgets.
This call may change fl_color(FL_FOREGROUND_COLOR) if it does not provide sufficient contrast to FL_BACKGROUND2_COLOR.

32.2.3.14 box_border_radius_max() [1/2]

```c
static int Fl::box_border_radius_max () {inline}, {static}
```

Get the maximum border radius of all "rounded" boxtypes in pixels.

Since

1.4.0

32.2.3.15 box_border_radius_max() [2/2]

```c
static void Fl::box_border_radius_max (int R) {inline}, {static}
```

Set the maximum border radius of all "rounded" boxtypes in pixels.
Must be at least 5, default = 15.

Note

This does not apply to the "round" boxtypes which have really round sides (i.e. composed of half circles) as opposed to "rounded" boxtypes that have only rounded corners with a straight border between corners.

The box border radius of "rounded" boxtypes is typically calculated as about 2/5 of the box height or width, whichever is smaller. The upper limit can be set by this method for all "rounded" boxtypes.

Since

1.4.0

32.2.3.16 box_color()

```c
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.

32.2.3.17 box_dh()

```c
int Fl::box_dh (Fl_Boxtype t) {static}
```

Returns the height offset for the given boxtype.

See also

box_dy().
32.2.3.18 box_dw()

```cpp
int Fl::box_dw (Fl_Boxtype t) [static]
```

Returns the width offset for the given boxtype.

See also

```cpp
box_dy()
```

32.2.3.19 box_dx()

```cpp
int Fl::box_dx (Fl_Boxtype t) [static]
```

Returns the X offset for the given boxtype.

See also

```cpp
box_dy()
```

32.2.3.20 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 overdraw of 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().

32.2.3.21 box_shadow_width() [1/2]

```cpp
static int Fl::box_shadow_width ( ) [inline], [static]
```

Get the box shadow width of all "shadow" boxtypes in pixels.

Since

1.4.0

32.2.3.22 box_shadow_width() [2/2]

```cpp
static void Fl::box_shadow_width ( int W) [inline], [static]
```

Set the box shadow width of all "shadow" boxtypes in pixels.

Must be at least 1, default = 3. There is no upper limit.

Since

1.4.0
32.2.3.23 check()

int Fl::check ( ) [static]
Same as Fl::wait(0).
Calling this during a big calculation will keep the screen up to date and the interface responsive:
while (!calculation_done()) {
    calculate();
    Fl::check();
    if (user_hit_abort_button()) break;
}
This returns non-zero if any windows are displayed, and 0 if no windows are displayed (this is likely to change in
future versions of FLTK).

32.2.3.24 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.

32.2.3.25 dnd_text_ops() [1/2]

static int Fl::dnd_text_ops ( ) [inline], [static]
Gets whether drag and drop text operations are supported.
This returns whether selected text can be dragged from text fields or dragged within a text field as a cut/paste
shortcut.

32.2.3.26 dnd_text_ops() [2/2]

static void Fl::dnd_text_ops ( int v ) [inline], [static]
Sets whether drag and drop text operations are supported.
This specifically affects whether selected text can be dragged from text fields or dragged within a text field as a
cut/paste shortcut.

32.2.3.27 draw_box_active()

int Fl::draw_box_active ( ) [static]
Determines if the currently drawn box is active or inactive.
If inactive, the box color should be changed to the inactive color.
See also
   Fl::box_color(Fl_Color c)

32.2.3.28 draw_GL_text_with_textures() [1/2]

static int Fl::draw_GL_text_with_textures ( ) [inline], [static]
returns whether whether OpenGL uses textures to draw all text.
Default is yes.
See also
   draw_GL_text_with_textures(int val)

Version
1.4.0
32.2.3.29 draw_GL_text_with_textures()  [2/2]

static void Fl::draw_GL_text_with_textures ( int val ) [inline], [static]
sets whether OpenGL uses textures to draw all text.
By default, FLTK draws OpenGL text using textures, if the necessary hardware support is available. Call Fl::draw_GL_text_with_textures(0) once in your program before the first call to gl_font() to have FLTK draw instead OpenGL text using a legacy, platform-dependent procedure. It's recommended not to deactivate

Parameters

| val | use 0 to prevent FLTK from drawing GL text with textures |

See also

  gl_texture_pile_height(int max)

Version

1.4.0

32.2.3.30 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.

32.2.3.31 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.

32.2.3.32 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.

32.2.3.33 is_scheme()

static int Fl::is_scheme ( const char * name ) [inline], [static]
Returns whether the current scheme is the given name.
This is a fast inline convenience function to support scheme-specific code in widgets, e.g. in their draw() methods, if required.
Use a valid scheme name, not NULL (although NULL is allowed, this is not a useful argument - see below). If Fl::scheme() has not been set or has been set to the default scheme ("none" or "base"), then this will always return 0 regardless of the argument, because Fl::scheme() is NULL in this case.

Note

The stored scheme name is always lowercase, and this method will do a case-sensitive compare, so you must provide a lowercase string to return the correct value. This is intentional for performance reasons.

Example:

```c
if (Fl::is_scheme("gtk+")) { your_code_here(); }
```

Parameters

| in | name lowercase | string of requested scheme name. |

Returns

1 if the given scheme is active, 0 otherwise.

See also

Fl::scheme(const char *name)

32.2.3.34 menu_linespacing() [1/2]

int Fl::menu_linespacing () [static]

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

32.2.3.35 menu_linespacing() [2/2]

void Fl::menu_linespacing ( int H ) [static]

Sets the default line spacing used by menus. Default is 4.

Parameters

| in | H | The new default line spacing between menu items, in pixels. |

32.2.3.36 option() [1/2]

bool Fl::option ( Fl::Option opt ) [static]

FLTK library options management.

This function needs to be documented in more detail. It can be used for more optional settings, such as using a native file chooser instead of the FLTK one wherever possible, disabling tooltips, disabling visible focus, disabling FLTK file chooser preview, etc. . There should be a command line option interface.

There should be an application that manages options system wide, per user, and per application.

Example:
If \( \text{Fl::option(Fl::OPTION_ARROW_FOCUS)} \)

{ ..on.. }

else

{ ..off.. }

Note

As of FLTK 1.3.0, options can be managed within fluid, using the menu Edit/Global FLTK Settings.

Parameters

| opt | which option |

Returns

true or false

See also

enum Fl::Fl_Option

Fl::option(Fl::Option, bool)

Since

FLTK 1.3.0

32.2.3.37 option() [2/2]

void Fl::option (  
    Fl_Option opt,  
    bool val ) [static]

Override an option while the application is running. This function does not change any system or user settings. Example:

Fl::option(Fl::OPTION_ARROW_FOCUS, true); // on  
Fl::option(Fl::OPTION_ARROW_FOCUS, false); // off

Parameters

| opt | which option |

| val | set to true or false |

See also

enum Fl::Fl_Option

bool Fl::option(Fl::Option)

32.2.3.38 own_colormap()

void Fl::own_colormap ( ) [static]

Makes FLTK use its own colormap. This may make FLTK display better and will reduce conflicts with other programs that want lots of colors. However the colors may flash as you move the cursor between windows. This does nothing if the current visual is not colormapped.
32.2.3.39 program_should_quit() [1/2]

static int Fl::program_should_quit () [inline], [static]
Returns non-zero when a request for program termination was received and accepted.
On the MacOS platform, the "Quit xxx" item of the application menu is such a request, that is considered 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

32.2.3.40 program_should_quit() [2/2]

static void Fl::program_should_quit ( int should_i ) [inline], [static]
Indicate to the FLTK library whether a program termination request was received and accepted.
A program may set this to 1, for example, while performing a platform-independent command asking the program to cleanly terminate, similarly to the "Quit xxx" item of the application menu under MacOS.

Version
1.4.0

32.2.3.41 readqueue()

Fl_Widget * Fl::readqueue ( ) [static]
Reads the default callback queue and returns the first widget.
All Fl_Widgets that don't have a callback defined use the default callback static Fl_Widget::default_callback() that puts a pointer to the widget in a queue. This method reads the oldest widget out of this queue.
The queue (FIFO) is limited (currently 20 items). If the queue overflows, the oldest entry (Fl_Widget *) is discarded.
Relying on the default callback and reading the callback queue with Fl::readqueue() is not recommended. If you need a callback, you should set one with Fl_Widget::callback(Fl_Callback *, void *) or one of its variants.

See also
Fl_Widget::callback()
Fl_Widget::callback(Fl_Callback *, void *)
Fl_Widget::default_callback()

32.2.3.42 ready()

int Fl::ready ( ) [static]
This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.
This returns true if Fl::check() would do anything (it will continue to return true until you call Fl::check() or Fl::wait()).
while (!calculation_done()) {
calculate();
if (Fl::ready()) {
do_expensive_cleanup();
Fl::check();
if (user_hit_abort_button()) break;
}
}
32.2.3.43  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*)

32.2.3.44  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 *)

32.2.3.45  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.

32.2.3.46  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.

32.2.3.47  repeat_timeout()

void Fl::repeat_timeout ( 
    double t,
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
You may only call this method inside a timeout callback 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();
}
```
32.2.3.48 run()

```cpp
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:

```cpp
while (!Fl::program_should_quit()) Fl::wait(1e20);
```

32.2.3.49 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()

32.2.3.50 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.

32.2.3.51 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.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>The new default size for widget scrollbars, in pixels.</td>
</tr>
</tbody>
</table>

32.2.3.52  set_box_color()

```cpp
decl void Fl::set_box_color (Fl_Color c) [static]
```

Sets the drawing color for the box that is currently drawn. This method sets the current drawing color `fl_color()` depending on the widget's state to either `c` or `fl_inactive(c)`. It should be used whenever a box background is drawn in the box (type) drawing code instead of calling `fl_color(Fl_Color bg)` with the background color `bg`, usually `Fl_Widget::color()`. This method is only useful inside box drawing code. Whenever a box is drawn with one of the standard box drawing methods, a static variable is set depending on the widget's current state - if the widget is ` 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)`

32.2.3.53  set_idle()

```cpp
decl 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.

32.2.3.54  use_high_res_GL() [1/2]

```cpp
decl 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

32.2.3.55  use_high_res_GL() [2/2]

```cpp
decl 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
32.2.3.56 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.

32.2.3.57 visible_focus() [1/2]

static int Fl::visible_focus ( ) [inline], [static]
Gets or sets the visible keyboard focus on buttons and other non-text widgets. The default mode is to enable keyboard focus for all widgets.

32.2.3.58 visible_focus() [2/2]

static void Fl::visible_focus ( int v ) [inline], [static]
Gets or sets the visible keyboard focus on buttons and other non-text widgets. The default mode is to enable keyboard focus for all widgets.

32.2.3.59 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.

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).

32.2.3.60 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.
32.2.3.61 wait() [2/2]

double Fl::wait ( double time_to_wait ) [static]
Waits a maximum of time_to_wait seconds or until "something happens".
See Fl::wait() for the description of operations performed when "something happens".

Returns
Always 1 on Windows. Otherwise, it is positive if an event or fd happens before the time elapsed. It is zero if nothing happens. It is negative if an error occurs (this will happen on X11 if a signal happens).

32.2.4 Member Data Documentation

32.2.4.1 help

const char *const Fl::help = helpmsg+13 [static]
Usage string displayed if Fl::args() detects an invalid argument. This may be changed to point to customized text at run-time.

32.2.4.2 idle

void(* Fl::idle)() [static]
The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
This is now used as part of a higher level system allowing multiple idle callback functions to be called.

See also
add_idle(), remove_idle()

The documentation for this class was generated from the following files:

- Fl.H
- Fl.cxx
- Fl_abort.cxx
- Fl_add_idle.cxx
- Fl_arg.cxx
- fl_boxtype.cxx
- fl_color.cxx
- Fl_compose.cxx
- Fl_display.cxx
- Fl_get_system_colors.cxx
- Fl_grab.cxx
- fl_labeltype.cxx
- Fl_lock.cxx
- Fl_own_colormap.cxx
- fl_set_font.cxx
- flShortcut.cxx
- Fl_visual.cxx
- Fl_Widget.cxx
- Fl_Window.cxx
- screen_xywh.cxx
- Fl_Cairo.cxx
32.3 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
   |
   v
Fl_Valuator
   |
   v
Fl_Adjuster
```

Public Member Functions

- **Fl_Adjuster**(int X, int Y, int W, int H, const char ∗l=0)
  
  Creates a new Fl_Adjuster widget using the given position, size, and label string.

- **int soft**(const)
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **void soft**(int s)
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

Protected Member Functions

- **void draw**(const)
  
  Draws the widget.

- **int handle**(int)
  
  Handles the specified event.

- **void value_damage**(const)
  
  Asks for partial redraw.

Additional Inherited Members

32.3.1 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.

![Figure 32.1 Fl_Adjuster](image)

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 * step(), the next by 10 * step() and that smallest button by step(). Clicking on the buttons increments by 10 times the amount dragging by a pixel does. Shift + click decrements by 10 times the amount.
### 32.3.2 Constructor & Destructor Documentation

#### 32.3.2.1 Fl_Adjuster()

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.

### 32.3.3 Member Function Documentation

#### 32.3.3.1 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.

#### 32.3.3.2 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>in 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.
32.3.3.3 soft() [1/2]

int Fl_Adjuster::soft ( ) const [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

32.3.3.4 soft() [2/2]

void Fl_Adjuster::soft ( int s ) [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

The documentation for this class was generated from the following files:

- Fl_Adjuster.H
- Fl_Adjuster.cxx

32.4 Fl_Bitmap Class Reference

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.
#include `<Fl_Bitmap.H>`

Inheritance diagram for Fl_Bitmap:

```
Inheritance | 
---|---
| Fl_Bitmap |
| Fl_Image |
| Fl_XBM_Image |
```

Public Member Functions

- Fl_Bitmap ∗ copy () const
- virtual Fl_Bitmap ∗ copy (int W, int H) const
  
  Creates a resized copy of the image.
- void draw (int X, int Y)
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws the image to the current drawing surface with a bounding box.
- Fl_Bitmap (const char ∗ bits, int W, int H)
  
  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗ bits, int W, int H)
  
  The constructors create a new bitmap from the specified bitmap data.
- virtual void label (Fl_Menu_Item ∗ m)
  
  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget ∗ w)
  
  This method is an obsolete way to set the image attribute of a widget or menu item.
- 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

32.4.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images. Images are drawn using the current color.

32.4.2 Member Function Documentation

32.4.2.1 copy()

Fl_Image ∗ Fl_Bitmap::copy (int W, int H) const [virtual]

Creates a resized copy of the image.

The new image should be released when you are done with it.

Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e., all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.

The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().

For the new image the following equations are true:

- w() == data_w() == W
- h() == data_h() == H

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Parameters

| in | W, H | Requested width and height of the new image |

Note

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

32.4.2.2 draw()

void Fl_Bitmap::draw (
int X,
int Y,
int W,
int H,
int cx = 0,
int cy = 0 ) [virtual]
Draws the image 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.

32.4.2.3 label() [1/2]

void Fl_Bitmap::label (  
Fl_Menu_Item * m ) [virtual]
This method is an obsolete way to set the image attribute of a menu item.
Deprecated Please use Fl_Menu_Item::image() instead.
Reimplemented from Fl_Image.

32.4.2.4 label() [2/2]

void Fl_Bitmap::label (  
Fl_Widget * widget ) [virtual]
This method is an obsolete way to set the image attribute of a widget or menu item.
Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.
Reimplemented from Fl_Image.

32.4.2.5 uncache()

void Fl_Bitmap::uncache ( ) [virtual]
If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:
- Fl_Bitmap.H
- Fl_Bitmap.cxx

32.5 Fl_BMP_Image Class Reference

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.
#include <Fl_BMP_Image.H>
Inheritance diagram for Fl_BMP_Image:

```
Fl_Image
 |   
|   Fl_RGB_Image
|   |
|   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, const long length=-1)
  
  This constructor loads a BMP image from memory.

Protected Member Functions

- void load_bmp_(class Fl_Image_Reader &rdr)

Additional Inherited Members

32.5.1 Detailed Description

The **Fl_BMP_Image** class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

32.5.2 Constructor & Destructor Documentation

32.5.2.1 **Fl_BMP_Image() [1/2]**

```cpp
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_BMP_Image::fail()** to check if **Fl_BMP_Image** failed to load. **fail()** returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

**Parameters**

| in  | filename | a full path and name pointing to a BMP file. |

See also

```cpp
Fl_BMP_Image::Fl_BMP_Image(const char* imagename, const unsigned char *data, const long length = -1);
```

32.5.2.2 **Fl_BMP_Image() [2/2]**

```cpp
Fl_BMP_Image::Fl_BMP_Image (const char *imagename,
 const unsigned char *data,
 const long length = -1)
```

This constructor loads a BMP image from memory. Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. **imagename** can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name. The destructor frees all memory and server resources that are used by the image. The (new and optional) third parameter **length** should be used so buffer overruns (i.e. truncated images) can be checked. See note below. If **length** is not used

- it defaults to -1 (unlimited size)
- buffer overruns will not be checked.
32.6 Fl_Box Class Reference

Note

The optional parameter length is available since FLTK 1.4.0. Not using it is deprecated and old code should be modified to use it. This parameter will likely become mandatory in a future FLTK version.

Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the image could not be read from memory, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

<table>
<thead>
<tr>
<th>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 BMP image in memory.</td>
</tr>
<tr>
<td>in</td>
<td>length</td>
<td>Length of the BMP image in memory.</td>
</tr>
</tbody>
</table>

See also

Fl_BMP_Image::Fl_BMP_Image(const char *filename)

Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_BMP_Image.H
- Fl_BMP_Image.hxx

32.6 Fl_Box Class Reference

This widget simply draws its box, and possibly its label.

#include <Fl_Box.H>

Inheritance diagram for Fl_Box:

```
Fl_Widget
    Fl_Box
```

Public Member Functions

- Fl_Box (Fl_Boxtype b, int X, int Y, int W, int H, const char *l)
  See Fl_Box::Fl_Box(int x, int y, int w, int h, const char * = 0)

- Fl_Box (int X, int Y, int W, int H, const char *l=0)
  virtual int handle (int)
  Handles the specified event.

Protected Member Functions

- void draw ()
  Draws the widget.

Additional Inherited Members

32.6.1 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.
32.6.2 Constructor & Destructor Documentation

32.6.2.1 Fl_Box()

```cpp
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.

32.6.3 Member Function Documentation

32.6.3.1 draw()

```cpp
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.

32.6.3.2 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**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

**Return values**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the event was not used or understood</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
<td></td>
</tr>
</tbody>
</table>
See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Box.H
- Fl_Box.cxx

32.7 Fl_Browser Class Reference

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.

```c
#include <Fl_Browser.H>
```

Inheritance diagram for Fl_Browser:

```
Fl_Widget
  |
  v
Fl_Group
  |
  v
Fl_Browser_
  |
  v
Fl_Browser
  |
  v
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)`

Generated by Doxygen
For back compatibility,

- **int displayed** (int line) const
  
  Returns non-zero if line has been scrolled to a position where it is being displayed.

- **Fl_Browser** (int X, int Y, int W, int H, const char *L=0)
  
  The constructor makes an empty browser.

- **char format_char () const**
  
  Gets the current format code prefix character, which by default is '@'.

- **void format_char (char c)**
  
  Sets the current format code prefix character to c.

- **void hide ()**
  
  Hides the entire Fl_Browser widget – opposite of show().

- **void hide (int line)**
  
  Makes line invisible, preventing selection by the user.

- **Fl_Image * icon (int line) const**
  
  Returns the icon currently defined for line.

- **void icon (int line, Fl_Image *icon)**
  
  Set the image icon for line to the value icon.

- **void insert (int line, const char *newtext, void *d=0)**
  
  Insert a new entry whose label is newtext above given line, optional data d.

- **void lineposition (int line, Fl_Line_Position pos)**
  
  Updates the browser so that line is shown at position pos.

- **int load (const char *filename)**
  
  Clears the browser and reads the file, adding each line from the file to the browser.

- **void make_visible (int line)**
  
  Make the item at the specified line visible().

- **void middleline (int line)**
  
  Scrolls the browser so the middle item in the browser is showing the specified line.

- **void move (int to, int from)**
  
  Line from is removed and reinserted at to.

- **void remove (int line)**
  
  Remove entry for given line number, making the browser one line shorter.

- **void remove_icon (int line)**
  
  Removes the icon for line.

- **void replace (int a, const char *b)**
  
  For back compatibility only.

- **int select (int line, int val=1)**
  
  Sets the selection state of the item at line to the value val.

- **int selected (int line) const**
  
  Returns 1 if specified line is selected, 0 if not.

- **void show ()**
  
  Shows the entire Fl_Browser widget – opposite of hide().

- **void show (int line)**
  
  Makes line visible, and available for selection by user.

- **int size () const**
  
  Returns how many lines are in the browser.

- **void size (int W, int H)**

- **void swap (int a, int b)**
  
  Swaps two browser lines a and b.

- **const char * text (int line) const**
  
  Returns the label text for the specified line.

- **void text (int line, const char *newtext)**
Sets the text for the specified line to newtext.

- **Fl_Fontsize textsize () const**
  Gets the default text size (in pixels) for the lines in the browser.

- **void textsize (Fl_Fontsize newSize)**
  Sets the default text size (in pixels) for the lines in the browser to newSize.

- **int topline () const**
  Returns the line that is currently visible at the top of the browser.

- **void topline (int line)**
  Scrolls the browser so the top item in the browser is showing the specified line.

- **int value () const**
  Returns the line number of the currently selected line, or 0 if none selected.

- **void value (int line)**
  Sets the browser's value(), which selects the specified line.

- **int visible (int line) const**
  Returns non-zero if the specified line is visible, 0 if hidden.

- **~Fl_Browser ()**
  The destructor deletes all list items and destroys the browser.

**Protected Member Functions**

- **FL_BLINE * _remove (int line)**
  Removes the item at the specified line.

- **FL_BLINE * find_line (int line) const**
  Returns the item for specified line.

- **int full_height () const**
  The height of the entire list of all visible() items in pixels.

- **int incr_height () const**
  The default 'average' item height (including inter-item spacing) in pixels.

- **void insert (int line, FL_BLINE * item)**
  Insert specified item above line.

- **void * item_at (int line) const**
  Return the item at specified line.

- **void item_draw (void * item, int X, int Y, int W, int H) const**
  Draws item at the position specified by XYWH.

- **void * item_first () const**
  Returns the very first item in the list.

- **int item_height (void * item) const**
  Returns height of item in pixels.

- **void * item_last () const**
  Returns the very last item in the list.

- **void * item_next (void * item) const**
  Returns the next item after item.

- **void * item_prev (void * item) const**
  Returns the previous item before item.

- **void item_select (void * item, int val)**
  Change the selection state of item to the value val.

- **int item_selected (void * item) const**
  See if item is selected.

- **void item_swap (void *a, void *b)**
  Swap the items a and b.

- **const char * item_text (void *item) const**
Returns the label text for item.

- int item_width (void *item) const
  Returns width of item in pixels.
- int lineno (void *item) const
  Returns line number corresponding to item, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b)
  Swap the two items a and b.

Additional Inherited Members

32.7.1 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 32.2 Fl_Hold_Browser](image)

Each line in the browser is identified by number. The numbers start at one (this is so that zero can be reserved for "no line" in the selective browsers). Unless otherwise noted, the methods do not check to see if the passed line number is in range and legal. It must always be greater than zero and \(<\) = size(). Each line contains a null-terminated string of text and a void * data pointer. The text string is displayed, the void * pointer can be used by the callbacks to reference the object the text describes.

The base class does nothing when the user clicks on it. The subclasses Fl_Select_Browser, Fl_Hold_Browser, and Fl_Multi_Browser react to user clicks to select lines in the browser and do callbacks.

The base class Fl_Browser_ provides the scrolling and selection mechanisms of this and all the subclasses, but the dimensions and appearance of each item are determined by the subclass. You can use Fl_Browser_ to display information other than text, or text that is dynamically produced from your own data structures. If you find that loading the browser is a lot of work or is inefficient, you may want to make a subclass of Fl_Browser_.

Some common coding patterns used for working with Fl_Browser:

```c
// How to loop through all the items in the browser
for ( int t=1; t<browser->size(); t++ ) {
  printf("item %d, label='%s'\n", t, browser->text(t));
}
```

Note: If you are subclassing Fl_Browser, it's more efficient to use the protected methods item_first() and item_next(), since Fl_Browser internally uses linked lists to manage the browser's items. For more info, see find_item(int).

32.7.2 Constructor & Destructor Documentation

32.7.2.1 Fl_Browser()

Fl_Browser::Fl_Browser {
  int X,
int Y,
int W,
int H,
const char ∗ L = 0 )
The constructor makes an empty browser.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y, W, H</td>
<td>position and size.</td>
</tr>
<tr>
<td>L</td>
<td>label string, may be NULL.</td>
</tr>
</tbody>
</table>

### 32.7.3 Member Function Documentation

#### 32.7.3.1 _remove()

```
FL_BLINE ∗ Fl_Browser::_remove ( int line ) [protected]
```

Removes the item at the specified `line`.

Caveat: See efficiency note in `find_line()`. You must call `redraw()` to make any changes visible.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>The line number to be removed. (1 based) Must be in range!</td>
</tr>
</tbody>
</table>

Returns

Pointer to browser item that was removed (and is no longer valid).

See also

`add()`, `insert()`, `remove()`, `swap(int,int)`, `clear()`

#### 32.7.3.2 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newtext</td>
<td>The label text used for the added item</td>
</tr>
<tr>
<td>d</td>
<td>Optional user <code>data()</code> for the item (0 if unspecified)</td>
</tr>
</tbody>
</table>

See also

`add()`, `insert()`, `remove()`, `swap(int,int)`, `clear()`
32.7.3.3  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()

32.7.3.4  clear()

void Fl_Browser::clear ( )

Removes all the lines in the browser.

See also

add(), insert(), remove(), swap(int,int), clear()

32.7.3.5  column_char() [1/2]

char Fl_Browser::column_char ( ) const [inline]

Gets the current column separator character.

The default is ‘t’ (tab).

See also

column_char(), column_widths()

32.7.3.6  column_char() [2/2]

void Fl_Browser::column_char (  
    char c  ) [inline]

Sets the column separator to c.

This will only have an effect if you also set column_widths(). The default is ‘t’ (tab).

See also

column_char(), column_widths()

32.7.3.7  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:

```c
FLBrowser *b = new FLBrowser(...);
static int widths[] = { 50, 50, 50, 70, 70, 70, 70, 70, 70, 70, 0 } ; // widths for each column
b->column_widths(widths) ; // assign array to widget
b->column_char(‘\t’); // use tab as the column character
b->add("USER \tPID \tCPU \tMEM \tVSZ \tRSS \tTTY \tSTAT \tSTART \tTIME \tCOMMAND");
```

Generated by Doxygen
See also

column_char(), column_widths()

32.7.3.8 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()

32.7.3.9 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)

32.7.3.10 data() [2/2]

void Fl_Browser::data ( int line,
    void * d )
Sets the user data for specified line to d.
Does nothing if line is out of range.

Parameters

| in | line | The line of the item whose data() is to be changed. (1 based) |
| in | d   | The new data to be assigned to the item. (can be NULL) |

32.7.3.11 display()

void Fl_Browser::display ( int line,
    int val = 1 )
For back compatibility.
This calls show(line) if val is true, and hide(line) otherwise. If val is not specified, the default is 1 (makes the line visible).
See also

show(int), hide(int), display(), visible(), make_visible()

32.7.3.12 displayed()

int Fl_Browser::displayed (int line) const [inline]

Returns non-zero if line has been scrolled to a position where it is being displayed.
Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does
NOT take into account the hide()/show() status of the widget or item.

Parameters

| in | line | The line to be checked |

Returns

1 if visible, 0 if not visible.

See also
topline(), middleline(), bottomline(), displayed(), lineposition()

32.7.3.13 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

| in | line | The line number of the item to return. (1 based) |

Return values

| item | that was found. |
| NULL | if line is out of range. |

See also

item_at(), find_line(), lineno()

32.7.3.14 format_char() [1/2]

c char Fl_Browser::format_char ( ) const [inline]

Gets the current format code prefix character, which by default is '@'.
A string of formatting codes at the start of each column are stripped off and used to modify how the rest of the line
is printed:
• '@.'  Print rest of line, don’t look for more '@' signs
• '@@'  Doubling the format character prints the format character once, followed by the rest of line
• '@l'  Use a LARGE (24 point) font
• '@m'  Use a medium large (18 point) font
• '@s'  Use a small (11 point) font
• '@b'  Use a bold font (adds FL_BOLD to font)
• '@i'  Use an italic font (adds FL_ITALIC to font)
• '@f' or '@t' Use a fixed-pitch font (sets font to FL_COURIER)
• '@c'  Center the line horizontally
• '@r'  Right-justify the text
• '@N'  Use fl_inactive_color() to draw the text
• '@B0', '@B1', ... '@B255'  Fill the background with fl_color(n)
• '@C0', '@C1', ... '@C255'  Use fl_color(n) to draw the text
• '@F0', '@F1', ... Use fl_font(n) to draw the text
• '@S1', '@S2', ... Use point size n to draw the text
• '@u' or '@_'. Underline the text.
• '@-'  draw an engraved line through the middle.

Notice that the '@.' command can be used to reliably terminate the parsing. To print a random string in a random
color, use sprintf("@C%d@.%s", color, string) and it will work even if the string starts with a digit or
has the format character in it.

32.7.3.15 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

32.7.3.16 full_height()
int Fl_Browser::full_height ( ) const [protected], [virtual]
The height of the entire list of all visible() items in pixels.
This returns the accumulated height of all the items in the browser that are not hidden with hide(), including items
scrolled off screen.

Returns  
The accumulated size of all the visible items in pixels.

See also  
item_height(), item_width(),
incr_height(), full_height()
Reimplemented from Fl_Browser_.
32.7.3.17 hide()

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()

32.7.3.18 icon() [1/2]

Fl_Image * Fl_Browser::icon (  
    int line ) const

Returns the icon currently defined for line.  
If no icon is defined, NULL is returned.

Parameters

| in | line | The line whose icon is returned |

Returns

The icon defined, or NULL if none.

32.7.3.19 icon() [2/2]

void Fl_Browser::icon (  
    int line,  
    Fl_Image * icon )

Set the image icon for line to the value icon. 
Caller is responsible for keeping the icon allocated. 
The line is automatically redrawn.

Parameters

| in | line | The line to be modified. If out of range, nothing is done |
| in | icon | The image icon to be assigned to the line. If NULL, any previous icon is removed |

32.7.3.20 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_.

### 32.7.3.21 insert() [1/2]

```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**

<table>
<thead>
<tr>
<th>in</th>
<th><strong>line</strong></th>
<th>Line position for insert. (1 based) If <code>line &gt; size()</code>, the entry will be added at the end.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>newtext</strong></td>
<td>The label text for the new line.</td>
</tr>
<tr>
<td>in</td>
<td><strong>d</strong></td>
<td>Optional pointer to user data to be associated with the new line.</td>
</tr>
</tbody>
</table>

### 32.7.3.22 insert() [2/2]

```cpp
void Fl_Browser::insert (  
  FL_BLINE * item ) [protected]
```

Insert specified `item` above `line`.
If `line > size()` then the line is added to the end.
Caveat: See efficiency note in find_line().

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><strong>line</strong></th>
<th>The new line will be inserted above this line (1 based).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>item</strong></td>
<td>The item to be added.</td>
</tr>
</tbody>
</table>

### 32.7.3.23 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><strong>line</strong></th>
<th>The line of the item to return. (1 based)</th>
</tr>
</thead>
</table>
32.7.3.24 item_draw()


```c
void Fl_Browser::item_draw (
    void * item,
    int X,  
    int Y,  
    int W,  
    int H  ) const [protected], [virtual]
```

Draws item at the position specified by XYWH. The W and H values are used for clipping. Should only be called within the context of an FLTK draw().

Parameters

| in | item            | The item to be drawn |
|    | X,Y,W,H position and size. |

Implements Fl_Browser_.

32.7.3.25 item_first()

```c
void * Fl_Browser::item_first ( ) const [protected], [virtual]
```

Returns the very first item in the list.

Example of use:
```c
// Walk the browser from beginning to end
for ( void *i=item_first(); i; i=item_next(i) ) {
    printf("item label=\'%s\', 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_.

32.7.3.26 item_height()

```c
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 item, or NULL if line out of range.

See also

item_at(), find_line(), lineno()
Returns

The height of the item in pixels.

See also

item_height(), item_width(), incr_height(), full_height()

Implements Fl_Browser_.

32.7.3.27  item_last()

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_.

32.7.3.28  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_.

32.7.3.29  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_.

### 32.7.3.30 item_select()

```cpp
void Fl_Browser::item_select ( void * item, int val ) [protected], [virtual]
```

Change the selection state of item to the value val.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to be changed.</th>
</tr>
</thead>
<tbody>
<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_.

### 32.7.3.31 item_selected()

```cpp
int Fl_Browser::item_selected ( void * item ) const [protected], [virtual]
```

See if item is selected.

**Parameters**

| in | item | The item whose selection state is to be checked. |

Returns

1 if selected, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

### 32.7.3.32 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__.

### 32.7.3.33 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__.

### 32.7.3.34 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__.

### 32.7.3.35 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().

Generated by Doxygen
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()

32.7.3.36  lineposition()

void Fl_Browser::lineposition (  
    int line,  
    Fl_Line_Position pos  
)  

Updates the browser so that line is shown at position pos.

Parameters

in  line  line number. (1 based)  

in  pos  position.

See also

topline(), middleline(), bottomline()

32.7.3.37  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

in  filename  The filename to load

Returns

1 if OK, 0 on error (errno has reason)

See also

add()

32.7.3.38  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

| in | line | The line to be made visible. |

See also

- show(int), hide(int), display(), visible(), make_visible()

### 32.7.3.39 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()

### 32.7.3.40 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 |

### 32.7.3.41 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

| in | line | Line to be removed. (1 based) |
| | | If line is out of range, no action is taken. |
See also

add(), insert(), remove(), swap(int,int), clear()

32.7.3.42 remove_icon()

```cpp
void Fl_Browser::remove_icon (int line)
```

Removes the icon for line.
It's ok to remove an icon if none has been defined.

Parameters

| in | line | The line whose icon is to be removed. |

32.7.3.43 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

| in | line | The line number of the item to be changed. (1 based) |
| in | val | The new selection state (1=select, 0=de-select). |

Returns

1 if the state changed, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

32.7.3.44 selected()

```cpp
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()
32.7.3.45  show()

```cpp
void Fl_Browser::show (  
  int line  )
```

Makes line visible, and available for selection by user. Opposite of hide(int). This changes the full_height() if the state was changed. redraw() is called automatically if a change occurred.

**Parameters**

| in  | line | The line to be shown. (1 based) |

See also

- show(int), hide(int), display(), visible(), make_visible()

32.7.3.46  size()

```cpp
int Fl_Browser::size ( ) const [inline]
```

Returns how many lines are in the browser. The last line number is equal to this. Returns 0 if browser is empty.

32.7.3.47  swap() [1/2]

```cpp
void Fl_Browser::swap (  
  FL_BLINE * a,  
  FL_BLINE * b ) [protected]
```

Swap the two items a and b. Uses swapping() to ensure list updates correctly.

**Parameters**

| in  | a,b | The two items to be swapped. |

See also

- swap(int,int), item_swap()

32.7.3.48  swap() [2/2]

```cpp
void Fl_Browser::swap (  
  int a,  
  int b )
```

Swaps two browser lines a and b. You must call redraw() to make any changes visible.

**Parameters**

| in  | a,b | The two lines to be swapped. (both 1 based) |

See also

- swap(int,int), item_swap()
32.7.3.49 text() [1/2]

const char * Fl_Browser::text ( 
    int line ) const
Returns the label text for the specified line.
Return value can be NULL if line is out of range or unset. The parameter line is 1 based.

Parameters

| in | line | The line number of the item whose text is returned. (1 based) |

Returns

The text string (can be NULL)

32.7.3.50 text() [2/2]

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. |

32.7.3.51 textsize()

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.

32.7.3.52 topline() [1/2]

int Fl_Browser::topline ( ) const
Returns the line that is currently visible at the top of the browser.
If there is no vertical scrollbar then this will always return 1.

Returns

The lineno() of the top() of the browser.
32.7.3.53 topline() [2/2]

```
void Fl_Browser::topline (  
    int line ) [inline]
```

Scrolls the browser so the top item in the browser is showing the specified line.

Parameters

| in  | line  | The line to be displayed at the top. |

See also
topline(), middleline(), bottomline(), displayed(), lineposition()

32.7.3.54 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()

32.7.3.55 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()

32.7.3.56 visible()

```
int Fl_Browser::visible (  
    int line ) const
```

Returns non-zero if the specified line is visible, 0 if hidden. Use show(int), hide(int), or make_visible(int) to change an item’s visible state.

Parameters

| in  | line  | The line in the browser to be tested. (1 based) |

See also
show(int), hide(int), display(), visible(), make_visible()

The documentation for this class was generated from the following files:

- Fl_Browser.H
### 32.8 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`
  
  Gets the vertical scroll position of the list as a pixel position `pos`.

- `void position (int pos)`
  
  Sets the vertical scroll position of the list to pixel position `pos`.

- `void resize (int X, int Y, int W, int H)`
  
  Repositions and/or resizes the browser.

- `void scrollbar_left ()`
  
  Moves the vertical scrollbar to the lefthand side of the list.
- `void scrollbar_right ()`
  Moves the vertical scrollbar to the right hand side of the list.
- `int scrollbar_size () const`
  Gets the current size of the scrollbars' troughs, in pixels.
- `void scrollbar_size (int newSize)`
  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- `int scrollbar_width () const`
  Returns the global value `Fl::scrollbar_size()`.
- `void scrollbar_width (int width)`
  Sets the global `Fl::scrollbar_size()`, and forces this instance of the widget to use it.
- `int select (void *item, int val=1, int docallbacks=0)`
  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- `int select_only (void *item, int docallbacks=0)`
  Selects `item` and returns 1 if the state changed or 0 if it did not.
- `void sort (int flags=0)`
  Sorts the items in the browser based on `flags`.
- `Fl_Color textcolor () const`
  Gets the default text color for the lines in the browser.
- `void textcolor (Fl_Color col)`
  Sets the default text color for the lines in the browser to color `col`.
- `Fl_Font textfont () const`
  Gets the default text font for the lines in the browser.
- `void textfont (Fl_Font font)`
  Sets the default text font for the lines in the browser to `font`.
- `Fl_Fontsize textsize () const`
  Gets the default text size (in pixels) for the lines in the browser.
- `void textsize (Fl_Fontsize newSize)`
  Sets the default text size (in pixels) for the lines in the browser to `size`.

### Public Attributes
- `Fl_Scrollbar hscrollbar`
  Horizontal scrollbar.
- `Fl_Scrollbar scrollbar`
  Vertical scrollbar.

### Protected Member Functions
- `void bbox (int &X, int &Y, int &W, int &H) const`
  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- `void deleting (void *item)`
  This method should be used when `item` is being deleted from the list.
- `int displayed (void *item) const`
  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- `void draw ()`
  Draws the list within the normal widget bounding box.
- `void *find_item (int ypos)`
  This method returns the item under mouse y position `ypos`.
- `Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)`
  The constructor makes an empty browser.
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

32.8.1 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.

32.8.1.1 Keyboard navigation of browser items

The keyboard navigation of browser items is only possible if visible_focus() is enabled. If disabled, the widget rejects
keyboard focus; Tab and Shift-Tab focus navigation will skip the widget.
In 'Select' and 'Normal' mode, the widget rejects keyboard focus; no navigation keys are supported (other than
scrollbar positioning).
In 'Hold' mode, the widget accepts keyboard focus, and Up/Down arrow keys can navigate the selected item.
In 'Multi' mode, the widget accepts keyboard focus, and Up/Down arrow keys navigate the focus box; Space toggles
the current item's selection, Enter selects only the current item (deselects all others). If Shift (or Ctrl) is combined
with Up/Down arrow keys, the current item's selection state is extended to the next item. In this way one can extend
a selection or de-selection.

32.8.2 Member Enumeration Documentation

32.8.2.1 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

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HORIZONTAL</td>
<td>Only show horizontal scrollbar.</td>
</tr>
<tr>
<td>VERTICAL</td>
<td>Only show vertical scrollbar.</td>
</tr>
<tr>
<td>BOTH</td>
<td>Show both scrollbars. (default)</td>
</tr>
<tr>
<td>ALWAYS_ON</td>
<td>Specified scrollbar(s) should 'always' be shown (to be used with HORIZONTAL/VERTICAL)</td>
</tr>
<tr>
<td>HORIZONTAL_ALWAYS</td>
<td>Horizontal scrollbar always on.</td>
</tr>
<tr>
<td>VERTICAL_ALWAYS</td>
<td>Vertical scrollbar always on.</td>
</tr>
<tr>
<td>BOTH_ALWAYS</td>
<td>Both scrollbars always on.</td>
</tr>
</tbody>
</table>

32.8.3 Constructor & Destructor Documentation
32.8.3.1 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>in</th>
<th>X,Y,W,H</th>
<th>position and size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>The label string, may be NULL</td>
</tr>
</tbody>
</table>

32.8.4 Member Function Documentation

32.8.4.1 bbox()

void Fl_Browser_::bbox (  
  int & X,  
  int & Y,  
  int & W,  
  int & H ) const [protected]

Returns the bounding box for the interior of the list's display window, inside the scrollbars.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>X,Y,W,H</th>
<th>The returned bounding box.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(The original contents of these parameters are overwritten)</td>
</tr>
</tbody>
</table>

32.8.4.2 deleting()

void Fl_Browser_::deleting (  
  void * item ) [protected]

This method should be used when item is being deleted from the list. It allows the Fl_Browser_ to discard any cached data it has on the item. This method does not actually delete the item, but handles the follow up bookkeeping after the item has just been deleted.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item being deleted.</th>
</tr>
</thead>
</table>

32.8.4.3 deselect()

int Fl_Browser_::deselect (  
  int docallbacks = 0 )

Deselects all items in the list and returns 1 if the state changed or 0 if it did not. If the optional docallbacks parameter is non-zero, deselect tries to call the callback function for the widget.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>callbacks</th>
</tr>
</thead>
</table>
| If non-zero, invokes widget callback if item changed. 
If 0, doesn't do callback (default). |

32.8.4.4 display()

```cpp
void Fl_Browser_::display (void * item)
```
Displays the item, scrolling the list as necessary.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>The item to be displayed.</td>
<td></td>
</tr>
</tbody>
</table>

See also

display(), displayed()

32.8.4.5 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

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>The item to check</td>
<td></td>
</tr>
</tbody>
</table>

Returns

1 if visible, 0 if not visible.

See also

display(), displayed()

32.8.4.6 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

<table>
<thead>
<tr>
<th>in</th>
<th>ypos</th>
</tr>
</thead>
<tbody>
<tr>
<td>The y position (eg. Fl::event_y()) to find an item under.</td>
<td></td>
</tr>
</tbody>
</table>
Returns

The item, or NULL if not found

### 32.8.4.7 full_height()

```
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.

### 32.8.4.8 full_width()

```
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.

### 32.8.4.9 handle()

```
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.

### 32.8.4.10 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.
32.8 Fl_Browser Class Reference

- **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.

32.8.4.11  hposition() [1/2]

```cpp
int Fl_Browser::hposition ( ) const [inline]
```

Gets the horizontal scroll position of the list as a pixel position `pos`.
The position returned is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' indicates the left 18 pixels of the list are scrolled off the left edge of the screen.

See also:
- position(), hposition()

32.8.4.12  hposition() [2/2]

```cpp
void Fl_Browser::hposition ( int pos )
```

Sets the horizontal scroll position of the list to pixel position `pos`.
The position is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' scrolls the left 18 pixels of the list off the left edge of the screen.

Parameters

- `pos` - The horizontal position (in pixels) to scroll the browser to.

See also:
- position(), hposition()

32.8.4.13  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.

32.8.4.14  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.

Generated by Doxygen
32.8.4.15 item_at()

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.

32.8.4.16 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_Check_Browser, and Fl_Browser.

32.8.4.17 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.

32.8.4.18 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.

32.8.4.19 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.

32.8.4.20 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.

32.8.4.21 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.

32.8.4.22 item_select()

void Fl_Browser::item_select (void * item,
                             int val = 1) [protected], [virtual]

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.

Sets the selection state for item, where optional val is 1 (select, the default) or 0 (de-select).
Parameters

<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 in Fl_Browser, and Fl_Check_Browser.

32.8.4.23 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 item.
The method should return 1 if item is selected, or 0 otherwise.

Parameters

| in | item | The item to test. |

Reimplemented in Fl_Browser, and Fl_Check_Browser.

32.8.4.24 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_Check_Browser, and Fl_Browser.

32.8.4.25 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_Check_Browser, and Fl_Browser.

32.8.4.26 item_width()

```cpp
virtual int Fl_Browser_::item_width ( void ∗ item ) const [inline], [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**

| in item | The item whose width is returned. |

**Returns**

The width of the item in pixels.

Implemented in Fl_Browser, and Fl_Check_Browser.

### 32.8.4.27 leftedge()

```cpp
def 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()

### 32.8.4.28 new_list()

```cpp
def 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.

### 32.8.4.29 position() [1/2]

```cpp
def 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()

### 32.8.4.30 position() [2/2]

```cpp
def 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. |
32.8.4.31 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()`

32.8.4.32 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()`

32.8.4.33 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.

**Parameters**

| in | a | Item being replaced |
| in | b | Item to replace 'a' |

32.8.4.34 resize()

```cpp
void Fl_Browser_::resize ( int X, int Y, int W, int H ) [virtual]
```

Repositions and/or resizes the browser.
Parameters

| in | X,Y,W,H | The new position and size for the browser, in pixels. |

Reimplemented from Fl_Widget.

32.8.4.35 scrollbar_left()

```cpp
void Fl_Browser_::scrollbar_left ( ) [inline]
```

Moves the vertical scrollbar to the lefthand side of the list.
For back compatibility.

32.8.4.36 scrollbar_right()

```cpp
void Fl_Browser_::scrollbar_right ( ) [inline]
```

Moves the vertical scrollbar to the righthand side of the list.
For back compatibility.

32.8.4.37 scrollbar_size() [1/2]

```cpp
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)

32.8.4.38 scrollbar_size() [2/2]

```cpp
void Fl_Browser_::scrollbar_size ( int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>newSize</th>
<th>Sets the scrollbar size in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

See also

Fl::scrollbar_size()

32.8.4.39 scrollbar_width() [1/2]

```cpp
int Fl_Browser_::scrollbar_width ( ) const [inline]
```

Generated by Doxygen
Returns the global value \code{Fl::scrollbar_size()}.

**Deprecated** Use \code{scrollbar_size()} instead.

**Todo** This method should eventually be removed in 1.4+

### 32.8.4.40 \code{scrollbar_width()} {2/2}

```cpp
void Fl_Browser_::scrollbar_width (int width) [inline]
```

Sets the global \code{Fl::scrollbar_size()}, and forces this instance of the widget to use it.

**Deprecated** Use \code{scrollbar_size()} instead.

**Todo** This method should eventually be removed in 1.4+

### 32.8.4.41 \code{select()}

```cpp
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.

### 32.8.4.42 \code{select_only()}

```cpp
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>
32.8.4.43 selection()

```cpp
typedef 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.

32.8.4.44 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>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FL_SORT_ASCENDING – sort in ascending order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL_SORT_DESCENDING – sort in descending order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other values may appear in the future</td>
</tr>
</tbody>
</table>

**Todo** Add a flag to ignore case

32.8.4.45 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**

<table>
<thead>
<tr>
<th>in</th>
<th><code>a</code>,<code>b</code></th>
<th>Items being swapped</th>
</tr>
</thead>
</table>

32.8.4.46 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()

### 32.8.5 Member Data Documentation

32.8.5.1 hscrollbar

```cpp
Fl.Scrollbar Fl_Browser_::hscrollbar
```

Horizontal scrollbar.
Public, so that it can be accessed directly.

Generated by Doxygen
32.8.5.2 scrollbar

Fl_Scrollbar Fl_Browser_::scrollbar
Vertical scrollbar.
Public, so that it can be accessed directly.
Use scrollbar_left() or scrollbar_right() to change what side the vertical scrollbar is drawn on.
Use scrollbar.align(int) (see Fl_Widget::align(Fl_Align)) to change what side either of the scrollbars is drawn on.
If the FL_ALIGN_LEFT bit is on, the vertical scrollbar is on the left. If the FL_ALIGN_TOP bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.
The documentation for this class was generated from the following files:

- Fl_Browser_.H
- Fl_Browser_.cxx

32.9 Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

#include <Fl_Button.H>

Inheritance diagram for Fl_Button:

Public Member Functions

- int clear ()
  *Same as value(0).*
- Fl_Boxtype down_box () const
  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)
  *Sets the down box type.*
- Fl_Color down_color () const
  (for backwards compatibility)
- void down_color (unsigned c)
  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char ∗L=0)
  The constructor creates the button using the given position, size, and label.
- virtual int handle (int)
  Handles the specified event.
- int set ()
  *Same as value(1).*
- void setonly ()
  Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this).
- int shortcut () const
  Returns the current shortcut key for the button.
- void shortcut (const char ∗s)
  (for backwards compatibility)
• void shortcut (int s)
  
  Sets the shortcut key to s.
• char value () const
  
  Returns the current value of the button (0 or 1).
• int value (int v)
  
  Sets the current value of the button.

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

Additional Inherited Members

32.9.1 Detailed Description
Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when(). Buttons can also generate callbacks in response to FL_SHORTCUT events. The button can either have an explicit shortcut(int s) value or a letter shortcut can be indicated in the label() with an ‘&’ character before it. For the label shortcut it does not matter if Alt is held down, but if you have an input field in the same window, the user will have to hold down the Alt key so that the input field does not eat the event first as an FL_KEYBOARD event.

See also
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).

32.9.2 Constructor & Destructor Documentation
### 32.9.2.1 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**

- `in X,Y,W,H`: position and size of the widget
- `in L`: widget label, default is no label

### 32.9.3 Member Function Documentation

#### 32.9.3.1 clear()

```cpp
int Fl_Button::clear ( ) [inline]
```

Same as `value(0)`.

See also

`value(int v)`

#### 32.9.3.2 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`

#### 32.9.3.3 down_box() [2/2]

```cpp
void Fl_Button::down_box (  
    Fl_Boxtype b ) [inline]
```

Sets the down box type. The default value of 0 causes FLTK to figure out the correct matching down version of `box()`. Some derived classes (e.g. `Fl_Round_Button` and `Fl_Light_Button`) use `down_box()` for special purposes. See docs of these classes.

**Parameters**

- `in b`: down box type
32.9.3.4 draw()

```cpp
doctor 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_Return_Button, and Fl_Light_Button.

32.9.3.5 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

<table>
<thead>
<tr>
<th><strong>event</strong></th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th><strong>value</strong></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_Return_Button, Fl_Repeat_Button, and Fl_Light_Button.

32.9.3.6 set()

```cpp
int Fl_Button::set ( ) [inline]
```

Same as ` value(1) `. 

See also

` value(int v) `

32.9.3.7 shortcut() [1/2]

```cpp
int Fl_Button::shortcut ( ) const [inline]
```

Returns the current shortcut key for the button.
Return values

32.9.3.8 shortcut() [2/2]

void Fl_Button::shortcut (  
    int s ) [inline]

Sets the shortcut key to s.
Setting this overrides the use of ‘&’ in the label(). The value is a bitwise OR of a key and a set of shift flags, for example: FL_ALT | 'a', or FL_ALT | (FL_F + 10), or just 'a'. A value of 0 disables the shortcut. The key can be any value returned by Fl::event_key(), but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down. The shift flags can be any set of values accepted by Fl::event_state(). If the bit is on, that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & s \\
\text{bitwise OR of key and shift flags} \\
\hline
\end{array}
\]

32.9.3.9 value()

int Fl_Button::value (  
    int v )

Sets the current value of the button. A non-zero value sets the button to 1 (ON), and zero sets it to 0 (OFF).

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & v \\
\text{button value.} \\
\hline
\end{array}
\]

See also

set(), clear()

The documentation for this class was generated from the following files:

- Fl_Button.H
- Fl_Button.cxx

32.10 Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.

#include <Fl_Cairo.H>

Public Member Functions

- bool autolink () const  
  
  \textit{Gets the autolink option. See Fl::cairo_autolink_context(bool)}

- void autolink (bool b)
  
  \textit{Sets the autolink option, only available with –enable-cairoext.}

- cairo_t * cc () const
  
  \textit{Generated by Doxygen}
32.11 Fl_Cairo_Window Class Reference

```
Gets the current cairo context.
• void cc (cairo_t *c, bool own=true)
  Sets the current cairo context.
• void *gc () const
  Gets the last gc attached to a cc.
• void gc (void *c)
  Sets the gc c to keep track on.
• void *window () const
  Gets the last window attached to a cc.
• void window (void *w)
  Sets the window w to keep track on.
```

### 32.10.1 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` or `--enable-cairoext` option or one or both of the CMake options `OPTION_CAIRO` or `OPTION_CAIROEXT` is set (ON)

### 32.10.2 Member Function Documentation

#### 32.10.2.1 cc()

```cpp
def 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

### 32.11 Fl_Cairo_Window Class Reference

This defines a FLTK window with cairo support.

```cpp
#include <Fl_Cairo_Window.H>
```

Inheritance diagram for Fl_Cairo_Window:

Generated by Doxygen
Public Types

- typedef void(\* cairo_draw_cb) (Fl_Cairo_Window \*self, cairo_t \*def)

  This defines the cairo draw callback prototype that you must further.

Public Member Functions

- Fl_Cairo_Window(int W, int H, 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

32.11.1 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.

The default coordinate system for cairo drawing commands within Fl_Cairo_Window is FLTK's coordinate system, where the \(x, y, w, h\) values are relative to the top/left corner of the Fl_Cairo_Window, as one would expect with regular FLTK drawing commands, e.g.:

\[
0 \leq x \leq w-1, \quad 0 \leq y \leq h-1
\]

Example:

```c
static void my_cairo_draw_cb(Fl_Cairo_Window \*window, cairo_t \*cr) {
    // Draw an "X"
    const double xmax = (window->w() - 1);
    const double ymax = (window->h() - 1);
    cairo_set_line_width(cr, 1.00); // line width for drawing
    cairo_set_source_rgb(cr, 1.0, 0.5, 0.0); // orange
    cairo_move_to(cr, 0.0, 0.0); cairo_line_to(cr, xmax, ymax); // draw diagonal "/"
    cairo_move_to(cr, 0.0, ymax); cairo_line_to(cr, xmax, 0.0); // draw diagonal "\"
    cairo_stroke(cr); // stroke the lines
}
```

The FLTK coordinate system differs from the default native cairo coordinate system which uses normalized \(0.0 \leftrightarrow 0...1.0\) values for \(x\) and \(y\), e.g.:

\[
0 \leq x \leq 1.0, \quad 0 \leq y \leq 1.0
\]

So beware of this when copy/pasting cairo example programs that assume normalized values. If need be, you can revert to the cairo coordinate system by simply calling cairo_scale() with the widget's \(w()\) and \(h()\) values. Example:

```c
static void my_cairo_draw_cb(Fl_Cairo_Window \*window, cairo_t \*cr) {
    cairo_scale(cr, window->w(), window->h()); // use cairo's default coordinate system
    ..use 0.0 to 1.0 values from here on..
}
```
32.12 Fl_Chart Class Reference

See also
examples/cairo-draw-x.cxx
test/cairo_test.cxx

Note

Class Fl_Cairo_Window requires the FLTK library to have been built with CMake option OPTION_CAIRO or
configure --enable-cairo.

You can alternatively define your custom cairo FLTK window, and thus at least override the draw() method
to provide custom cairo support. In this case you will probably use Fl::cairo_make_current(Fl_Window*) to
attach a context to your window. You should do it only when your window is the current window.

See also
Fl_Window::current()

32.11.2 Member Function Documentation

32.11.2.1 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

32.12 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)**
  
  Sets the chart's text color to n.

- **Fl_Font textfont () const**
  
  Gets the chart's text font.

- **void textfont (Fl_Font s)**
  
  Sets the chart's text font to s.

- **Fl_Fontsize textsize () const**
  
  Gets the chart's text size.

- **void textsize (Fl_Fontsize s)**
  
  Sets the chart's text size to s.

- **~Fl_Chart ()**

  Destroys the Fl_Chart widget and all of its data.

Protected Member Functions

- **void draw ()**

  Draws the widget.

Additional Inherited Members

### 32.12.1 Detailed Description

**FL_Chart** displays simple charts.

It is provided for Forms compatibility.

![Figure 32.4 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.

### 32.12.2 Constructor & Destructor Documentation

#### 32.12.2.1 Fl_Chart()

```cpp
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in X,Y,W,H</td>
<td>position and size of the widget</td>
</tr>
<tr>
<td>in L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

### 32.12.3 Member Function Documentation

#### 32.12.3.1 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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in val</td>
<td>data value</td>
</tr>
<tr>
<td>in str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>
32.12.3.2 autosize() [1/2]

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.

32.12.3.3 autosize() [2/2]

void Fl_Chart::autosize ( uchar n ) [inline]
Set whether the chart will automatically adjust the bounds of the chart.
Parameters
   in  n  non-zero to enable automatic resizing, zero to disable.

32.12.3.4 bounds() [1/2]

void Fl_Chart::bounds ( double ∗ a, double ∗ b ) const [inline]
Gets the lower and upper bounds of the chart values.
Parameters
   out  a,b  are set to lower, upper

32.12.3.5 bounds() [2/2]

void Fl_Chart::bounds ( double a, double b )
Sets the lower and upper bounds of the chart values.
Parameters
   in  a,b  are used to set lower, upper

32.12.3.6 draw()

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:

Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
32.12.3.7 insert()

```c
void Fl_Chart::insert (
    int ind,
    double val,
    const char * str = 0,
    unsigned col = 0 )
```

Inserts a data value `val` at the given position `ind`. Position 1 is the first data value.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ind</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>data value</td>
</tr>
<tr>
<td>in</td>
<td>str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

32.12.3.8 maxsize()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>m</th>
<th>maximum number of data values allowed</th>
</tr>
</thead>
</table>

32.12.3.9 replace()

```c
void Fl_Chart::replace ( int ind,
    double val,
    const char * str = 0,
    unsigned col = 0 )
```

Replace a data value `val` at the given position `ind`. Position 1 is the first data value.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ind</th>
<th>insertion position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>val</td>
<td>data value</td>
</tr>
<tr>
<td>in</td>
<td>str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Chart.H
- Fl_Chart.cxx
32.13  FL_CHART_ENTRY Struct Reference

For internal use only.
#include <Fl_Chart.H>

Public Attributes

• unsigned col
  For internal use only.
• char str [FL_CHART_LABEL_MAX+1]
  For internal use only.
• float val
  For internal use only.

32.13.1 Detailed Description

For internal use only.
The documentation for this struct was generated from the following file:
• Fl_Chart.H

32.14  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.
#include <Fl_Check_Browser.H>

Inheritance diagram for Fl_Check_Browser:

```
  Fl_Widget
    Fl_Group
      Fl_Browser_
        Fl_Check_Browser
```

Public Member Functions

• int add (char *s)
  Add a new unchecked line to the end of the browser.
• int add (char *s, int b)
  Add a new line to the end of the browser.
• int add (const char *s)
  See int Fl_Check_Browser::add(char *s)
• int add (const char *s, int b)
  See int Fl_Check_Browser::add(char *s)
• void check_all ()
  Sets all the items checked.
• void check_none ()
  Sets all the items unchecked.
• int checked (int item) const
  Gets the current status of item item.
• void **checked** (int item, int b)
  Sets the check status of item item to b.
• void **clear** ()
  Remove every item from the browser.
• **Fl_Check_Browser** (int x, int y, int w, int h, const char∗ l=0)
  The constructor makes an empty browser.
• 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

32.14.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

32.14.2 Member Function Documentation

32.14.2.1 add() [1/2]

```cpp
int Fl_Check_Browser::add (char ∗s)
```

Add a new unchecked line to the end of the browser.

See also

```cpp
add(char ∗s, int b)
```

32.14.2.2 add() [2/2]

```cpp
int Fl_Check_Browser::add (char ∗s, int b)
```

Add a new line to the end of the browser. The text is copied using the strdup() function. It may also be NULL to make a blank line. It can set the item checked if b is not 0.

32.14.2.3 handle()

```cpp
int Fl_Check_Browser::handle (int event) [protected], [virtual]
```

Handles the event within the normal widget bounding box.

Parameters

| in | event | The event to process. |

Returns

1 if event was processed, 0 if not.

Reimplemented from Fl_Browser_.

32.14.2.4 item_at()

```cpp
void ∗Fl_Check_Browser::item_at (int index) const [virtual]
```

This method must be provided by the subclass to return the item for the specified index.

Parameters

| in | index | The index of the item to be returned |

Generated by Doxygen
Returns

The item at the specified index.

Reimplemented from Fl_Browser_.

32.14.2.5 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_.

32.14.2.6 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_.

32.14.2.7 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_.

32.14.2.8 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_.

Generated by Doxygen
32.14.2.9 item_select()

```cpp
template<class Fl_Browser> void Fl_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_.

32.14.2.10 item_selected()

```cpp
template<class Fl_Browser> int Fl_Browser::item_selected (void * item) const [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for item.

The method should return 1 if item is selected, or 0 otherwise.

**Parameters**

| in | item | The item to test. |

Reimplemented from Fl_Browser_.

32.14.2.11 item_swap()

```cpp
template<class Fl_Browser> void Fl_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_.

32.14.2.12 item_text()

```cpp
template<class Fl_Browser> const char * Fl_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)

Reimplemented from Fl_Browser_.

### 32.14.2.13 item_width()

```cpp
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**

| in | item | The item whose width is returned. |

**Returns**

The width of the item in pixels.

Implements Fl_Browser_.

### 32.14.2.14 nitems()

```cpp
int Fl_Check_Browser::nitems ( ) const [inline]
```

Returns how many lines are in the browser. The last line number is equal to this.

### 32.14.2.15 remove()

```cpp
int Fl_Check_Browser::remove (int item)
```

Remove line n and make the browser one line shorter. Returns the number of lines left in the browser.

The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

### 32.15 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

32.15.1 Detailed Description
A button with a "checkmark" to show its status.

![Fl_Check_Button](image)

Figure 32.5 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.

32.15.2 Constructor & Destructor Documentation

32.15.2.1 Fl_Check_Button()

**Fl_Check_Button::Fl_Check_Button** (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗L = 0)

Creates a new **Fl_Check_Button** widget using the given position, size, and label string.
The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark.
The **selection_color()** sets the color of the checkmark. Default is FL_FOREGROUND_COLOR (usually black).
You can use **down_box()** to change the box type of the checkmark. Default is FL_DOWN_BOX.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- **Fl_Check_Button.H**
- **Fl_Check_Button.cxx**

32.16 **Fl_Choice Class Reference**

A button that is used to pop up a menu.

```c
#include <Fl_Choice.H>
```

Inheritance diagram for Fl_Choice:
**Public Member Functions**

- **Fl_Choice** (int X, int Y, int W, int H, const char ∗L=0)
  
  Create a new Fl_Choice widget using the given position, size and label string.

- int **handle** (int)
  
  Handles the specified event.

- int **value** () const
  
  Gets the index of the last item chosen by the user.

- int **value** (const Fl_Menu_Item ∗v)
  
  Sets the currently selected value using a pointer to menu item.

- int **value** (int v)
  
  Sets the currently selected value using the index into the menu item array.

**Protected Member Functions**

- void **draw** ()
  
  Draws the widget.

**Additional Inherited Members**

**32.16.1 Detailed Description**

A button that is used to pop up a menu.

This is a button that, when pushed, pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects. Motif calls this an OptionButton.

The only difference between this and a Fl_Menu_Button is that the name of the most recent chosen menu item is displayed inside the box, while the label is displayed outside the box. However, since the use of this is most often to control a single variable rather than do individual callbacks, some of the Fl_Menu_Button methods are redescribed here in those terms.

When the user clicks a menu item, value() is set to that item and then:

- The item’s callback is done if one has been set; the Fl_Choice is passed as the Fl_Widget* argument, along with any userdata configured for the callback.
- If the item does not have a callback, the Fl_Choice widget’s
  callback is done instead, along with any userdata configured
  for it. The callback can determine which item was picked using
  value(), mvalue(), item_pathname(), etc.

All three mouse buttons pop up the menu. The Forms behavior of the first two buttons to increment/decrement the
choice is not implemented. This could be added with a subclass, however.
The menu will also pop up in response to shortcuts indicated by putting a ‘&’ character in the label(). See
Fl_Button::shortcut(int s) for a description of this.
Typing the shortcut() of any of the items will do exactly the same as when you pick the item with the mouse. The ‘&’
character in item names are only looked at when the menu is popped up, however.

Todo Refactor the doxygen comments for Fl_Choice changed() documentation.

- int Fl_Widget::changed() const This value is true the user picks a different value. It is turned
  off by value() and just before doing a callback (the callback can turn it back on if desired).
- void Fl_Widget::set_changed() This method sets the changed() flag.
- void Fl_Widget::clear_changed() This method clears the changed() flag.
- Fl_Boxtype Fl_Choice::down_box() const Gets the current down box, which is used when the
  menu is popped up. The default down box type is FL_DOWN_BOX.
- void Fl_Choice::down_box(Fl_Boxtype b) Sets the current down box type to b.

Simple example:

```cpp
#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();
}
```

### 32.16.2 Constructor & Destructor Documentation

#### 32.16.2.1 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

<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>
## 32.16.3 Member Function Documentation

### 32.16.3.1 draw()

```c++
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`.

### 32.16.3.2 handle()

```c++
int Fl_Choice::handle(int event) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

| in event | the kind of event received |

**Return values**

- `0` if the event was not used or understood
- `1` if the event was used and can be deleted

**See also**

`Fl_Event`

Reimplemented from `Fl_Widget`.

### 32.16.3.3 value() [1/3]

```c++
int Fl_Choice::value() const [inline]
```

Gets the index of the last item chosen by the user. The index is -1 initially.

### 32.16.3.4 value() [2/3]

```c++
int Fl_Choice::value(const Fl_Menu_Item *v)
```

Sets the currently selected value using a pointer to menu item. Changing the selected value causes a `redraw()`.

**Parameters**

| in v | pointer to menu item in the menu item array |

Generated by Doxygen
32.16.3.5 value() [3/3]

int Fl_Choice::value (int v)

Sets the currently selected value using the index into the menu item array. Changing the selected value causes a `redraw()`.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>v</td>
<td>index of value in the menu item array.</td>
</tr>
</tbody>
</table>

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

32.17 Fl_Clock Class Reference

This widget provides a round analog clock display.

```plaintext
#include <Fl_Clock.H>
```

Inheritance diagram for Fl_Clock:

![Inheritance Diagram](image)

**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

32.17.1 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 32.7 FL_SQUARE_CLOCK type](image)

![Figure 32.8 FL_ROUND_CLOCK type](image)

See also class **Fl_Clock_Output**

32.17.2 Constructor & Destructor Documentation

32.17.2.1 **Fl_Clock()** [1/2]

```cpp
Fl_Clock::Fl_Clock ( int X, int Y, int W, int H, const char * L = 0 )
```

Create an **Fl_Clock** widget using the given position, size, and label string. The default clock type is FL_SQUARE_CLOCK and the default boxtype is FL_UP_BOX.

Parameters

| in | X,Y,W,H | position and size of the widget |
| in | L       | widget label, default is no label |
32.17.2 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>

See also

class Fl_Clock_Output

32.17.3 Member Function Documentation

32.17.3.1 handle()

int Fl_Clock::handle (int event) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in   | event | the kind of event received |

Return values

| 0    | if the event was not used or understood |

| 1    | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

• Fl_Clock.H
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.

- `ulong value () const`
  Returns the displayed time.

- `void value (int H, int m, int s)`
  Set the displayed time.

- `void value (ulong v)`
  Set the displayed time.

**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**

**32.18.1 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.
Figure 32.9 FL_SQUARE_CLOCK type

Figure 32.10 FL_ROUND_CLOCK type

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

32.18.2 Constructor & Destructor Documentation

32.18.2.1 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 |

32.18.3 Member Function Documentation
32.18.3.1 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

32.18.3.2 hour()

```cpp
int Fl_Clock_Output::hour ( ) const [inline]
```
Returns the displayed hour (0 to 23).

**See also**
- value(), minute(), second()

32.18.3.3 minute()

```cpp
int Fl_Clock_Output::minute ( ) const [inline]
```
Returns the displayed minute (0 to 59).

**See also**
- value(), hour(), second()

32.18.3.4 second()

```cpp
int Fl_Clock_Output::second ( ) const [inline]
```
Returns the displayed second (0 to 60, 60=leap second).

**See also**
- value(), hour(), minute()

32.18.3.5 shadow() [1/2]

```cpp
int Fl_Clock_Output::shadow ( ) const [inline]
```
Returns the shadow drawing mode of the hands.

**Returns**
- shadow drawing mode of the hands

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</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>
32.18.3.6 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 |

32.18.3.7 value() [1/3]

ulong Fl_Clock_Output::value ( ) const [inline]

Returns the displayed time.
Returns the time in seconds since the UNIX epoch (January 1, 1970).
See also

    value(ulong)

32.18.3.8 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()

32.18.3.9 value() [3/3]

void Fl_Clock_Output::value (  
    ulong v )

Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).

Parameters

| in  | v | seconds since epoch |
The Fl_Color_Chooser widget provides a standard RGB color chooser.

#include <Fl_Color_Chooser.H>

Inheritance diagram for Fl_Color_Chooser:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Color_Chooser
```

Public Member Functions

- double b () const
  
  Returns the current blue value.

- Fl_Color_Chooser (int X, int Y, int W, int H, const char ∗L=0)
  
  Creates a new Fl_Color_Chooser widget using the given position, size, and label string.

- double g () const
  
  Returns the current green value.

- int handle (int e)
  
  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.

See also

value()
### 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

#### 32.19.1 Detailed Description

The **Fl_Color_Chooser** widget provides a standard RGB color chooser.

![Image](fl_color_chooser.png)

**Figure 32.11 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.

#### 32.19.2 Constructor & Destructor Documentation

#### 32.19.2.1 Fl_Color_Chooser()

**Fl_Color_Chooser**: Fl_Color_Chooser {

- int X,
- int Y,
- int W,
- int H,
- const char * L = 0
}
Creates a new Fl_Color_Chooser widget using the given position, size, and label string.
The recommended dimensions are 200x95. The color is initialized to black.

Parameters

| in | X,Y,W,H | position and size of the widget |
| in | L     | widget label, default is no label |

### 32.19.3 Member Function Documentation

#### 32.19.3.1 b()

```cpp
double Fl_Color_Chooser::b ( ) const [inline]
```

Returns the current blue value.

\[ 0 < b < 1 \]

#### 32.19.3.2 g()

```cpp
double Fl_Color_Chooser::g ( ) const [inline]
```

Returns the current green value.

\[ 0 < g < 1 \]

#### 32.19.3.3 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

| in | e | current event |

Returns

1 if event has been handled, 0 otherwise
### 32.19.3.4 hsv()

```c
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.

### 32.19.3.5 hsv2rgb()

```c
void Fl_Color_Chooser::hsv2rgb (  
    double H,  
    double S,  
    double V,  
    double & R,  
    double & G,  
    double & B  
)  
```

This static method converts HSV colors to RGB colorspace.

**Parameters**

- **in** $H,S,V$ color components
- **out** $R,G,B$ color components

### 32.19.3.6 hue()

```c
double Fl_Color_Chooser::hue ( ) const  
```

Returns the current hue.

$0 \leq \text{hue} < 6$. 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.*

### 32.19.3.7 mode [1/2]

```c
int Fl_Color_Chooser::mode ( )  
```

Returns which Fl_Color_Chooser variant is currently active.

**Returns**

- color modes are rgb(0), byte(1), hex(2), or hsv(3)
32.19.3.8 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) |

32.19.3.9 r()

```cpp
double Fl_Color_Chooser::r () const [inline]
```

Returns the current red value.

0 \leq r \leq 1.

32.19.3.10 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.

32.19.3.11 rgb2hsv()

```cpp
void Fl_Color_Chooser::rgb2hsv (double R, double G, double B, double & H, double & S, double & V) [static]
```

This static method converts RGB colors to HSV colorspace.

**Parameters**

| in | R,G,B | color components |
| out | H,S,V | color components |
32.19.3.12 saturation()

```cpp
double Fl_Color_Chooser::saturation ( ) const [inline]
```

Returns the saturation.

0 \leq \text{saturation} \leq 1.

32.19.3.13 value()

```cpp
double Fl_Color_Chooser::value ( ) const [inline]
```

Returns the value/brightness.

0 \leq \text{value} \leq 1.

The documentation for this class was generated from the following files:

- Fl_Color_Chooser.H
- Fl_Color_Chooser.cxx

32.20 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

```cpp
#include <Fl_Copy_Surface.H>
```

Inheritance diagram for Fl_Copy_Surface:

```
Fl_Surface_Device
    `-- Fl_Widget_Surface
        `-- Fl_Copy_Surface
```

Public Member Functions

- 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

32.20.1 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 *g = ...; // a widget you want to copy to the clipboard
Fl_Copy_Surface *copy_surf = new Fl_Copy_Surface(g->w(), g->h()); // create an Fl_Copy_Surface object
Fl_Surface_Device::push_current(copy_surf); // direct graphics requests to the clipboard
fl_color(FL_WHITE); fl_rectf(0, 0, g->w(), g->h()); // draw a white background
copy_surf->draw(g); // draw the g widget in the clipboard
Fl_Surface_Device::pop_current(); // direct graphics requests back to their previous destination
delete copy_surf; // after this, the clipboard is loaded
```

Platform details:

- Windows: Transparent RGB images copy without transparency. The graphical data are copied to the clipboard in two formats: 1) as an 'enhanced metafile'; 2) as a color bitmap. Applications to which the clipboard content can be pasted 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 and Wayland: the graphical data are copied to the clipboard as an image in BMP format.

32.20.2 Constructor & Destructor Documentation

32.20.2.1 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 |

32.20.3 Member Function Documentation

32.20.3.1 origin() [1/2]

void Fl_Copy_Surface::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>
</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.

32.20.3.2 origin() [2/2]

void Fl_Copy_Surface::origin {
  int x,
  int y) [virtual]

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, printable_rect(&w, &h); origin(w/2, 0); sets the graphics origin at the top center of the drawable area. Successive origin() calls don’t combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

| in | x, y |
| Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

32.20.3.3 printable_rect()

int Fl_Copy_Surface::printable_rect {
  int * w,
  int * h) [virtual]

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

32.20.3.4 set_current()

void Fl_Copy_Surface::set_current {
  void ) [virtual]

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Note

It’s recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
• draw to that surface;
• make the display current again with Fl_Display_Device::display_device()->set_current();. Don’t do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current().

Reimplemented from Fl_Surface_Device.

32.20.3.5 translate()

void Fl_Copy_Surface::translate (  
    int x,
    int y ) [protected], [virtual]  

Translates the current graphics origin accounting for the current rotation. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects. Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following files:

• Fl_Copy_Surface.H
• Fl_Copy_Surface.cxx

32.21 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.

#include <Fl_Counter.H>

Inheritance diagram for Fl_Counter:

```
Fl_Widget
   
Fl_Valuator
   
Fl_Counter
   
Fl_Simple_Counter
```

Public Member Functions

• Fl_Counter (int X, int Y, int W, int H, const char ∗L=0)
  
  Creates a new Fl_Counter widget using the given position, size, and label string.

• int handle (int)
  
  Handles the specified event.

• void lstep (double a)
  
  Sets the increment for the large step buttons.

• double step () const
  
  Returns the increment for normal step buttons.

• void step (double a)
  
  Sets the increment for the normal step buttons.

• void step (double a, double b)
  
  Sets the increments for the normal and large step buttons.

• Fl_Color textcolor () const
  
  Gets the font color.
• void textcolor (Fl_Color s)
  
  Sets the font color to s.

• Fl_Font textfont () const
  
  Gets the text font.

• void textfont (Fl_Font s)
  
  Sets the text font to s.

• Fl_Fontsize textsize () const
  
  Gets the font size.

• void textsize (Fl_Fontsize s)
  
  Sets the font size to s.

• ~Fl_Counter ()
  
  Destroys the valuator.

Protected Member Functions

• void draw ()
  
  Draws the widget.

Additional Inherited Members

32.21.1 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.

![Fl_Counter](image)

Figure 32.12 Fl_Counter

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.

32.21.2 Constructor & Destructor Documentation

32.21.2.1 Fl_Counter()

Fl_Counter::Fl_Counter {
  int X,
  int Y,
  int W,
  int H,
  const char * L = 0
}

Creates a new Fl_Counter widget using the given position, size, and label string.
The default type is FL_NORMAL_COUNTER.
Parameters

<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>

### 32.21.3 Member Function Documentation

#### 32.21.3.1 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`.

#### 32.21.3.2 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

| 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`.

#### 32.21.3.3 lstep()

```cpp
void Fl_Counter::lstep (double a) [inline]
```

Sets the increment for the large step buttons.

The default value is 1.0.
32.21.3.4 step() [1/2]

void Fl_Counter::step (double a) [inline]

Sets the increment for the normal step buttons.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in a</td>
<td>large step increment.</td>
</tr>
</tbody>
</table>

32.21.3.5 step() [2/2]

void Fl_Counter::step (double a, double b) [inline]

Sets the increments for the normal and large step buttons.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in a,b</td>
<td>normal and large step increments.</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Counter.H
- Fl_Counter.cxx

32.22 Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Device_Plugin:

```
Fl_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.

### 32.22.1 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.

### 32.22.2 Member Function Documentation

#### 32.22.2.1 rectangle_capture()

virtual Fl_RGB_Image * Fl_Device_Plugin::rectangle_capture ( 
  Fl_Widget * widget, 
  int x, 
  int y, 
  int w, 
  int h ) [pure virtual]

Captures a rectangle of a widget as an image.

Returns

- The captured pixels as an RGB image

The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

### 32.23 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)
  
  Allows subclasses to handle event based on current position and size.

Protected Member Functions

- void draw ()
  
  Draws dial at current position and size.

- void draw (int X, int Y, int W, int H)
  
  Draws dial at given position and size.

- int handle (int event, int X, int Y, int W, int H)
  
  Allows subclasses to handle event based on given position and size.

Additional Inherited Members

32.23.1 Detailed Description

The Fl_Dial widget provides a circular dial to control a single floating point value.

![Figure 32.13 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.

32.23.2 Constructor & Destructor Documentation
32.23.1 Fl_Dial()

Fl_Dial::Fl_Dial (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * l = 0 )

Creates a new Fl_Dial widget using the given position, size, and label string.
The default type is FL_NORMAL_DIAL.

32.23.3 Member Function Documentation

32.23.3.1 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.

32.23.3.2 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 |

32.23.3.3 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

32.24 Fl_Display_Device Class Reference

A display to which the computer can draw.
#include &lt;Fl_Device.H&gt;

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**

### 32.24.1 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

### 32.25 Fl_Double_Window Class Reference

The Fl_Double_Window provides a double-buffered window.

```c
#include &lt;Fl_Double_Window.H&gt;
```

Inheritance diagram for Fl_Double_Window:

```
Fl_Widget
   ^
   |   Fl_Group
       |   Fl_Window
           |   Fl_Double_Window
                   |
                   |   Fl_Cairo_Window
                   |   Fl_Overlay_Window
```

**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)
  
  Same as Fl_Window::show(int a, char **b)

• Fl_Double_Window ()
  
  The destructor also deletes all the children.

### Additional Inherited Members

#### 32.25.1 Detailed Description

The Fl_Double_Window provides a double-buffered window. It will draw the window data into an off-screen pixmap, and then copy it to the on-screen window.

#### 32.25.2 Constructor & Destructor Documentation

32.25.2.1 Fl_Double_Window()

Fl_Double_Window::Fl_Double_Window ( )

The destructor also deletes all the children.

This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.

32.25.3 Member Function Documentation

32.25.3.1 hide()

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.

32.25.3.2 resize()

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.

32.25.3.3 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

32.26 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()*

32.26.1 Detailed Description

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:
class MyClass { 
  Fl_Group group;
  Fl_Button button_in_group;
  Fl_End end;
  Fl_Button button_outside_group;
  MyClass();
};

MyClass::MyClass() :
group(10,10,100,100),
button_in_group(20,20,60,30),
end(),
button_outside_group(10,120,60,30) {
  [.ctor code..]
}

The documentation for this class was generated from the following file:

- Fl_Group.H

Generated by Doxygen
32.27  Fl_EPS_File_Surface Class Reference

Encapsulated PostScript drawing surface.

#include <Fl_PostScript.H>

Inheritance diagram for Fl_EPS_File_Surface:

Public Member Functions

- int close ()
  Completes all EPS output.
- FILE * file ()
  Returns the underlying FILE pointer.
- Fl_EPS_File_Surface (int width, int height, FILE *eps_output, Fl_Color background=FL_WHITE, Fl_PostScript_Close_Command closef=NULL)
  Constructor.
- virtual void origin (int *px, int *py)
  Computes the coordinates of the current origin of graphics functions.
- virtual void origin (int x, int y)
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- 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

32.27.1  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:

Fl_Window *win = ... // Window to draw to an .eps file
int ww = win->decorated_w();
inw = win->decorated_h();
FILE *eps = fl_fopen("/path/to/mywindow.eps", "w");
if (eps) {
  Fl_EPS_File_Surface *surface = new Fl_EPS_File_Surface(ww, wh, eps, win->color());
  Fl_Surface_Device::push_current(surface);
  surface->draw_decorated_window(win);
  Fl_Surface_Device::pop_current();
  delete surface;  // the .eps file is not complete until the destructor was run

Generated by Doxygen
32.27.2 Constructor & Destructor Documentation

32.27.2.1 Fl_EPS_File_Surface()

```cpp
Fl_EPS_File_Surface::Fl_EPS_File_Surface (  
    int width,  
    int height,  
    FILE * eps_output,  
    Fl_Color background = FL_WHITE,  
    Fl_PostScript_Close_Command closef = NULL )
```

Constructor.

### Parameters

<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_output</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. Under the X11 + pango platform, transparent RGB images are correctly blended to their background, thus this parameter has no effect.</td>
</tr>
<tr>
<td>closef</td>
<td>If not NULL, the destructor or close() will call closef(eps_output) after all EPS data has been sent. If NULL, fclose(eps_output) is called instead. This allows to close the FILE pointer by, e.g., pclose, or, using a function such as &quot;int keep_open(FILE*){return 0; }&quot;, to keep it open after completion of all output to eps_output. Function closef should return non zero to indicate an error.</td>
</tr>
</tbody>
</table>

32.27.2.2 ~Fl_EPS_File_Surface()

```cpp
Fl_EPS_File_Surface::~Fl_EPS_File_Surface ( )
```

Destructor.

By default, the destructor closes with function fclose() the underlying FILE. See the constructor for how to close it differently or to keep it open. Use close() before object destruction to receive the status code of output operations. If close() is not used and if EPS output results in error, the destructor displays an alert message with fl_alert().

32.27.3 Member Function Documentation

32.27.3.1 close()

```cpp
int Fl_EPS_File_Surface::close ( )
```

Completes all EPS output.

The only operation possible with the Fl_EPS_File_Surface object after calling close() is its destruction.

Returns

The status code of output operations to the FILE object. 0 indicates success.

32.27.3.2 origin() [1/2]

```cpp
virtual void Fl_EPS_File_Surface::origin (  
    int * x,  
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.
### 32.27.3.3 origin() [2/2]

```cpp
class Fl_EPS_File_Surface:
virtual void 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`.

### 32.27.3.4 printable_rect()

```cpp
class Fl_EPS_File_Surface:
virtual int 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`.

### 32.27.3.5 translate()

```cpp
class Fl_EPS_File_Surface:
virtual void 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`

## 32.28 Fl_File_Browser Class Reference

The `Fl_File_Browser` widget displays a list of filenames, optionally with file-specific icons.

```cpp
#include <Fl_File_Browser.H>
```

Inheritance diagram for `Fl_File_Browser`:

```
Generated by Doxygen
```
Public Types

- enum { FILES, DIRECTORIES }

Public Member Functions

- const char * errmsg () const
  
  Returns OS error messages, or NULL if none.

- void errmsg (const char * emsg)
  
  Sets OS error message to a string, which can be NULL.

- int filetype () const
  
  Sets or gets the file browser type, FILES or DIRECTORIES.

- void filetype (int t)
  
  Sets or gets the file browser type, FILES or DIRECTORIES.

- const char * filter () const
  
  Sets or gets the filename filter.

- void filter (const char * pattern)
  
  Sets or gets the filename filter.

- Fl_File_Browser (int, int, int, int, const char * = 0)
  
  The constructor creates the Fl_File_Browser widget at the specified position and size.

- uchar iconsize () const
  
  Sets or gets the size of the icons.

- void iconsize (uchar s)
  
  Sets or gets the size of the icons.

- int load (const char * directory, Fl_File_Sort_F * sort = fl_numericsort)
  
  Loads the specified directory into the browser.

- Fl_Fontsize textsize () const
  
  Sets or gets the size of the icons.

- void textsize (Fl_Fontsize s)

Additional Inherited Members

32.28.1 Detailed Description

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.

32.28.2 Constructor & Destructor Documentation
32.28.2.1 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.

32.28.3 Member Function Documentation

32.28.3.1 errmsg [1/2]

const char* Fl_File_Browser::errmsg ( ) const [inline]

Returns OS error messages, or NULL if none.  
Use when advised.

32.28.3.2 errmsg [2/2]

void Fl_File_Browser::errmsg (  
  const char * emsg )

Sets OS error message to a string, which can be NULL.  
Frees previous if any.  void errmsg(const char *emsg);

32.28.3.3 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.

32.28.3.4 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.

32.28.3.5 filter [1/2]

const char* Fl_File_Browser::filter ( ) const [inline]

Sets or gets the filename filter.  
The pattern matching uses the fl_filename_match() function in FLTK.

32.28.3.6 filter [2/2]

void Fl_File_Browser::filter (  
  const char * pattern )

Sets or gets the filename filter.  
The pattern matching uses the fl_filename_match() function in FLTK.

32.28.3.7 iconsize [1/2]

uchar Fl_File_Browser::iconsize ( ) const [inline]

Sets or gets the size of the icons.  
The default size is 20 pixels.
32.28.3.8 iconsize() [2/2]  
void Fl_File_Browser::iconsize (  
    uchar s ) [inline]  
Sets or gets the size of the icons.  
The default size is 20 pixels.

32.28.3.9 load()  
int Fl_File_Browser::load (  
    const char * directory,  
    Fl_File_Sort_F * sort = fl_numericsort )  
Loads the specified directory into the browser.  
If icons have been loaded then the correct icon is associated with each file in the list.  
If directory is "", all mount points (unix) or drive letters (Windows) are listed.  
The sort argument specifies a sort function to be used with fl_filename_list().  
Return value is the number of filename entries, or 0 if none. On error, 0 is returned, and errmsg() has OS error  
string if non-NULL.  
The documentation for this class was generated from the following files:  
  • FL_File_Browser.H  
  • FL_File_Browser.cxx

32.29 Fl_File_Chooser Class Reference  
The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Public Types  
  • enum { SINGLE = 0 , MULTI = 1 , CREATE = 2 , DIRECTORY = 4 }

Public Member Functions  
  • Fl_Widget * add_extra (Fl_Widget *gr)  
    Adds 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.  
  • Fl_Color color ()  
    Gets the background color of the Fl_File_Browser list.  
  • void color (Fl_Color c)  
    Sets the background color of the Fl_File_Browser list.  
  • int count ()  
    Returns the number of selected files.  
  • char * directory ()  
    Gets the current directory.  
  • void directory (const char *d)  
    Sets the current directory.  
  • const char * filter ()  
    Gets the current filename filter patterns.  
  • void filter (const char *p)  
    Sets the current filename filter patterns.  
  • int filter_value ()  
    Gets the current filename filter selection.  
  • void filter_value (int f)
Sets the current filename filter selection.

- **FLFileChooser** (const char *d, const char *p, int t, const char *title)
  
  The constructor creates the FLFileChooser dialog shown.

- int h () const
  
  Hides the FLFileChooser window.

- uchar iconsize ()
  
  Gets the size of the icons in the FLFileBrowser.

- void iconsize (uchar s)
  
  Sets the size of the icons in the FLFileBrowser.

- const char * label ()
  
  Gets the title bar text for the FLFileChooser.

- void label (const char *l)
  
  Sets the title bar text for the FLFileChooser.

- const char * ok_label ()
  
  Gets the label for the "ok" button in the FLFileChooser.

- void ok_label (const char *l)
  
  Sets the label for the "ok" button in the FLFileChooser.

- void position (int x, int y)

- int preview () const
  
  Returns the current state of the preview box.

- void preview (int e)
  
  Enable or disable the preview tile.

- void rescan ()
  
  Reloads the current directory in the FLFileBrowser.

- void rescan_keep_filename ()
  
  Rescan the current directory without clearing the filename, then select the file if it is in the list.

- void resize (int x, int y, int w, int h)

- void show ()
  
  Shows the FLFileChooser window.

- int shown ()
  
  Returns non-zero if the file chooser main window show() has been called, but not hide().

- void textcolor (int w, int h)

- Fl_Color textcolor ()
  
  Gets the current FLFileBrowser text color.

- void textcolor (Fl_Color c)
  
  Sets the current FLFileBrowser text color.

- Fl_Font textfont ()
  
  Gets the current FLFileBrowser text font.

- void textfont (Fl_Font f)
  
  Sets the current FLFileBrowser text font.

- Fl_Fontsize textsize ()
  
  Gets the current FLFileBrowser text size.

- void textsize (Fl_Fontsize s)
  
  Sets the current FLFileBrowser text size.

- int type ()
  
  Gets the current type of FLFileChooser.

- void type (int t)
  
  Sets the current type of FLFileChooser.

- void * user_data () const
  
  Gets the file chooser user data.
• void **user_data** (void \*d)
  
  Sets the file chooser user data d.
• void **value** (const char \*filename)
  
  Sets the current value of the selected file.
• const char **value** (int f=1)
  
  Gets the current value of the selected file(s).
• int **visible** ()
  
  Returns 1 if the Fl_File_Chooser window is visible.
• int **w** () const
• int **x** () const
• int **y** () const
• \~Fl_File_Chooser ()
  
  Destroys the widget and frees all memory used by it.

Public Attributes

• Fl_Button \* newButton
  
  The "new directory" button is exported so that application developers can control the appearance and use.
• Fl_Check_Button \* previewButton
  
  The "preview" button is exported so that application developers can control the appearance and use.
• Fl_Check_Button \* showHiddenButton
  
  When checked, hidden files (i.e., filename begins with dot) are displayed.

Static Public Attributes

• static const char \* addFavorites_label = "Add to Favorites"
  
  [standard text may be customized at run-time]
• static const char \* allFiles_label = "All Files (*)"
  
  [standard text may be customized at run-time]
• static const char \* customFilter_label = "Custom Filter"
  
  [standard text may be customized at run-time]
• static const char \* existingFile_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 \* manageFavorites_label = "Manage Favorites"
  
  [standard text may be customized at run-time]
• static const char \* newDirectory_label = "New Directory?"
  
  [standard text may be customized at run-time]
• static const char \* newDirectory_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:"

Generated by Doxygen
**Protected Member Functions**

- **void show_error_box (int val)**
  
  Show error box if val=1, hide if val=0.

**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.

### 32.29.1 Detailed Description

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

![Figure 32.14 Fl_File_Chooser](image.png)

**Figure 32.14 Fl_File_Chooser**

Features include:

- Multiple filter patterns can be specified, with parenthesis around filters, and tabs to separate each pattern,
  
e.g.:

  ```
  char pattern[] = "Image Files (*.bmp,gif,jpg,png,xbm,xpm)\t"
  "Web Files (*.htm,html,php)\t"
  "All Files (*)";
  ```
• If no "*" pattern is provided, then an entry for "All Files (*)" is automatically added.

• An optional file preview box is provided which can be toggled by programmer or user showing images, or the first 2048 bytes of printable text.

• Preview image loading functions can be registered to provide custom file previews.

• The favorites button shows up to 100 user-saved favorite directories, the user’s home directory, and a filesystems item.

• A simple dialog is provided for managing saved directories.

• Shortcut keys are provided:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+a</td>
<td>Adds a directory to the favorites list</td>
</tr>
<tr>
<td>Alt+m</td>
<td>Manages the favorites list</td>
</tr>
<tr>
<td>Alt+f</td>
<td>Shows the filesystem list</td>
</tr>
<tr>
<td>Alt+h</td>
<td>Go to the home directory</td>
</tr>
<tr>
<td>Alt+0..9</td>
<td>going to any of the first 10 favorites</td>
</tr>
</tbody>
</table>

The Fl_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.
32.29.2 Constructor & Destructor Documentation

32.29.2.1 Fl_File_Chooser()

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)\t"
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.

32.29.3 Member Function Documentation

32.29.3.1 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.

### 32.29.3.2 filter()

```cpp
void Fl_File_Chooser::filter (const char *p)
```

Sets the current filename filter patterns. The filter patterns use `fl_filename_match()`. Multiple patterns can be used by separating them with tabs, like "\*.jpg\t\*.png\t\*.gif\t\*". In addition, you can provide human-readable labels with the patterns inside parenthesis, like "JPEG Files (*\+.jpg)\tPNG Files (*\+.png)\tGIF Files (*\+.gif)\tAll Files (*)". Use `filter(NULL)` to show all files.

### 32.29.3.3 iconsize() [1/2]

```cpp
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()`.

### 32.29.3.4 iconsize() [2/2]

```cpp
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()`.

### 32.29.3.5 preview()

```cpp
void Fl_File_Chooser::preview (int e)
```

Enable or disable the preview tile. 1 = enable preview, 0 = disable preview.

### 32.29.3.6 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()**

### 32.29.3.7 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);
  ...
}
```
32.29.4 Member Data Documentation

32.29.4.1 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

32.30 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 (float x, float y)
  Adds a vertex value to the icon array, returning a pointer to it.
- short *add_vertex (int x, int y)
  Adds a vertex value to the icon array, returning a pointer to it.
- void clear ()
  Clears all icon data from the icon.
- void draw (int x, int y, int w, int h, Fl_Color ic, int active=1)
  Draws an icon in the indicated area.
- Fl_File_Icon (const char *p, int t, int nd=0, short *d=0)
  Creates a new Fl_File_Icon with the specified information.
- void label (Fl_Widget *w)
  Applies the icon to the widget, registering the Fl_File_Icon label type as needed.
- void load (const char *f)
  Loads the specified icon image.
- int load_fti (const char *fti)
  Loads an SGI icon file.
- int load_image (const char *i)
  Load an image icon file from an image filename.
• Fl_File_Icon * next ()
  Returns next file icon object.
• const char * pattern ()
  Returns the filename matching pattern for the icon.
• int size ()
  Returns the number of words of data used by the icon.
• int type ()
  Returns the filetype associated with the icon, which can be one of the following:
• short * value ()
  Returns the data array for the icon.
• ~Fl_File_Icon ()
  The destructor destroys the icon and frees all memory that has been allocated for it.

Static Public Member Functions
• static Fl_File_Icon * find (const char *filename, int filetype=ANY)
  Finds an icon that matches the given filename and file type.
• static Fl_File_Icon * first ()
  Returns a pointer to the first icon in the list.
• static void labeltype (const Fl_Label *o, int x, int y, int w, int h, Fl_Align a)
  Draw the icon label.
• static void load_system_icons (void)
  Loads all system-defined icons.

32.30.1 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.

32.30.2 Constructor & Destructor Documentation

32.30.2.1 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

<table>
<thead>
<tr>
<th>inout</th>
<th>p</th>
<th>filename pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>inout</td>
<td>t</td>
<td>file type</td>
</tr>
<tr>
<td>inout</td>
<td>nd</td>
<td>number of data values</td>
</tr>
<tr>
<td>inout</td>
<td>d</td>
<td>data values</td>
</tr>
</tbody>
</table>

32.30.3 Member Function Documentation
32.30.3.1 add()

```cpp
short * Fl_File_Icon::add ( short d )
```

Adds a keyword value to the icon array, returning a pointer to it.

**Parameters**

| in | d | data value |

32.30.3.2 add_color()

```cpp
short* Fl_File_Icon::add_color ( Fl_Color c ) [inline]
```

Adds a color value to the icon array, returning a pointer to it.

**Parameters**

| in | c | color value |

32.30.3.3 add_vertex() [1/2]

```cpp
short* Fl_File_Icon::add_vertex ( float x, float y ) [inline]
```

Adds a vertex value to the icon array, returning a pointer to it. The floating point version goes from 0.0 to 1.0. The origin (0.0) is in the lower-lefthand corner of the icon.

**Parameters**

| in | x, y | vertex coordinates |

32.30.3.4 add_vertex() [2/2]

```cpp
short* Fl_File_Icon::add_vertex ( int x, int y ) [inline]
```

Adds a vertex value to the icon array, returning a pointer to it. The integer version accepts coordinates from 0 to 10000. The origin (0.0) is in the lower-lefthand corner of the icon.

**Parameters**

| in | x, y | vertex coordinates |

32.30.3.5 draw()

```cpp
void Fl_File_Icon::draw ( int x, int y, int w, int h, int hspace, int vspace )
```

Generated by Doxygen
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>

32.30.3.6  find()

Fl_File_Icon * Fl_File_Icon::find (  
    const char * filename,  
    int filetype = ANY ) [static]

Finds an icon that matches the given filename and file type.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>name of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>filetype</td>
<td>enumerated file type</td>
</tr>
</tbody>
</table>

Returns

matching file icon or NULL

32.30.3.7  label()

void Fl_File_Icon::label (  
    Fl_Widget * w )

Applies the icon to the widget, registering the Fl_File_Icon label type as needed.

Parameters

| in  | w | widget for which this icon will become the label |

32.30.3.8  labeltype()

void Fl_File_Icon::labeltype (  
    const Fl_Label * o,  
    int x,  
    int y,  
    int w,  
    int h,  
    Fl_Align a ) [static]

Draw the icon label.

Parameters

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

in  a  label alignment [not used]

32.30.3.9  load()

```cpp
void Fl_File_Icon::load (  
    const char * f )
```

Loads the specified icon image.  
The format is deduced from the filename.

Parameters

in  f  filename

32.30.3.10  load_fti()

```cpp
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

32.30.3.11  load_image()

```cpp
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

32.30.3.12  load_system_icons()

```cpp
void Fl_File_Icon::load_system_icons (  
    void ) [static]
```

Loads all system-defined icons.  
This call is useful when using the Filechooser widget and should be used when the application starts:

Fl_File_Icon::load_system_icons();
32.30.3.13 next()

*Fl_File_Icon* :: next () [inline]

Returns next file icon object.

See *Fl_File_Icon::first()*

32.30.3.14 type()

int Fl_File_Icon::type () [inline]

Returns the filetype associated with the icon, which can be one of the following:

- *Fl_File_Icon::ANY*, any kind of file.
- *Fl_File_Icon::PLAIN*, plain files.
- *Fl_File_Icon::FIFO*, named pipes.
- *Fl_File_Icon::DEVICE*, character and block devices.
- *Fl_File_Icon::LINK*, symbolic links.
- *Fl_File_Icon::DIRECTORY*, directories.

The documentation for this class was generated from the following files:

- *Fl_File_Icon.H*
- *Fl_File_Icon.cxx*
- *Fl_File_Icon2.cxx*

32.31 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.
32.31 Fl_File_Input Class Reference

- virtual int handle (int event)
  Handle events in the widget.
- const char * value ()
  Returns the current value, which is a pointer to an internal buffer and is valid only until the next event is handled.
- int value (const char *str)
  Sets the value of the widget given a new string value.
- int value (const char *str, int len)
  Sets the value of the widget given a new string value and its length.

Protected Member Functions

- virtual void draw ()
  Draws the file input widget.

Additional Inherited Members

32.31.1 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 32.15 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.

32.31.2 Constructor & Destructor Documentation

32.31.2.1 Fl_File_Input()

Fl_File_Input::Fl_File_Input ( int X, int Y, int W, int H, const char *L = 0 )

Creates a new Fl_File_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

Generated by Doxygen
32.31.3 Member Function Documentation

32.31.3.1 down_box()

```cpp
void Fl_File_Input::down_box (Fl_Boxtype b) [inline]
```
Sets the box type to use for the navigation bar.

32.31.3.2 errorcolor() [1/2]

```cpp
Fl_Color Fl_File_Input::errorcolor ( ) const [inline]
```
Gets the current error color.
Returns FL_RED since FLTK 1.4.0 (default in 1.3.x). Retained for backwards compatibility.

**Deprecated**  Will be removed in FLTK 1.5.0 or higher.

**Todo** Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

32.31.3.3 errorcolor() [2/2]

```cpp
void Fl_File_Input::errorcolor (Fl_Color c) [inline]
```
Sets the current error color to c.
Does nothing since FLTK 1.4.0. Retained for backwards compatibility.

**Deprecated**  Will be removed in FLTK 1.5.0 or higher.

**Todo** Remove Fl_File_Input::errorcolor(Fl_Color) in FLTK 1.5.0 or higher.

32.31.3.4 handle()

```cpp
int Fl_File_Input::handle (int event) [virtual]
```
Handle events in the widget.
Return non zero if event is handled.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
</tr>
</thead>
</table>

Reimplemented from Fl_Widget.

32.31.3.5 value() [1/2]

```cpp
int Fl_File_Input::value (const char * str)
```
Sets the value of the widget given a new string value.
Returns non 0 on success.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>new string value</th>
</tr>
</thead>
</table>
32.31.3.6  value() [2/2]

```cpp
int Fl_File_Input::value (
    const char ∗ str,
    int len )
```

Sets the value of the widget given a new string value and its length. Returns non 0 on success.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>new string value</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>length of value</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_File_Input.H
- Fl_File_Input.cxx

32.32  Fl_Fill_Dial Class Reference

Draws a dial with a filled arc.

```cpp
#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

32.32.1  Detailed Description

Draws a dial with a filled arc.

The documentation for this class was generated from the following files:

- Fl_Fill_Dial.H
- Fl_Dial.cxx

32.33  Fl_Fill_Slider Class Reference

Widget that draws a filled horizontal slider, useful as a progress or value meter.

```cpp
#include <Fl_Fill_Slider.H>
```

Inheritance diagram for Fl_Fill_Slider:

Generated by Doxygen
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

32.33.1 Detailed Description

Widget that draws a filled horizontal slider, useful as a progress or value meter. The documentation for this class was generated from the following files:

- Fl_Fill_Slider.H
- Fl_Slider.cxx

32.34 Fl_Flex Class Reference

**Fl_Flex** is a container (layout) widget for one row or one column of widgets.

```c
#include <Fl_Flex.H>
```

Inheritance diagram for Fl_Flex:

```
Fl_Widget
|     |
|     |
|     |
```

Public Types

- enum { VERTICAL = 0, HORIZONTAL = 1, COLUMN = 0, ROW = 1 }

Public Member Functions

- virtual void end ()
  
  Ends automatic child addition and resizes all children.

- Fl_Flex (int direction)
  
  Construct a new Fl_Flex widget specifying its layout.

- Fl_Flex (int w, int h, int direction)
  
  Construct a new Fl_Flex widget specifying its layout and size.

- Fl_Flex (int X, int Y, int W, int H, const char *L=0)
  
  Construct a new Fl_Flex widget with the given position, size, and label.

- Fl_Flex (int x, int y, int w, int h, int direction)
Construct a new Fl_Flex widget specifying its layout, position, and size.

- int gap () const
  Return the gap size of the widget.
- void gap (int g)
  Set the gap size of the widget.
- int horizontal () const
  Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).
- bool isSetSize (Fl_Widget *w) const
  Deprecated.
- void layout ()
  Calculates the layout of the widget and redraws it.
- int margin () const
  Returns the left margin size of the widget.
- void margin (int left, int top, int right, int bottom)
  Set the margin sizes at all four edges of the Fl_Flex widget.
- void margin (int m, int g=-1)
  Set the margin and optionally the gap size of the widget.
- int margins (int *left, int *top, int *right, int *bottom) const
  Returns all (four) margin sizes of the widget.
- virtual void resize (int x, int y, int w, int h)
  Changes the size or position of the widget.
- void set_size (Fl_Widget &w, int size)
  Set the horizontal or vertical size of a child widget.
- int set_size (Fl_Widget *w) const
  Return whether the given widget has a fixed size or resizes dynamically.
- void setSize (Fl_Widget *w, int size)
  Set the horizontal or vertical size of a child widget.
- 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

- virtual int alloc_size (int size) const
  Return new size to be allocated for array of fixed size widgets.
- void init (int t=VERTICAL)

Additional Inherited Members

32.34.1 Detailed Description

Fl_Flex is a container (layout) widget for one row or one column of widgets. It provides flexible positioning of its children either in one row or in one column. Fl_Flex is designed to be as simple as possible. You can set individual widget sizes or let Fl_Flex position and size the widgets to fit in the container. All "flexible" (i.e. non-fixed size) widgets are assigned the same width or height, respectively. For details see below.

You can set the margins around all children at the inner side the box frame (if any). Fl_Flex supports setting different margin sizes on top, bottom, left and right sides. The default margin size is 0 on all edges of the container. You can set the gap size between all children. The gap size is always the same between all of its children. This is similar to the 'spacing' of Fl_Pack. The default gap size is 0.
**FL_Flex** can either consist of a single row, i.e. `type(Fl_Flex::HORIZONTAL)` or a single column, i.e. `type(Fl_Flex::VERTICAL)`. The default value is **FL_Flex::VERTICAL** for consistency with **FL_Pack** but you can use `type()` to assign a row (**FL_Flex::HORIZONTAL**) layout.

If `type() == FL_Flex::HORIZONTAL` widgets are resized horizontally to fit in the container and their height is the full **FL_Flex** height minus border size and margins. You can set a fixed widget width by using `set_size()`.

If `type() == FL_Flex::VERTICAL` widgets are resized vertically to fit in the container and their width is the full **FL_Flex** width minus border size and margins. You can set a fixed widget height by using `set_size()`.

To create arbitrary spacing you can use invisible boxes of flexible or fixed sizes (see example below). Alternate constructors let you specify the layout as **FL_Flex::HORIZONTAL** or **FL_Flex::VERTICAL** directly. **FL_Flex::ROW** is an alias of **FL_Flex::HORIZONTAL** and **FL_Flex::COLUMN** is an alias of **FL_Flex::VERTICAL**.

The default box type is **FL_NO_BOX** as inherited from **FL_Group**. You **may** need to set a box type with a solid background depending on your layout.

**Important:** You should always make sure that the **FL_Flex** container cannot be resized smaller than its designed minimal size. This can usually be done by setting a `size_range()` on the window as shown in the example below. **FL_Flex** does not take care of sensible sizes. If it is resized too small the behavior is undefined, i.e. widgets may overlap and/or shrink to zero size.

**Hint:** In many cases **FL_Flex** can be used as a drop-in replacement for **FL_Pack**. This is the recommended single row/column container since FLTK 1.4.0. Its resizing behavior is much more predictable (as expected) than that of **FL_Pack** which "resizes itself to shrink-wrap itself around all of the children". **FL_Flex** containers can be nested so you can create flexible layouts with multiple columns and rows. However, if your UI design is more complex you may want to use **FL_Grid** instead. At the time of this writing (Aug 7, 2022) **FL_Grid** is not yet available but will be added before FLTK 1.4.0 gets released.

**Example:**

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Flex.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>

int main(int argc, char **argv) {
    Fl_Double_Window window(410, 40, "Simple Fl_Flex Demo");
    Fl_Flex flex(5, 5, 400, 30, Fl_Flex::HORIZONTAL);
    Fl_Button b1(0, 0, 0, 0, "File");
    Fl_Button b2(0, 0, 0, 0, "Save");
    Fl_Box bx(0, 0, 0, 0);
    Fl_Button b3(0, 0, 0, 0, "Exit");
    flex.set_size(bx, 60); // set fix width of invisible box
    flex.gap(10);
    flex.end();
    window.resizable(flex);
    window.end();
    window.size_range(300, 30);
    window.show(argc, argv);
    return Fl::run();
}
```

Since

1.4.0

### 32.34.2 Member Enumeration Documentation

#### 32.34.2.1 anonymous enum

**anonymous enum**

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERTICAL</td>
<td>vertical layout (one column)</td>
</tr>
<tr>
<td>HORIZONTAL</td>
<td>horizontal layout (one row)</td>
</tr>
</tbody>
</table>

---

Generated by Doxygen
32.34.3 Constructor & Destructor Documentation

32.34.3.1 Fl_Flex() [1/4]

Fl_Flex::Fl_Flex (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ L = 0 )

Construct a new Fl_Flex widget with the given position, size, and label. You can set type(Fl_Flex::HORIZONTAL) or type(Fl_Flex::VERTICAL). The default is type(Fl_Flex::VERTICAL).
Alternate constructors let you specify the layout as Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL directly. Fl_Flex::ROW is an alias of Fl_Flex::HORIZONTAL and Fl_Flex::COLUMN is an alias of Fl_Flex::VERTICAL.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>XY</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W, H</td>
<td>size (width and height)</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>label (optional)</td>
</tr>
</tbody>
</table>

See also

Fl_Flex::Fl_Flex(int direction)
Fl_Flex::Fl_Flex(int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char ∗ L)

32.34.3.2 Fl_Flex() [2/4]

Fl_Flex::Fl_Flex (  
  int direction )

Construct a new Fl_Flex widget specifying its layout. Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.
This constructor sets the position and size to (0, 0, 0, 0) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position and size as well.

Parameters

| in | direction  | horizontal (row) or vertical (column) layout |
See also

Fl_Flex::Fl_Flex(int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

32.34.3.3 Fl_Flex() [3/4]

Fl_Flex::Fl_Flex (  
    int w,  
    int h,  
    int direction )  
Construct a new Fl_Flex widget specifying its layout and size.
Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.
This constructor sets the position to (x = 0, y = 0) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position as well.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>w,h</th>
<th>widget size</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>direction</td>
<td>horizontal (row) or vertical (column) layout</td>
</tr>
</tbody>
</table>

See also

Fl_Flex::Fl_Flex(int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

32.34.3.4 Fl_Flex() [4/4]

Fl_Flex::Fl_Flex (  
    int x,  
    int y,  
    int w,  
    int h,  
    int direction )  
Construct a new Fl_Flex widget specifying its layout, position, and size.
Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the direction argument.
This constructor sets the position and size of the widget which is suitable for top level Fl_Flex widgets but does not set a widget label. Use Fl_Widget::label() to set one if desired.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>widget position</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w,h</td>
<td>widget size</td>
</tr>
<tr>
<td>in</td>
<td>direction</td>
<td>horizontal (row) or vertical (column) layout</td>
</tr>
</tbody>
</table>

See also

Fl_Flex::Fl_Flex(int direction)
Fl_Flex::Fl_Flex(int w, int h, int direction)
32.34.4 Member Function Documentation

32.34.4.1 alloc_size()

```cpp
int Fl_Flex::alloc_size (int size) const [protected], [virtual]
```

Return new size to be allocated for array of fixed size widgets. This method is called when the array of fixed size widgets needs to be expanded. The current size is provided (size can be 0). The default method adds 8 to the current size. This can be used in derived classes to change the allocation strategy. Note that this method only queries the new size which shall be allocated but does not allocate the memory.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>size</th>
<th>current size</th>
</tr>
</thead>
</table>

Returns

int new size (to be allocated)

32.34.4.2 end()

```cpp
void Fl_Flex::end () [virtual]
```

Ends automatic child addition and resizes all children. This calculates the layout depending on all children and whether they have been assigned fix sizes or not.

32.34.4.3 gap() [1/2]

```cpp
int Fl_Flex::gap ( ) const [inline]
```

Return the gap size of the widget.

Returns

gap size between all child widgets.

32.34.4.4 gap() [2/2]

```cpp
void Fl_Flex::gap (int g) [inline]
```

Set the gap size of the widget. The gap size is some free space between child widgets. The size must be \( \geq 0 \). Negative values are clamped to 0.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>g</th>
<th>gap size</th>
</tr>
</thead>
</table>

32.34.4.5 horizontal()

```cpp
int Fl_Flex::horizontal ( ) const [inline]
```
Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).

Returns

non-zero if Fl_Flex alignment is horizontal

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if type() == Fl_Flex::HORIZONTAL</td>
</tr>
<tr>
<td>0</td>
<td>if type() == Fl_Flex::VERTICAL</td>
</tr>
</tbody>
</table>

See class Fl_Flex documentation for details.

### 32.34.4.6 isSetSize()

```cpp
bool Fl_Flex::isSetSize ( Fl_Widget ∗ w ) const [inline]
```

Deprecated.

**Deprecated** Please use set_size(Fl_Widget ∗) instead.

See also

int set_size(Fl_Widget ∗)

### 32.34.4.7 layout()

```cpp
void Fl_Flex::layout ( ) [inline]
```

Calculates the layout of the widget and redraws it. If you change widgets in the Fl_Flex container you should call this method to force recalculation of child widget sizes and positions. This can be useful (necessary) if you hide(), show(), add() or remove() children. This method also calls redraw() on the Fl_Flex widget.

### 32.34.4.8 margin() {1/3}

```cpp
int Fl_Flex::margin ( ) const [inline]
```

Returns the left margin size of the widget.

This returns the left margin of the widget which is not necessarily the same as all other margins.

**Note**

This method is useful if you never set different margin sizes.

See also

int margins(int ∗left, int ∗top, int ∗right, int ∗bottom) to get all four margin values.

Returns

size of left margin.
### 32.34.4.9 margin() [2/3]

```cpp
void Fl_Flex::margin (  
    int left,  
    int top,  
    int right,  
    int bottom ) [inline]
```

Set the margin sizes at all four edges of the Fl_Flex widget. The margin is the free space inside the widget border around all child widgets. You must use all four parameters of this method to set the four margins in the order `left,top,right,bottom`. Negative values are set to 0 (zero). To set all margins to equal sizes, use `margin(int m)` This method sets the margin to the same size at all four edges of the widget.

**Parameters**

| in | left,top,right,bottom | margin sizes, must be >= 0 |

See also

`margin(int, int)`

### 32.34.4.10 margin() [3/3]

```cpp
void Fl_Flex::margin (  
    int m,  
    int g = -1 ) [inline]
```

Set the margin and optionally the gap size of the widget. This method can be used to set both the margin and the gap size. If you don’t use the second parameter `g` or supply a negative value the gap size is not changed. The margin is some free space inside the widget border around all child widgets. This method sets the margin to the same size at all four edges of the Fl_Flex widget. The gap size `g` is some free space between child widgets. Negative values (the default if this argument is omitted) do not change the gap value.

**Parameters**

| in | m | margin size, must be >= 0 |
| in | g | gap size, ignored (if negative) |

See also

`gap(int)`

### 32.34.4.11 margins()

```cpp
int Fl_Flex::margins (  
    int * left,  
    int * top,  
    int * right,  
    int * bottom ) const [inline]
```

Returns all (four) margin sizes of the widget. All margin sizes are returned in the given arguments. If any argument is `NULL` the respective value is not returned.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>left</code></td>
<td>returns left margin if not NULL</td>
</tr>
<tr>
<td><code>top</code></td>
<td>returns top margin if not NULL</td>
</tr>
<tr>
<td><code>right</code></td>
<td>returns right margin if not NULL</td>
</tr>
<tr>
<td><code>bottom</code></td>
<td>returns bottom margin if not NULL</td>
</tr>
</tbody>
</table>

### Returns

whether all margins are equal

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all margins have the same size</td>
</tr>
<tr>
<td>0</td>
<td>at least one margin has a different size</td>
</tr>
</tbody>
</table>

### 32.34.4.12 resize()

```cpp
void Fl_Flex::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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x,y</code></td>
<td>new position relative to the parent window</td>
</tr>
<tr>
<td><code>w,h</code></td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int,int)`, `size(int,int)`

Reimplemented from `Fl_Widget`.

### 32.34.4.13 set_size() [1/3]

```cpp
void Fl_Flex::set_size (  
    Fl_Widget & w,  
    int size )  [inline]
```

Set the horizontal or vertical size of a child widget.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w</code></td>
<td>widget to be affected</td>
</tr>
<tr>
<td><code>size</code></td>
<td>width (<code>Fl_Flex::HORIZONTAL</code>) or height (<code>Fl_Flex::VERTICAL</code>)</td>
</tr>
</tbody>
</table>
32.34.4.14 set_size() [2/3]

int Fl_Flex::set_size (Fl_Widget * w) const

Return whether the given widget has a fixed size or resizes dynamically.

Parameters

| in  | w     | widget |

Returns

whether the widget has a fixed size

Return values

| 1 | the widget has a fixed size |
| 0 | the widget resizes dynamically |

32.34.4.15 set_size() [3/3]

void Fl_Flex::set_size (Fl_Widget * w, int size)

Set the horizontal or vertical size of a child widget. This sets either the width or height of a child widget, depending on the type() of the Fl_Flex container (Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL). The other dimension is set to the full width or height of the Fl_Flex widget. This can be used to set a fixed widget width or height of children of Fl_Flex so they are not resized dynamically. If size is 0 (zero) or negative the widget size is reset to flexible size.

Parameters

| in  | w         | widget to be affected |
|     | size      | width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL) |

32.34.4.16 setSize()

void Fl_Flex::setSize (Fl_Widget * w, int size) [inline]

Set the horizontal or vertical size of a child widget.

Deprecated Please use set_size(Fl_Widget *, int) instead.

See also

set_size(Fl_Widget *, int)
32.34.4.17  spacing() [1/2]

int Fl_Flex::spacing () const [inline]

Gets the number of extra pixels of blank space that are added between the children.
This method is the same as `int gap()` and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

   int gap()

32.34.4.18  spacing() [2/2]

void Fl_Flex::spacing ( int i ) [inline]

Sets the number of extra pixels of blank space that are added between the children.
This method is the same as `gap(int)` and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

   void gap(int)

The documentation for this class was generated from the following files:

   • Fl_Flex.H
   • Fl_Flex.cxx

32.35  Fl_Float_Input Class Reference

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits).
#include <Fl_Float_Input.H>

Inheritance diagram for Fl_Float_Input:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Input_</td>
</tr>
<tr>
<td>Fl_Input</td>
</tr>
<tr>
<td>Fl_Float_Input</td>
</tr>
</tbody>
</table>
```

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

32.35.1  Detailed Description

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits).

32.35.2  Constructor & Destructor Documentation
32.35.2.1 Fl_Float_Input()

Fl_Float_Input::Fl_Float_Input (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ l = 0 )

Creates a new Fl_Float_Input widget using the given position, size, and label string.  
The default boxtype is FL_DOWN_BOX.  
Inherited destructor destroys the widget and any value associated with it.  
The documentation for this class was generated from the following files:

- Fl_Float_Input.H
- Fl_Input.cxx

32.36 Fl_FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.

#include <Fl_FormsBitmap.H>

Inheritance diagram for Fl_FormsBitmap:

```
Inheritance diagram for Fl_FormsBitmap:
```

**Public Member Functions**

- **Fl_Bitmap ∗ bitmap () const**  
  Gets the current associated Fl_Bitmap objects.

- **void bitmap (Fl_Bitmap ∗B)**  
  Sets a new bitmap.

- **Fl_FormsBitmap (Fl_Boxtype, int, int, int, int, const char ∗=0)**  
  Creates a bitmap widget from a box type, position, size and optional label specification.

- **void set (int W, int H, const uchar ∗bits)**  
  Sets a new bitmap bits with size W,H.

**Protected Member Functions**

- **void draw ()**  
  Draws the bitmap and its associated box.

**Additional Inherited Members**

32.36.1 Detailed Description

Forms compatibility Bitmap Image Widget.

32.36.2 Member Function Documentation
32.36.2.1 set()

```c
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

### 32.37 Fl_FormsPixmap Class Reference

Forms pixmap drawing routines.

```
#include <Fl_FormsPixmap.H>
```

Inheritance diagram for Fl_FormsPixmap:

```
Fl_Widget
  ↓
Fl_FormsPixmap
```

#### Public Member Functions

- **Fl_FormsPixmap (Fl_Boxtype t, int X, int Y, int W, int H, const char *L=0)**
  
  Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.

- **Fl_Pixmap * Pixmap () const**
  
  Get the internal pixmap pointer.

- **void Pixmap (Fl_Pixmap *B)**
  
  Set the internal pixmap pointer to an existing pixmap.

- **void set (char *const *bits)**
  
  Set/create the internal pixmap using raw data.

#### Protected Member Functions

- **void draw ()**
  
  Draws the widget.

### Additional Inherited Members

#### 32.37.1 Detailed Description

Forms pixmap drawing routines.

#### 32.37.2 Constructor & Destructor Documentation

##### 32.37.2.1 Fl_FormsPixmap()

```
Fl_FormsPixmap::Fl_FormsPixmap (  
    Fl_Boxtype t,  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char *L=0
```

- **void Fl_FormsPixmap::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
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

| in | t | box type |
| in | X,Y,W,H | position and size |
| in | L | widget label, default is no label |

#### Member Function Documentation

##### 32.37.3.1 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.

##### 32.37.3.2 Pixmap()

```cpp
void Fl_FormsPixmap::Pixmap ( Fl_Pixmap ∗ B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

### Parameters

| in | B | existing pixmap |

##### 32.37.3.3 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
- forms_pixmap.cxx
32.38 Fl_FormsText Class Reference

Inheritance diagram for Fl_FormsText:

```
Fl_Widget
    Fl_FormsText
```

Public Member Functions

- **Fl_FormsText** (*Fl_Boxtype* t, int X, int Y, int W, int H, const char *l=0)

Protected Member Functions

- **void draw()**

  Draws the widget.

Additional Inherited Members

32.38.1 Member Function Documentation

32.38.1.1 draw()

```cpp
default draw()
```

This function is declared as protected and virtual in the Fl_Widget base class. It is used to draw the widget. 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.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call **redraw()** instead.

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.

The documentation for this class was generated from the following file:

- **forms.H**

32.39 Fl_Free Class Reference

Emulation of the Forms “free” widget.

```cpp
#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.
32.39 Fl_Free Class Reference

- 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

32.39.1 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.

32.39.2 Constructor & Destructor Documentation

32.39.2.1 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:

```cpp
int handle_function(Fl_Widget *w, int event, float event_x, float event_y, char key)
```

This function is called from the handle() method in response to most events, and is called by the draw() method.

Generated by Doxygen
The event argument contains the event type:

// old event names for compatibility:
#define FL_MOUSE FL_DRAG
#define FL_DRAW 0
#define FL_STEP 9
#define FL_FREEMEM 12
#define FL_FREEZE FL_UNMAP
#define FL_THAW FL_MAP

32.39.3 Member Function Documentation

32.39.3.1 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.

32.39.3.2 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 retwal.

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

32.40 Fl_GIF_Image Class Reference

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF images.
#include <Fl_GIF_Image.H>
Inheritance diagram for Fl_GIF_Image:

```
Fl_GIF_Image
    Fl_Pixmap
    Fl_Image
```

**Public Member Functions**

- `Fl_GIF_Image (const char ∗filename)`
  *This constructor loads a GIF image from the given file.*
- `Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data)`
  *This constructor loads a GIF image from memory (deprecated).*
- `Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data, const size_t length)`
  *This constructor loads a GIF image from memory.*

**Protected Member Functions**

- `void load_gif_ (class Fl_Image_Reader &rdr)`

**Additional Inherited Members**

32.40.1 Detailed Description

The `Fl_GIF_Image` class supports loading, caching, and drawing of Compuserve GIF images. The class loads the first image and supports transparency.

32.40.2 Constructor & Destructor Documentation

32.40.2.1 `Fl_GIF_Image()` [1/3]

```
Fl_GIF_Image::Fl_GIF_Image (const char ∗filename)
```

This constructor loads a GIF image from the given file.

If a GIF image is animated, `Fl_GIF_Image` will only read and display the first frame of the animation. The destructor frees all memory and server resources that are used by the image.

Use `Fl_Image::fail()` to check if `Fl_GIF_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

```
in  filename  a full path and name pointing to a GIF image file.
```

See also

`Fl_GIF_Image::Fl_GIF_Image(const char ∗imagename, const unsigned char ∗data, const size_t length)`

32.40.2.2 `Fl_GIF_Image()` [2/3]

```
Fl_GIF_Image::Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data, const long length)
```
This constructor loads a GIF image from memory (deprecated).

**Deprecated** Please use `FL_GIF_Image(const char *imagename, const unsigned char *data, const size_t length)` instead.

**Note**
Buffer overruns will not be checked.

This constructor should not be used because the caller can't supply the memory size and the image reader can't check for "end of memory" errors.

**Note**
A new constructor with parameter `length` is available since FLTK 1.4.0.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> <code>imagename</code></td>
</tr>
<tr>
<td><strong>in</strong> <code>data</code></td>
</tr>
</tbody>
</table>

**See also**
`FL_GIF_Image(const char *filename)`
`FL_GIF_Image(const char *imagename, const unsigned char *data, const size_t length)`

### 32.40.2.3 `FL_GIF_Image()` [3/3]

`FL_GIF_Image::FL_GIF_Image (const char *imagename, const unsigned char *data, const size_t length)`

This constructor loads a GIF image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. `imagename` can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

If a GIF image is animated, `FL_GIF_Image` will only read and display the first frame of the animation.

The destructor frees all memory and server resources that are used by the image.

The third parameter `length` is used to test for buffer overruns, i.e. truncated images.

Use `FL_Image::fail()` to check if `FL_GIF_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> <code>imagename</code></td>
</tr>
<tr>
<td><strong>in</strong> <code>data</code></td>
</tr>
<tr>
<td><strong>in</strong> <code>length</code></td>
</tr>
</tbody>
</table>

**See also**
`FL_GIF_Image::FL_GIF_Image(const char *filename)`
`FL_Shared_Image`
Since 1.4.0

The documentation for this class was generated from the following files:
  
  • Fl_GIF_Image.H
  • Fl_GIF_Image.cxx

### 32.41 Fl_Gl_Choice Class Reference

**Public Member Functions**

- `Fl_Gl_Choice (int m, const int ∗alistp, Fl_Gl_Choice ∗n)`

**Friends**

- `class Fl_Gl_Window_Driver`

The documentation for this class was generated from the following file:

- `Fl_Gl_Choice.H`

### 32.42 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 window’s OpenGL rendering context.

- `void context (GLContext, int destroy_flag=0)`
  
  Sets a pointer to the GLContext that this window is using.

- `char context_valid () const`
  
  Will only be set if the OpenGL context is created or recreated.
• **void** **context_valid** (char v)
  See **Fl_Gl_Window::context_valid()** const.
• **Fl_Gl_Window** (int W, int H, const char ∗=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 ∗=0)
  Creates a new **Fl_Gl_Window** widget using the given position, size, and label string.
• **void** **flush** ()
  Forces the window to be drawn, this window is also made current and calls **draw()**.
• **int** **handle** (int)
  Handle some FLTK events as needed.
• **void** **hide** ()
  Hides the window and destroys the OpenGL context.
• **void** **hide_overlay** ()
  Hides the window if it is not this window, does nothing in Windows.
• **void** **invalidate** ()
  The **invalidate()** method turns off **valid()** and is equivalent to calling **value(0)**.
• **void** **make_current** ()
  The **make_current()** method selects the OpenGL context for the widget.
• **void** **make_overlay_current** ()
  Selects the OpenGL context for the widget's overlay.
• **Fl_Mode** **mode** () const
  Returns the current OpenGL capabilities of the window.
• **int** **mode** (const int ∗a)
  Set the OpenGL capabilities of the window using platform-specific data.
• **int** **mode** (int a)
  Set or change the OpenGL capabilities of the window.
• **void** **ortho** ()
  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.
• **int** **pixel_h** ()
  Gives the window height in OpenGL pixels.
• **int** **pixel_w** ()
  Gives the window width in OpenGL pixels.
• **float** **pixels_per_unit** ()
  The number of pixels per FLTK unit of length for the window.
• **void** **redraw_overlay** ()
  Causes **draw_overlay()** to be called at a later time.
• **void** **resize** (int, int, int, int)
  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 **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 (const int *m)
  
  Returns non-zero if the hardware supports the given OpenGL mode.
- static int can_do (int m)
  
  Returns non-zero if the hardware supports the given OpenGL mode.

Protected Member Functions

- virtual void draw ()
  
  Draws the Fl_Gl_Window.
- void draw_begin ()
  
  Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.
- void draw_end ()
  
  To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

Friends

- class Fl_Gl_Window_Driver

Additional Inherited Members

32.42.1 Detailed Description

The Fl_Gl_Window widget sets things up so OpenGL works. It also keeps an OpenGL "context" for that window, so that changes to the lighting and projection may be reused between redraws. Fl_Gl_Window also flushes the OpenGL streams and swaps buffers after draw() returns.

OpenGL hardware typically provides some overlay bit planes, which are very useful for drawing UI controls atop your 3D graphics. If the overlay hardware is not provided, FLTK tries to simulate the overlay. This works pretty well if your graphics are double buffered, but not very well for single-buffered.

Please note that the FLTK drawing and clipping functions will not work inside an Fl_Gl_Window. All drawing should be done using OpenGL calls exclusively.

See also

OpenGL and support of HighDPI displays

Note

FLTK 1.4 introduces a driver system for graphic calls. It is now possible to add a selection of widgets to an OpenGL window. The widgets will draw on top of any OpenGL rendering. The number of supported widgets will increase as the driver development improves. Program test/cube.cxx illustrates how to do that.

FLTK expects that when an Fl_Gl_Window is a child of a parent Fl_Window, the child window lies entirely inside its parent window. If that's not the case, what happens to the part of the GL subwindow which leaks outside its parent is undefined and susceptible to be platform-specific.

32.42.2 Constructor & Destructor Documentation

32.42.2.1 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.
32.42.2.2 Fl_Gl_Window() [2/2]

Fl_Gl_Window::Fl_Gl_Window (  
int \textit{X},  
int \textit{Y},  
int \textit{W},  
int \textit{H},  
const char ∗ \textit{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.

32.42.3 Member Function Documentation

32.42.3.1 as_gl_window()

\texttt{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.

\textbf{Return values}

\begin{tabular}{|c|c|}
\hline
NULL & if this widget is not derived from Fl_Gl_Window. \\
\hline
\end{tabular}

\textbf{Note}

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_window()

Reimplemented from Fl_Widget.

32.42.3.2 can_do()

\texttt{static int Fl\_Gl\_Window::can\_do (  
const int ∗ \textit{m} ) [inline], [static]}

Returns non-zero if the hardware supports the given OpenGL mode.

See also

Fl_Gl_Window::mode(const int ∗\textit{a})

32.42.3.3 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.

32.42.3.4 context() [1/2]

GLContext Fl_Gl_Window::context ( ) const [inline]

Returns a pointer to the window's OpenGL rendering context.
See also

```cpp
void context(GLContext c, int destroy_flag)
```

### 32.42.3.5 context()

```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.

### 32.42.3.6 context_valid()

`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.

### 32.42.3.7 draw()

```cpp
void Fl_Gl_Window::draw ( ) [protected], [virtual]
```

Draws the `Fl_Gl_Window`. You must subclass `Fl_Gl_Window` and provide an implementation for `draw()`. You 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()) {
    glLoadIdentity();
    glViewport(0,0,pixel_w(),pixel_h());
    glFrustum(... or glOrtho(...)
    ...other initialization...
  }
  if (!context_valid()) {
    ...load textures, etc. ...
  }
  // clear screen
gClearColor(...);
gClear(...);
  ... 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
gClearColor(GL_COLOR_BUFFER_BIT);
  // Draw white 'X'
gColor3f(1.0, 1.0, 1.0);
gBegin(GL_LINE_STRIP); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
gBegin(GL_LINE_STRIP); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

Regular FLTK widgets can be added as children to the `Fl_Gl_Window`. To correctly overlay the widgets, `Fl_Gl_Window::draw()` must be called after rendering the main scene.

```cpp
void mywindow::draw() {
  // draw 3d graphics scene
  Fl_Gl_Window::draw();
  // -- or --
```

Generated by Doxygen
draw_begin();
Fl_Window::draw();
// other 2d drawing calls, overlays, etc.
draw_end();

Implements Fl_Widget.
Reimplemented in Fl_Glut_Window.

32.42.3.8 draw_begin()

void Fl_Gl_Window::draw_begin ( ) [protected]
Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.

See also
Using FLTK widgets in OpenGL Windows

32.42.3.9 draw_end()

void Fl_Gl_Window::draw_end ( ) [protected]
To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

See also
Using FLTK widgets in OpenGL Windows

32.42.3.10 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.

32.42.3.11 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.

32.42.3.12 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).

32.42.3.13 mode() [2/3]

int Fl_Gl_Window::mode {
    const int ∗ a } [inline]
Set the OpenGL capabilities of the window using platform-specific data.

Parameters

| a | zero-ending array of platform-specific attributes and attribute values |

Unix/Linux platform: attributes are GLX attributes adequate for the 3rd argument of the glXChooseVisual()
function (e.g., GLX_DOUBLEBUFFER, defined by including <GL/glx.h>).

Note

What attributes are adequate here is subject to change. The preferred, stable public API is Fl_Gl_Window::mode(int a).

Windows platform: this member function is of no use.
Mac OS X platform: attributes belong to the CGLPixelFormatAttribute enumeration (defined by including <OpenGL/OpenGL.h>, e.g., kCGLPFADoubleBuffer) and may be followed by adequate attribute values.

32.42.3.14 mode()

int Fl_Gl_Window::mode (int a) [inline]

Set or change the OpenGL capabilites of the window. The value can be any of the following OR'd together:

- FL_RGB - RGB color (not indexed)
- FL_RGB8 - RGB color with at least 8 bits of each color
- FL_INDEX - Indexed mode
- FL_SINGLE - not double buffered
- FL_DOUBLE - double buffered
- FL_ACCUM - accumulation buffer
- FL_ALPHA - alpha channel in color
- FL_DEPTH - depth buffer
- FL_STENCIL - stencil buffer
- FL_MULTISAMPLE - multisample antialiasing
- FL_OPENGL3 - use OpenGL version 3.0 or more.

FL_RGB and FL_SINGLE have a value of zero, so they are "on" unless you give FL_INDEX or FL_DOUBLE.
If the desired combination cannot be done, FLTK will try turning off FL_MULTISAMPLE. If this also fails the show() will call Fl::error() and not show the window.
You can change the mode while the window is displayed. This is most useful for turning double-buffering on and off. Under X this will cause the old X window to be destroyed and a new one to be created. If this is a top-level window this will unfortunately also cause the window to blink, raise to the top, and be de-iconized, and the xid() will change, possibly breaking other code. It is best to make the GL window a child of another window if you wish to do this!
mode() must not be called within draw() since it changes the current context.
The FL_OPENGL3 flag is required to access OpenGL version 3 or more under the X11 and MacOS platforms; it's optional under Windows and Wayland. See more details in Using OpenGL 3.0 (or higher versions).

Version

the FL_OPENGL3 flag appeared in version 1.3.4

32.42.3.15 ortho()

void Fl_Gl_Window::ortho ()

Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.
If you are drawing 2D images, your draw() method may want to call this if valid() is false.
32.42.3.16  pixel_h()

int Fl_Gl_Window::pixel_h ( ) [inline]
Gives the window height in OpenGL pixels.
When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::h() which is expressed
in FLTK units, may differ from the window height in pixels. Calls to OpenGL functions expecting pixel values (e.g.,
glViewport) should therefore use pixel_h() rather than h(). Method pixel_h() detects when the GUI is rescaled or
when the window has been moved between low and high resolution displays and automatically adjusts the returned
value.
Version
1.3.4

32.42.3.17  pixel_w()

int Fl_Gl_Window::pixel_w ( ) [inline]
Gives the window width in OpenGL pixels.
When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::w() which is expressed
in FLTK units, may differ from the window width in pixels. Calls to OpenGL functions expecting pixel values (e.g.,
glViewport) should therefore use pixel_w() rather than w(). Method pixel_w() detects when the GUI is rescaled or
when the window has been moved between low and high resolution displays and automatically adjusts the
returned value.
Version
1.3.4

32.42.3.18  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

32.42.3.19  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.

32.42.3.20  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`.

### 32.42.3.21 show()

`void Fl_Gl_Window::show ( ) [virtual]`

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

If the window is already shown then it is restored and raised to the top. This is really convenient because your program can call `show()` at any time, even if the window is already up. It also means that `show()` serves the purpose of `raise()` in other toolkits.

`Fl_Window::show(int argc, char **argv)` is used for top-level windows and allows standard arguments to be parsed from the command-line.

**Note**

For some obscure reasons `Fl_Window::show()` resets the current group by calling `Fl_Group::current(0)`. The comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have unwanted side effects if you `show()` an unrelated window (maybe for an error message or warning) while building a window or any other group widget.

**Todo**

Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

**See also**

`Fl_Window::show(int argc, char **argv)`

Reimplemented from `Fl_Window`.

### 32.42.3.22 swap_buffers()

`void Fl_Gl_Window::swap_buffers ( )`

The `swap_buffers()` method swaps the back and front buffers.

It is called automatically after the `draw()` method is called.

### 32.42.3.23 valid()

`char Fl_Gl_Window::valid ( ) const [inline]`

Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after `draw()` is called.

You can use this inside your `draw()` method to avoid unnecessarily initializing the OpenGL context. Just do this:

```cpp
void mywindow::draw() {
  if (!valid()) {
    glViewport(0,0,pixel_w(),pixel_h());
    glFrustum(...);
    ...other initialization...
  }
  if (!context_valid()) {
    ...load textures, etc. ...
    ...draw your geometry here ...
  }
}
```

You can turn `valid()` on by calling `valid(1)`. You should only do this after fixing the transformation inside a `draw()` or after `make_current()`. This is done automatically after `draw()` returns.

The documentation for this class was generated from the following files:

- `Fl_Gl_Window.H`
- `Fl_Gl_Overlay.cxx`
- `Fl_Gl_Window.cxx`

Generated by Doxygen
32.43 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

32.43.1 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

32.44 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

32.45 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

32.46 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
32.47 Fl_Glut_StrokeVertex Struct Reference

Public Attributes

- GLfloat X
- GLfloat Y

The documentation for this struct was generated from the following file:

- glut.H

32.48 Fl_Glut_Window Class Reference

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

```cpp
#include <glut.H>
```

Inheritance diagram for Fl_Glut_Window:

```
   Fl_Widget
      |
      Fl_Group
         |
         Fl_Window
            |
            Fl_Gl_Window
               |
               Fl_Glut_Window
```

Public Member Functions

- Fl_Glut_Window (int w, int h, const char *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

32.48.1 Detailed Description

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

32.48.2 Member Function Documentation

32.48.2.1 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:

```cpp
void mywindow::draw() {
  if (!valid()) {
    glViewport(0,0,pixel_w(),pixel_h());
    glFrustum(...) or glOrtho(...)
    ...other initialization...
  }
  if (!context_valid()) {
    ...load textures, etc. ...
  }
  // clear screen
  glClearColor(...);
  glClear(...);
  ... draw your geometry here ...
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```cpp
void mywindow::draw() {
  if (!valid()) {
    glLoadIdentity();
    glViewport(0,0,pixel_w(),pixel_h());
    glOrtho(-w(),w(),-h(),h(),-1,1);
  }
  // Clear screen
  glClearColor(GL_COLOR_BUFFER_BIT);
  glClear(GL_COLOR_BUFFER_BIT);
  // Draw white 'X'
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_TRIANGLES);
  glVertex2f(w(), h());
  glVertex2f(-w(),-h());
  glEnd();
  glBegin(GL_TRIANGLES);
  glVertex2f(w(),-h());
  glVertex2f(-w(), h());
  glEnd();
}
```

Regular FLTK widgets can be added as children to the Fl_Gl_Window. To correctly overlay the widgets, Fl_Gl_Window::draw() must be called after rendering the main scene.

```cpp
void mywindow::draw() {
  // draw 3D graphics scene
  Fl_Gl_Window::draw();
  // -- or --
  draw_begin();
  Fl_Window::draw();
  // other 2D drawing calls, overlays, etc.
  draw_end();
}
Reimplemented from Fl_Gl_Window.

32.48.2.2 draw_overlay()

void Fl_Glut_Window::draw_overlay ( ) [protected], [virtual]
You must implement this virtual function if you want to draw into the overlay.
The overlay is cleared before this is called. You should draw anything that is not clear using OpenGL. You must use
gl_color(i) to choose colors (it allocates them from the colormap using system-specific calls), and remember that
you are in an indexed OpenGL mode and drawing anything other than flat-shaded will probably not work.
Both this function and Fl_Gl_Window::draw() should check Fl_Gl_Window::valid() and set the same transformation.
If you don't your code may not work on other systems. Depending on the OS, and on whether overlays are real or
simulated, the OpenGL context may be the same or different between the overlay and main window.
Reimplemented from Fl_Gl_Window.
The documentation for this class was generated from the following files:

- glut.H
- glut_compatibility.cxx

32.49 Fl_Group Class Reference

The Fl_Group class is the FLTK container widget.
#include <Fl_Group.H>
Inheritance diagram for Fl_Group:
Public Member Functions

- **Fl_Widget ∗_ddfdesign_kludge ()**
  This is for forms compatibility only.

- **void add (Fl_Widget &)**
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (Fl_Widget ∗o)**
  See void Fl_Group::add(Fl_Widget &w)

- **void add_resizable (Fl_Widget &o)**
  Adds a widget to the group and makes it the resizable widget.

- **Fl_Widget ∗const ∗array () const**
  Returns a pointer to the array of children.

- **virtual Fl_Group ∗as_group ()**
  Returns an Fl_Group pointer if this widget is an Fl_Group.

- **void begin ()**
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **Fl_Widget ∗child (int n) const**
  Returns array()[n].

- **int children () const**
  Returns how many child widgets the group has.

- **void clear ()**
  Deletes all child widgets from memory recursively.

- **unsigned int clip_children ()**
  Returns the current clipping mode.

- **void clip_children (int c)**
  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- **virtual int delete_child (int n)**
  Removes the widget at index from the group and deletes it.

- **void end ()**
  Exactly the same as current(this->parent()).

- **int find (const Fl_Widget &o) const**
  See int Fl_Group::find(const Fl_Widget ∗w) const.

- **int find (const Fl_Widget ∗) const**
  Searches the child array for the widget and returns the index.

- **Fl_Group (int, int, int, int, const char ∗=0)**
  Creates a new Fl_Group widget using the given position, size, and label string.

- **void focus (Fl_Widget ∗W)**

- **void init_sizes ()**
  This is for forms compatibility only.

- **int handle (int)**
  Handles the specified event.

- **void init_sizes ()**
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &, int i)**
  The widget is removed from its current group (if any) and then inserted into this group.

- **void insert (Fl_Widget &o, Fl_Widget ∗before)**
  This does insert(w, find(before)).

- **void remove (Fl_Widget &)**
  Removes a widget from the group but does not delete it.

- **void remove (Fl_Widget ∗o)**
  Removes the widget o from the group.
• void remove (int index)
  
  Removes the widget at index from the group but does not delete it.

• Fl_Widget * resizable () const
  
  Returns the group's resizable widget.

• void resizable (Fl_Widget &o)
  
  Sets the group's resizable widget.

• void resizable (Fl_Widget *o)
  
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

• void resize (int, int, int, int)
  
  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
  
  Forc es 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

32.49.1 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.

32.49.2 Constructor & Destructor Documentation
32.49.2.1 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.

32.49.2.2 ~Fl_Group()

Fl_Group::~Fl_Group ( ) [virtual]
The destructor also deletes all the children.  
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.  
It is allowed that the Fl_Group and all of its children are automatic (local) variables, but you must declare the Fl_Group first, so that it is destroyed last.  
If you add static or automatic (local) variables to an Fl_Group, then it is your responsibility to remove (or delete) all such static or automatic child widgets before destroying the group - otherwise the child widgets’ destructors would be called twice!

32.49.3 Member Function Documentation

32.49.3.1 array()

Fl_Widget ∗const ∗ Fl_Group::array ( ) const
Returns a pointer to the array of children.  
Note
  This pointer is only valid until the next time a child is added or removed.

32.49.3.2 as_group()

virtual Fl_Group* Fl_Group::as_group ( ) [inline], [virtual]
Returns an Fl_Group pointer if this widget is an Fl_Group.  
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Group.  
If it returns non-NULL, then the widget in question is derived from Fl_Group, and you can use the returned pointer to access its children or other Fl_Group-specific methods.  
Example:
void my_callback (Fl_Widget ∗w, void ∗) {  
  FL_Group ∗g = w->as_group();  
  if (g)  
    printf ("This group has %d children\n", g->children());  
  else  
    printf ("This widget is not a group!\n");  
}
Return values

| NULL | if this widget is not derived from Fl_Group. |

Note
  This method is provided to avoid dynamic_cast.
See also

`Fl_Widget::as_window(), Fl_Widget::as_gl_window()`

Reimplemented from `Fl_Widget`.

32.49.3.3 `begin()`

```cpp
void Fl_Group::begin ( )
```

Sets the current group so you can build the widget tree by just constructing the widgets. `begin()` is automatically called by the constructor for `Fl_Group` (and thus for `Fl_Window` as well). `begin()` is exactly the same as `current(this)`. Don't forget to `end()` the group or window!

32.49.3.4 `bounds()`

```cpp
Fl_Rect * Fl_Group::bounds ( ) [protected]
```

Returns the internal array of widget sizes and positions. If the `bounds()` array does not exist, it will be allocated and filled with the current widget sizes and positions. The `bounds()` array stores the initial positions of widgets as `Fl_Rect`s. The size of the array is `children() + 2`.

- The first `Fl_Rect` is the group,
- the second is the resizable (clipped to the group),
- 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/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

32.49.3.5 `child()`

```cpp
Fl_Widget* Fl_Group::child ( int n ) const [inline]
```

Returns `array()[n]`. No range checking is done!
32.49.3.6 clear()

void Fl_Group::clear ( )
Deletes all child widgets from memory recursively.
This method differs from the remove() method in that it affects all child widgets and deletes them from memory.
The resizable() widget of the Fl_Group is set to the Fl_Group itself.

32.49.3.7 clip_children() [1/2]

unsigned int Fl_Group::clip_children ( ) [inline]
Returns the current clipping mode.
Returns
true, if clipping is enabled, false otherwise.

See also
void Fl_Group::clip_children(int c)

32.49.3.8 clip_children() [2/2]

void Fl_Group::clip_children ( int c ) [inline]
Controls whether the group widget clips the drawing of child widgets to its bounding box.
Set c to 1 if you want to clip the child widgets to the bounding box.
The default is to not clip (0) the drawing of child widgets.

32.49.3.9 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).

32.49.3.10 current() [2/2]

void Fl_Group::current ( Fl_Group * g ) [static]
Sets the current group.
See also
Fl_Group::current()

32.49.3.11 delete_child()

int Fl_Group::delete_child ( int index ) [virtual]
Removes the widget at index from the group and deletes it.
This method does nothing if index is out of bounds.
This method differs from the remove() method in that it deletes the widget from memory. Since this method is virtual it can be reimplemented in subclasses with additional requirements and consequences. See the documentation of subclasses.
Many subclasses don't need to reimplement this method.
Note

This method may refuse to remove and delete the widget if it is an essential part of the Fl_Group, for instance a scrollbar in an Fl_Scroll group. In this case the widget is neither removed nor deleted.

This method does not call init_sizes() or redraw(). This is left to user code if necessary.

Returns 0 if the widget was removed and deleted. Return values > 0 are reserved for use by FLTK core widgets. Return values < 0 are free to be used by user defined widgets.

Todo Reimplementation of Fl_Group::delete_widget(int) in more FLTK subclasses. This is not yet complete.

Parameters

| in | index | index of child to be removed |

Returns

success (0) or error code

Return values

| 0  | success |
| 1  | index out of range |
| 2  | widget not allowed to be removed (see note) |
| >2 | reserved for FLTK use |

Since

FLTK 1.4.0

Reimplemented in Fl_Scroll.

### 32.49.3.12 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:

```c++
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.


### 32.49.3.13 draw_child()

void Fl_Group::draw_child ( Fl_Widget & widget ) const [protected]

Forces a child to redraw.

This draws a child widget, if it is not clipped. The damage bits are cleared after drawing.

### 32.49.3.14 draw_children()

void Fl_Group::draw_children ( ) [protected]
Draws all children of the group. This is useful, if you derived a widget from Fl_Group and want to draw a special border or background. You can call draw_children() from the derived draw() method after drawing the box, border, or background.

### 32.49.3.15 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.

### 32.49.3.16 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.

### 32.49.3.17 focus()

```cpp
void Fl_Group::focus ( Fl_Widget * W ) [inline]
```

**Deprecated** This is for backwards compatibility only.

You should use W->take_focus() instead.

See also

```cpp
Fl_Widget::take_focus();
```

### 32.49.3.18 handle()

```cpp
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**

<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

```cpp
Fl_Event
```

Reimplemented from Fl_Widget.

Reimplemented in Fl_Glut_Window, Fl_Window, Fl_Tabs, Fl_Scroll, Fl_Help_View, Fl_Tile, Fl_Table_Row, Fl_Spinner, Fl_Tree, Fl_Text_Editor, Fl_Text_Display, and Fl_Table.
32.49.3.19 init_sizes()

```cpp
void Fl_Group::init_sizes ()
```

Resets the internal array of widget sizes and positions.
The Fl_Group widget keeps track of the original widget sizes and positions when resizing occurs so that if you resize
a window back to its original size the widgets will be in the correct places. If you rearrange the widgets in your group,
call this method to register the new arrangement with the Fl_Group that contains them.
If you add or remove widgets, this will be done automatically.

Note

The internal array of widget sizes and positions will be allocated and filled when the next `resize()` occurs. For
more information on the contents and structure of the `bounds()` array see `bounds()`.

See also

- `bounds()`
- `sizes()` (deprecated)

32.49.3.20 insert() [1/2]

```cpp
void Fl_Group::insert (Fl_Widget & o, int index)
```

The widget is removed from its current group (if any) and then inserted into this group.
It is put at index n - or at the end, if n >= `children()`. This can also be used to rearrange the widgets inside a group.

32.49.3.21 insert() [2/2]

```cpp
void Fl_Group::insert (Fl_Widget & o, Fl_Widget * before) [inline]
```

This does `insert(w, find(before))`. This will append the widget if `before` is not in the group.

32.49.3.22 remove() [1/3]

```cpp
void Fl_Group::remove (Fl_Widget & o)
```

Removes a widget from the group but does not delete it.
This method does nothing if the widget is not a child of the group.
This method differs from the `clear()` method in that it only affects a single widget and does not delete it from memory.

Note

If you have the child's index anyway, use `remove(int index)` instead, because this doesn't need a child lookup
in the group's table of children. This can be much faster, if there are lots of children.

32.49.3.23 remove() [2/3]

```cpp
void Fl_Group::remove (Fl_Widget * o) [inline]
```

Removes the widget o from the group.

See also

- `void remove(Fl_Widget&)`
32.49.3.24 remove() [3/3]

```cpp
text
do nothing if index is out of bounds.

Since

FLTK 1.3.0
```

32.49.3.25 resizable() [1/3]

```cpp
Fl_Widget* Fl_Group::resizable () const [inline]
```

Returns the group's resizable widget.

See void Fl_Group::resizable(Fl_Widget *o)

32.49.3.26 resizable() [2/3]

```cpp
void Fl_Group::resizable ( Fl_Widget & o ) [inline]
```

Sets the group's resizable widget.

See void Fl_Group::resizable(Fl_Widget *o)

32.49.3.27 resizable() [3/3]

```cpp
void Fl_Group::resizable ( Fl_Widget * o ) [inline]
```

The resizable widget defines both the resizing box and the resizing behavior of the group and its children. If the resizable is NULL the group's size is fixed and all of the widgets in the group remain a fixed size and distance from the top-left corner. This is the default for groups derived from `Fl_Window` and `Fl_Pack`. The resizable may be set to the group itself, in which case all of the widgets that are its direct children are resized proportionally. This is the default value for `Fl_Group`. The resizable widget defines the resizing box for the group, which could be the group itself or one of the group's direct children. When the group is resized it calculates a new size and position for all of its children. Widgets that are horizontally or vertically inside the dimensions of the box are scaled to the new size. Widgets outside the box are moved.

In these examples the gray area is the resizable:

![Figure 32.17 before resize](image-url)
It is possible to achieve any type of resize behavior by using an invisible \texttt{Fl\_Box} as the resizable and/or by using a hierarchy of \texttt{Fl\_Group} widgets, each with their own resizing strategies. See the \textit{How Does Resizing Work?} chapter for more examples and detailed explanation.

\textbf{Note}

The \texttt{resizable()} widget of a window can also affect the window's resizing behavior if \texttt{Fl\_Window::size\_range()} is not called. Please see \texttt{Fl\_Window::default\_size\_range()} for more information on how the default size range is calculated.

\textbf{See also}

\texttt{Fl\_Window::size\_range()}

\texttt{Fl\_Window::default\_size\_range()}

\subsection{32.49.3.28 \texttt{resize()}}

\begin{verbatim}
void Fl\_Group::resize (  
    int \textit{X},  
    int \textit{Y},  
    int \textit{W},  
    int \textit{H} ) [virtual]
\end{verbatim}

Resizes the \texttt{Fl\_Group} widget and all of its children. The \texttt{Fl\_Group} widget first resizes itself, and then it moves and resizes all its children according to the rules documented for \texttt{Fl\_Group::resizable(Fl\_Widget\*)}

\textbf{See also}

\texttt{Fl\_Group::resizable(Fl\_Widget\*)}

\texttt{Fl\_Group::resizable()}

\texttt{Fl\_Widget::resize(int,int,int,int)}

Reimplemented from \texttt{Fl\_Widget}.

Reimplemented in \texttt{Fl\_Tree}, \texttt{Fl\_Overlay\_Window}, \texttt{Fl\_Help\_View}, \texttt{Fl\_Window}, \texttt{Fl\_Tile}, \texttt{Fl\_Text\_Display}, \texttt{Fl\_Table}, \texttt{Fl\_Spinner}, \texttt{Fl\_Scroll}, and \texttt{Fl\_Input\_Choice}.

\subsection{32.49.3.29 \texttt{sizes()}}

\begin{verbatim}
int \* Fl\_Group::sizes ( ) [protected]
\end{verbatim}

Returns the internal array of widget sizes and positions.
For backward compatibility with FLTK versions before 1.4. The 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**

bounds()

### 32.49.3.30 update_child()

```cpp
void Fl_Group::update_child( Fl_Widget & widget ) const [protected]
```

Draws a child only if it needs it. This draws a child widget, if it is not clipped and if any damage() bits are set. The damage bits are cleared after drawing.

**See also**

Fl_Group::draw_child(Fl_Widget& widget) const

The documentation for this class was generated from the following files:

- Fl_Group.H
- Fl_Group.cxx
- forms_compatibility.cxx

### 32.50 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
32.51 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)**
  Sets the screen position of the dialog.

- **void resize (int xx, int yy, int ww, int hh)**
  Change the position and size of the dialog.

- **void show ()**
  Shows the Fl_Help_Dialog window.

- **void show (int argc, char ∗∗argv)**
  Shows the main Help Dialog Window Delegates call to encapsulated window_ void Fl_Window::show(int argc, char ∗∗argv) instance method.

- **Fl_Fontsize textsize ()**
  Sets or gets the default text size for the help view.

- **void textsize (Fl_Fontsize s)**
  Sets or gets the default text size for the help view.

- **void topline (const char ∗n)**
  Sets the top line in the Fl_Help_View widget to the named or numbered line.

- **void topline (int n)**
  Sets the top line in the Fl_Help_View widget to the named or numbered line.

- **const char ∗ value () const**
  The first form sets the current buffer to the string provided and reformats the text.

- **void value (const char ∗f)**
  The first form sets the current buffer to the string provided and reformats the text.

- **int visible ()**
  Returns 1 if the Fl_Help_Dialog window is visible.

- **int w ()**
  Returns the position and size of the help dialog.

- **int x ()**
  Returns the position and size of the help dialog.

- **int y ()**
  Returns the position and size of the help dialog.

- **~Fl_Help_Dialog ()**
  The destructor destroys the widget and frees all memory that has been allocated for the current file.
32.51.1 Detailed Description

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget. The Fl_Help_Dialog class is not part of the FLTK core library, but instead of fltk_images. Use -use-images when compiling with fltk-config.

![Figure 32.19 Fl_Help_Dialog]

32.51.2 Member Function Documentation

32.51.2.1 load()

```cpp
int Fl_Help_Dialog::load (const char * f)
```

Loads the specified HTML file into the Fl_Help_View widget. The filename can also contain a target name ("filename.html#target"). Always use forward slashes as path delimiters, MSWindows-style backslashes are not supported here.

Parameters

- `f` the name and path of an HTML file

Returns

0 on success, -1 on error

See also

- Fl_Help_View::load(), fl_load_uri()
Sets the internal Fl_Help_View instance text size.
Delegates call to encapsulated view: void Fl_Help_View::textsize(Fl_Fontsize s) instance method

32.51.2.4 value() [1/2]

const char * Fl_Help_Dialog::value () const
The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

32.51.2.5 value() [2/2]

void Fl_Help_Dialog::value {
    const char * v
}  

The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

The documentation for this class was generated from the following files:

- Fl_Help_Dialog.H
- Fl_Help_Dialog.cxx
- Fl_Help_Dialog_Dox.cxx

32.52 Fl_Help_Font_Stack Struct Reference

Public Member Functions

- size_t count () const
  Gets the current count of font style elements in the stack.
- Fl_Help_Font_Stack ()
  font stack construction, initialize attributes.
- void init (Fl_Font f, Fl_Fontsize s, Fl_Color c)
- void pop (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)
  Pops from the stack the font style triplet and calls fl_font() & fl_color() adequately.
- void push (Fl_Font f, Fl_Fontsize s, Fl_Color c)
  Pushes the font style triplet on the stack, also calls fl_font() & fl_color() adequately.
- void top (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)
  Gets the top (current) element on the stack.

Protected Attributes

- Fl_Help_Font_Style elts_[MAX_FL_HELP_FS_ELTS]
  font elements
- size_t nfonts_
  current number of fonts in stack

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

32.53 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.

32.53 Detailed Description

Fl_Help_View font stack element definition.
The documentation for this struct was generated from the following file:

- Fl_Help_View.H

32.54 Fl_Help_Link Struct Reference

Definition of a link for the html viewer.
#include <Fl_Help_View.H>

Public Attributes

- char filename [192]
  
  Reference filename.

- int h
  
  Height of link text.

- char name [32]
  
  Link target (blank if none)

- int w
  
  Width of link text.

- int x
  
  X offset of link text.

- int y
  
  Y offset of link text.

32.54.1 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

32.55 Fl_Help_Target Struct Reference

Fl_Help_Target structure.
#include <Fl_Help_View.H>
Public Attributes

- char name [32]
  Target name.
- int y
  Y offset of target.

32.55.1 Detailed Description

Fl_Help_Target structure.
The documentation for this struct was generated from the following file:

- Fl_Help_View.H

32.56 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.
- int leftline () const
  Gets the left position in pixels.
- void leftline (int)
  Scrolls the text to the indicated position, given a pixel column.
- void link (Fl_Help_Func *fn)
  This method assigns a callback function to use when a link is followed or a file is loaded (via Fl_Help_View::load()) that requires a different file or path.
- int load (const char *f)
  Loads the specified file.
- void resize (int, int, int, int)
  Resizes the help widget.
• int scrollbar_size () const
  Gets the current size of the scrollbars’ troughs, in pixels.
• void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.
• void select_all ()
  Selects all the text in the view.
• int size () const
  Gets the size of the help view.
• void size (int W, int H)
• Fl_Color textcolor () const
  Returns the current default text color.
• void textcolor (Fl_Color c)
  Sets the default text color.
• Fl_Font textfont () const
  Returns the current default text font.
• void textfont (Fl_Font f)
  Sets the default text font.
• Fl_Fontsize textsize () const
  Gets the default text size.
• void textsize (Fl_Fontsize s)
  Sets the default text size.
• const char ∗ title ()
  Returns the current document title, or NULL if there is no title.
• int topline () const
  Returns the current top line in pixels.
• void topline (const char ∗ n)
  Scrolls the text to the indicated position, given a named destination.
• void topline (int)
  Scrolls the text to the indicated position, given a pixel line.
• const char ∗ value () const
  Returns the current buffer contents.
• void value (const char ∗ val)
  Sets the current help text buffer to the string provided and reformats the text.
• ~Fl_Help_View ()
  Destroys the Fl_Help_View widget.

Protected Member Functions

• void draw ()
  Draws the Fl_Help_View widget.

Additional Inherited Members

32.56.1 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
Supported color names:
  - black, red, green, yellow, blue, magenta, fuchsia, cyan, 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 grave amp Aring aring Atilde atilde Auml auml
  - brvbar bull
  - Ccedil ccedil cedil cent copy curren
  - dagger deg divide
32.56.2 Constructor & Destructor Documentation

32.56.2.1 ~Fl_Help_View()

Fl_Help_View::~Fl_Help_View ()

Destroys the Fl_Help_View widget.

The destructor destroys the widget and frees all memory that has been allocated for the current document.

32.56.3 Member Function Documentation

32.56.3.1 find()

int Fl_Help_View::find ( 
    const char * s,
    int p = 0 )

Finds the specified string s at starting position p.

The argument p and the return value are offsets in Fl_Help_View::value(), counting from 0. If p is out of range, 0 is used.

The string comparison is simple but honors some special cases:

- the specified string s must be in UTF-8 encoding
- HTML tags in value() are filtered (not compared as such, they never match)
• HTML entities like '&lt;' or '&x#20ac;' are converted to Unicode (UTF-8)
• ASCII characters (7-bit, < 0x80) are compared case insensitive
• every newline (LF, '\n') in value() is treated like a single space
• all other strings are compared as-is (byte by byte)

Todo complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and do not (yet) match correctly ("<" matches "<" but "€" doesn't match "€", and "û" doesn't match "û")

Parameters

<table>
<thead>
<tr>
<th>in s</th>
<th>search string in UTF-8 encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>in p</td>
<td>starting position for search (0,...), Default = 0</td>
</tr>
</tbody>
</table>

Returns

the matching position or -1 if not found

32.56.3.2 leftline()

void Fl_Help_View::leftline (  
    int left )

Scrolls the text to the indicated position, given a pixel column.
If the given pixel value left is out of range, then the text is scrolled to the left or right side of the document, resp.

Parameters

| in left | left column number in pixels (0 = left side) |

32.56.3.3 link()

void Fl_Help_View::link (  
    Fl_Help_Func * fn ) [inline]

This method assigns a callback function to use when a link is followed or a file is loaded (via Fl_Help_View::load()) that requires a different file or path.
The callback function receives a pointer to the Fl_Help_View widget and the URI or full pathname for the file in question. It must return a pathname that can be opened as a local file or NULL:
const char * fn(Fl_Widget *w, const char *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.

32.56.3.4 load()

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()`

### 32.56.3.5 scrollbar_size() [1/2]

`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)`

### 32.56.3.6 scrollbar_size() [2/2]

`void Fl_Help_View::scrollbar_size ( int newSize ) [inline]`

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.

Normally you should not need this method, and should use `Fl::scrollbar_size(int)` instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.

Setting `newSize` to the special value of 0 causes the widget to track the global `Fl::scrollbar_size()`, which is the default.

**Parameters**

| in | `newSize` | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global `Fl::scrollbar_size()` |

**See also**

`Fl::scrollbar_size()`

### 32.56.3.7 topline() [1/2]

`void Fl_Help_View::topline ( const char * n )`
Scrolls the text to the indicated position, given a named destination.
Parameters

\textbf{in} \ n \ \textbf{target name}

\subsection{32.56.3.8 \ topline() [2/2]}

\begin{verbatim}
void Fl_Help_View::topline ( 
    int top )
\end{verbatim}

Scrolls the text to the indicated position, given a pixel line.
If the given pixel value \texttt{top} is out of range, then the text is scrolled to the top or bottom of the document, resp.

Parameters

\begin{tabular}{|l|c|}
\hline
\textbf{in} \ & \ \textbf{top} \ & \ top line number in pixels (0 = start of document) \\
\hline
\end{tabular}

\subsection{32.56.3.9 \ value()}

\begin{verbatim}
void Fl_Help_View::value ( 
    const char * val )
\end{verbatim}

Sets the current help text buffer to the string provided and reformats the text.
The provided character string \texttt{val} is copied internally and will be freed when \texttt{value()} is called again, or when the
widget is destroyed.
If \texttt{val} is NULL, then the widget is cleared.

The documentation for this class was generated from the following files:

- \texttt{Fl_Help_View.H}
- \texttt{Fl_Help_View.cxx}

\section{32.57 \ Fl_Hold_Browser Class Reference}

The \texttt{Fl_Hold_Browser} is a subclass of \texttt{Fl_Browser} which lets the user select a single item, or no items by clicking
on the empty space.

\begin{verbatim}
#include <Fl_Hold_Browser.H>
\end{verbatim}

Inheritance diagram for \texttt{Fl_Hold_Browser}:

```
Fl_Widget
   \-------------
   \         \nFl_Group
   \         \nFl_Browser_
   \         \nFl_Browser
   \         \nFl_Hold_Browser
```

Public Member Functions

- \texttt{Fl_Hold_Browser \ (int X, int Y, int W, int H, const char *L=0)}

  Creates a new \texttt{Fl_Hold_Browser} widget using the given position, size, and label string.
32.57.1 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 32.20 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.

32.57.2 Constructor & Destructor Documentation

32.57.2.1 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

32.58 Fl_Hor_Fill_Slider Class Reference

Inheritance diagram for Fl_Hor_Fill_Slider:

```
Fl_Widget
  |
  └── Fl_Valuator
       └── Fl_Slider
            └── Fl_Hor_Fill_Slider
```

Public Member Functions

- Fl_Hor_Fill_Slider (int X, int Y, int W, int H, const char *L=0)
Additional Inherited Members
The documentation for this class was generated from the following files:

- Fl_Hor_Fill_Slider.H
- Fl_Slider.cxx

32.59 Fl_Hor_Nice_Slider Class Reference

Inheritance diagram for Fl_Hor_Nice_Slider:

```
Fl_Widget
    |
    v
Fl_Valuator
    |
    v
Fl_Slider
    |
    v
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

32.60 Fl_Hor_Slider Class Reference

Horizontal Slider class.

```
#include <Fl_Hor_Slider.H>
```

Inheritance diagram for Fl_Hor_Slider:

```
Fl_Widget
    |
    v
Fl_Valuator
    |
    v
Fl_Slider
    |
    v
Fl_Hor_Slider
```

Public Member Functions

- **Fl_Hor_Slider** (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Hor_Slider widget using the given position, size, and label string.
Additional Inherited Members

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

Fl_Decade_Slider Class Reference

Inheritance diagram for Fl_Decade_Slider:

```
Fl_Widget
    ↓
Fl_Valuator
    ↓
Fl_Slider
    ↓
Fl_Value_Slider
    ↓
Fl_Hor_Value_Slider
```

Public Member Functions

- Fl_Decade_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

Fl_Decade_Slider Class Reference

Base class for image caching, scaling and drawing.

```
#include <Fl_Image.H>
```

Inheritance diagram for Fl_Image:

Generated by Doxygen
Public Member Functions

- virtual class Fl_Shared_Image * as_shared_image ()
  
  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)
  
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
  
  Creates a copy of the image in the same size.
- virtual Fl_Image * copy (int W, int H) const
  
  Creates a resized copy of the image.
- int count () const
  
  Returns the number of data values associated with the image.
- int d () const
  
  Returns the image depth.
- const char * const * data () const
  
  Returns a pointer to the current image data array.
- int data_h () const
  
  Returns the height of the image data.
- int data_w () const
  
  Returns the width of the image data.
- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
  
  Draws the image to the current drawing surface.
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws the image to the current drawing surface with a bounding box.
- int fail () const
  
  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)
  
  The constructor creates an empty image with the specified width, height, and depth.
- int h () const
  
  Returns the current image drawing height in FLTK units.
- void inactive ()
  
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- virtual void label (Fl_Menu_Item *m)
  
  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget *w)
This method is an obsolete way to set the image attribute of a widget or menu item.

- **int ld() const**
  
  Returns the current line data size in bytes.

- **virtual void release()**
  
  Releases an Fl_Image - the same as 'delete this'.

- **virtual void scale(int width, int height, int proportional=1, int can Expand=0)**
  
  Sets the drawing size of the image.

- **virtual void uncache()**
  
  If the image has been cached for display, delete the cache data.

- **int w() const**
  
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image()**
  
  The destructor is a virtual method that frees all memory used by the image.

**Static Public Member Functions**

- **static Fl_Labeltype define_FL_IMAGE_LABEL()**

- **static Fl_RGB_Scaling RGB_scaling()**

  Returns the currently used RGB image scaling method.

- **static void RGB_scaling(Fl_RGB_Scaling)**

  Sets the RGB image scaling method used for copy(int, int).

- **static Fl_RGB_Scaling scaling_algorithm()**

  Gets what algorithm is used when resizing a source image to draw it.

- **static void scaling_algorithm(Fl_RGB_Scaling algorithm)**

  Sets what algorithm is used when resizing a source image to draw it.

**Static Public Attributes**

- **static const int ERR_FILE_ACCESS = -2**

- **static const int ERR_FORMAT = -3**

- **static const int ERR_NO_IMAGE = -1**

- **static bool register_images_done = false**

  True after fl_register_images() was called, false before.

**Protected Member Functions**

- **void d(int D)**

  Sets the current image depth.

- **void data(const char * const *p, int c)**

  Sets the current data pointer and count of pointers in the array.

- **void draw_empty(int X, int Y)**

  The protected method draw_empty() draws a box with an X in it.

- **int draw_scaled(int X, int Y, int W, int H)**

  Draw the image to the current drawing surface rescaled to a given width and height.

- **void h(int H)**

  Sets the height of the image data.

- **void ld(int LD)**

  Sets the current line data size in bytes.

- **void w(int W)**

  Sets the width of the image data.
Static Protected Member Functions

- static void labeltype (const Fl_Label *lo, int tx, int ty, int lw, int lh, Fl_Align la)
- static void measure (const Fl_Label *lo, int &lw, int &lh)

Friends

- class Fl_Graphics_Driver

32.62.1 Detailed Description

Base class for image caching, scaling and drawing.

Fl_Image is the base class used for caching, scaling and drawing all kinds of images in FLTK. This class keeps track of common image data such as the pixels, colormap, width, height, and depth. Virtual methods are used to provide type-specific image handling.

Each image possesses two (width, height) pairs:

1. The width and height of the raw image data are returned by data_w() and data_h(). These values are set when the image is created and remain unchanged.

2. The width and height of the area filled by the image when it gets drawn are returned by w() and h(). These values are equal to data_w() and data_h() when the image is created and can be changed by the scale() member function.

Since the Fl_Image class does not support image drawing by itself, calling the Fl_Image::draw() method results in a box with an X in it being drawn instead.

32.62.2 Constructor & Destructor Documentation

32.62.2.1 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.

32.62.3 Member Function Documentation

32.62.3.1 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.
32.62.3.2 color_average()

```cpp
void Fl_Image::color_average (  
    Fl_Color c,  
    float i  ) [virtual]
```

The `color_average()` method averages the colors in the image with the provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory. Reimplemented in `Fl_Tiled_Image`, `Fl_SVG_Image`, `Fl_Shared_Image`, `Fl_Pixmap`, and `Fl_RGB_Image`.

32.62.3.3 copy() [1/2]

```cpp
Fl_Image* Fl_Image::copy ( ) const [inline]
```

Creates a copy of the image in the same size. The new image should be released when you are done with it. This does exactly the same as `Fl_Image::copy(int W, int H) const` where `W` and `H` are the width and height of the source image, respectively. This applies also to all subclasses of `Fl_Image` in the FLTK library. The following two `copy()` calls are equivalent:

```cpp
Fl_Image *img1 = new Fl_Image(...);
// ...  
Fl_Image *img2 = img1->copy();  
Fl_Image *img3 = img1->copy(img1->w(), img1->h());
```

For details see `Fl_Image::copy(int w, int h) const`.

See also

`Fl_Image::release()`

**Note**

Since FLTK 1.4.0 this method is ‘const’. If you derive your own class from `Fl_Image` or any subclass your overridden methods of `Fl_Image::copy(int w, int h) const` and `Fl_Image::copy(int, int) const` must also be ‘const’ for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been ‘const’.

32.62.3.4 copy() [2/2]

```cpp
Fl_Image * Fl_Image::copy (  
    int W,  
    int H ) const [virtual]
```

Creates a resized copy of the image. The new image should be released when you are done with it. Note: since FLTK 1.4.0 you can use `Fl_Image::release()` for all types of images (i.e. all subclasses of `Fl_Image`) instead of `operator delete` for `Fl_Image`'s and `Fl_Image::release()` for `Fl_Shared_Image`'s. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by `Fl_Image::RGB_scaling()`.

For the new image the following equations are true:

- `w() == data_w() == W`
- `h() == data_h() == H`

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of `Fl_RGB_Image`, `Fl_SVG_Image`, `Fl_Pixmap`, `Fl_Bitmap`, `Fl_Tiled_Image`, and `Fl_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `Fl_GIF_Image` is an object of class `Fl_Pixmap`.

Generated by Doxygen
Parameters

| in | W, H | Requested width and height of the new image |

Note
Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_image or any subclass your overridden methods of 'Fl_image::copy() const' and 'Fl_image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented in Fl_Tiled_image, Fl_SVG_image, Fl_Shared_image, Fl_Pixmap, Fl_RGB_image, and Fl_Bitmap.

32.62.3.5 count()

int Fl_image::count() const [inline]
Returns the number of data values associated with the image.
The value will be 0 for images with no associated data, 1 for bitmap and color images, and greater than 2 for pixmap images.
See also
  data()

32.62.3.6 d()

int Fl_image::d() const [inline]
Returns the image depth.
The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

32.62.3.7 data() [1/2]

const char* const* Fl_image::data() const [inline]
Returns a pointer to the current image data array.
There can be 0, 1, or more pointers to actual image data in an image.
Use the count() method to find the size of the data array. You must not dereference the data() pointer if count() equals zero.

Note
  data() may return NULL.

Example:
Fl_RGB_image has exactly one pointer which points at the R, G, B [A] data array of the image. The total size of this array depends on several attributes like data_w(), data_h(), d() and ld() and is basically data_w() * data_h() * d() but there are exceptions if ld() is non-zero: see description of ld(). Since FLTK 1.4.0 w() and h() are no longer significant for the image data size if scale() has been called on the image to set a different display size.
Other image types have different numbers and types of data pointers which are implementation details and not documented here.
See also
  count(), w(), h(), data_w(), data_h(), d(), ld()
32.62.3.8  data() [2/2]

void Fl_Image::data (const char *const * p, int c) [inline], [protected]

Sets the current data pointer and count of pointers in the array.
There can be 0, 1, or more pointers to actual image data in an image.

See also
const char* const* data(), count(), w(), h(), data_w(), data_h(), d(), ld() 

32.62.3.9  desaturate()

void Fl_Image::desaturate () [virtual]
The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original
image data in memory.
Reimplemented in Fl_Tiled_Ima ge, Fl_SVG_Ima ge, Fl_Shared_Ima ge, Fl_Pixmap, and Fl_RGB_Ima ge.

32.62.3.10  draw() [1/2]

void Fl_Image::draw (int X, int Y) [inline]

Draws the image to the current drawing surface.

Parameters

**X, Y** specify the upper-lefthand corner of the image.

32.62.3.11  draw() [2/2]

void Fl_Image::draw (int X, int Y, int W, int H, 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_Tiled_Ima ge, Fl_SVG_Ima ge, Fl_Shared_Ima ge, Fl_Pixmap, Fl_RGB_Ima ge, and
Fl_Bitmap.

32.62.3.12  draw_empty()

void Fl_Image::draw_empty (int X, int Y) [protected]
The protected method `draw_empty()` draws a box with an X in it. It can be used to draw any image that lacks image data.

### 32.62.3.13 draw_scaled()

```cpp
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**

| X, Y | position of the image’s top-left |
| W, H | width and height for the drawn image |

**Returns**

1

**Deprecated** Only for API compatibility with FLTK 1.3.4.

### 32.62.3.14 fail()

```cpp
int Fl_Image::fail ( ) const
```

Returns a value that is not 0 if there is currently no image available.

**Example use:**

```cpp
// [..]
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**

Image load failure if non-zero

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the image was loaded successfully</td>
</tr>
<tr>
<td>ERR_NO_IMAGE</td>
<td>no image was found</td>
</tr>
<tr>
<td>ERR_FILE_ACCESS</td>
<td>there was a file access related error (errno should be set)</td>
</tr>
<tr>
<td>ERR_FORMAT</td>
<td>image decoding failed</td>
</tr>
</tbody>
</table>

### 32.62.3.15 h() [1/2]

```cpp
int Fl_Image::h ( ) const [inline]
```

Returns the current image drawing height in FLTK units.
The values of \( h() \) and \( \text{data}_h() \) are identical unless \( \text{scale}() \) has been called after which they may differ.

### 32.62.3.16 \( h() \) [2/2]

```c
void Fl_Image::h ( int H ) [inline], [protected]
```

Sets the height of the image data.
This protected function sets both image heights: the height of the image data returned by \( \text{data}_h() \) and the image drawing height in FLTK units returned by \( h() \).

### 32.62.3.17 \( \text{inactive}() \)

```c
void Fl_Image::inactive ( ) [inline]
```

The \( \text{inactive}() \) method calls `color_average(FLBACKGROUND_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.

### 32.62.3.18 \( \text{label}() \) [1/2]

```c
void Fl_Image::label ( Fl_Menu_Item * m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

**Deprecated** Please use \( \text{Fl_Menu_Item::image()} \) instead.

Reimplemented in \( \text{Fl_Pixmap, Fl_RGB_IMAGE, and Fl_Bitmap} \).

### 32.62.3.19 \( \text{label}() \) [2/2]

```c
void Fl_Image::label ( Fl_Widget * widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

**Deprecated** Please use \( \text{Fl_Widget::image()} \) or \( \text{Fl_Widget::deimage()} \) instead.

Reimplemented in \( \text{Fl_Pixmap, Fl_RGB_IMAGE, and Fl_Bitmap} \).

### 32.62.3.20 \( \text{ld}() \) [1/2]

```c
int Fl_Image::ld ( ) const [inline]
```

Returns the current line data size in bytes.

See also
- \( \text{ld(int)} \)

### 32.62.3.21 \( \text{ld}() \) [2/2]

```c
void Fl_Image::ld ( int LD ) [inline], [protected]
```

Sets the current line data size in bytes.
Color images may contain extra data (padding) that is included after every line of color image data and is normally not present.
If \( LD \) is zero, then line data size is assumed to be \( \text{data}_w() \times d() \) bytes.
If \( LD \) is non-zero, then it must be positive and larger than \( \text{data}_w() \times d() \) to account for the extra data per line.
32.62.3.22  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 and its subclasses this virtual method is reimplemented and maintains
shared images.
This virtual method makes it possible to destroy all image types in the same way by calling
image->release();
Reasoning: If you have an Fl_Image * base class pointer and don’t know if the object is one of the class
Fl_Shared_Image or any other subclass of Fl_Image (for instance Fl_RGB_Image) then you can’t just use oper-
ator 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.

32.62.3.23  RGB_scaling()

void Fl_Image::RGB_scaling ( Fl_RGB_Scaling method ) [static]
Sets the RGB image scaling method used for copy(int, int).
Applies to all RGB images, defaults to FL_RGB_SCALING_NEAREST.

32.62.3.24  scale()

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

| width, height | maximum values, in FLTK units, that w() and h() should return |
| proportional  | if not null, keep the values returned by w() and h() proportional to data_w() and data_h() |
| can_expand    | if null, the values returned by w() and h() will not be larger than data_w() and data_h(), respectively |

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

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

32.62.3.25 scaling_algorithm()

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

32.62.3.26 uncache()

void Fl_Image::uncache ( ) [virtual]
If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented in Fl_Shared_Image, Fl_Pixmap, Fl_RGB_Image, and Fl_Bitmap.

32.62.3.27 w() [1/2]

int Fl_Image::w ( ) const [inline]
Returns the current image drawing width in FLTK units.
The values of w() and data_w() are identical unless scale() has been called after which they may differ.

32.62.3.28 w() [2/2]

void Fl_Image::w ( int W ) [inline], [protected]
Sets the width of the image data.
This protected function sets both image widths: the width of the image data returned by data_w() and the image
drawing width in FLTK units returned by w().
The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

32.63 Fl_Image_Reader Class Reference

Public Member Functions

- int error () const
- const char * name () const
- int open (const char *filename)
- int open (const char *imagename, const unsigned char *data)
- int open (const char *imagename, const unsigned char *data, const size_t datasize)
- unsigned char read_byte ()
- unsigned int read_dword ()
- int read_long ()

Generated by Doxygen
unsigned short read_word ()
void seek (unsigned int n)
void skip (unsigned int n)
long tell () const

The documentation for this class was generated from the following files:
  - Fl_Image_Reader.h
  - Fl_Image_Reader.cxx

32.64  Fl_Image_Surface Class Reference

Directs all graphics requests to an Fl_Image.

#include <Fl_Image_Surface.H>

Inheritance diagram for Fl_Image_Surface:

```
Fl_Surface_Device
   |
   v
Fl_Widget_Surface
   |
   v
Fl_Image_Surface
```

Public Member Functions

- **Fl_Image_Surface** (int w, int h, int high_res=0, Fl_Offscreen off=0)
  Constructor with optional high resolution.
- **Fl_Shared_Image * highres_image ()**
  Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object.
- **Fl_RGB_Image * image ()**
  Returns a depth 3 image made of all drawings sent to the Fl_Image_Surface object.
- **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

32.64.1 Detailed Description

Directs all graphics requests to an Fl_Image.

After creation of an Fl_Image_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the image. It's possible to draw widgets (using Fl_Image_Surface::draw()) or to use any of the Drawing functions or the Color & Font functions. Finally, call image() on the object to obtain a newly allocated Fl_RGB_Image object. Fl_Gl_Window objects can be drawn in the image as well.

Usage example:

```c++
// this is the widget that you want to draw into an image
Fl_Widget *g = ...;
// create an Fl_Image_Surface object
Fl_Image_Surface *image_surface = new Fl_Image_Surface(g->w(), g->h());
// direct all further graphics requests to the image
Fl_Surface_Device::push_current(image_surface);
// draw a white background
fl_color(FL_WHITE);
fl_rectf(0, 0, g->w(), g->h());
// 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;
```

32.64.2 Constructor & Destructor Documentation

32.64.2.1 Fl_Image_Surface()

**Fl_Image_Surface::Fl_Image_Surface**

```c++
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,h</td>
<td>Width and height of the resulting image. The value of the high_res parameter controls whether w and h are interpreted as pixels or FLTK units.</td>
</tr>
<tr>
<td>high_res</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::screen_scale(int)): 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>off</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)

32.64.3 Member Function Documentation
32.64.3.1 highres_image()

Fl_Shared_Image * Fl_Image_Surface::highres_image ()

Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object. The Fl_Image_Surface object should have been constructed with Fl_Image_Surface(W, H, 1). The returned Fl_Shared_Image object is scaled to a size of WxH FLTK units and may have a pixel size larger than these values. The returned object should be deallocated with Fl_Shared_Image::release() after use.

Deprecated  Use image() instead.

Version

1.4 (1.3.4 for MacOS platform only)

32.64.3.2 image()

Fl_RGB_Image * Fl_Image_Surface::image ()

Returns a depth 3 image made of all drawings sent to the Fl_Image_Surface object. The returned object contains its own copy of the RGB data. The caller is responsible for deleting the image.

32.64.3.3 offscreen()

Fl_Offscreen Fl_Image_Surface::offscreen ()

Returns the Fl_Offscreen object associated to the image surface. The returned Fl_Offscreen object is deleted when the Fl_Image_Surface object is deleted, unless the Fl_Image_Surface was constructed with non-null Fl_Offscreen argument.

32.64.3.4 origin() [1/2]

void Fl_Image_Surface::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>
</tr>
</thead>
<tbody>
<tr>
<td>If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin.</td>
<td></td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.

32.64.3.5 origin() [2/2]

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>
<tbody>
<tr>
<td>Horizontal and vertical positions in the drawing surface of the desired origin of graphics.</td>
<td></td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.
### printable_rect()

```cpp
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`.

### rescale()

```cpp
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

### set_current()

```cpp
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, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current()`.

Note

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device() -> set_current()`; Don't do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current()`.

Reimplemented from `Fl_Surface_Device`.

### translate()

```cpp
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.cxx

## 32.65 Fl_Input Class Reference

This is the FLTK text input widget.

```c
#include <Fl_Input.H>
```

**Inheritance diagram for Fl_Input:**

<table>
<thead>
<tr>
<th>Fl_Widget</th>
<th>Fl_Input</th>
<th>Fl_Input_</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_File_Input</td>
<td>Fl_Float_Input</td>
<td>Fl_Int_Input</td>
</tr>
<tr>
<td>Fl_Multiline_Input</td>
<td>Fl_Output</td>
<td>Fl_Secret_Input</td>
</tr>
<tr>
<td>Fl_Spinner::Fl_Spinner_Input</td>
<td>Fl_Multiline_Output</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

- **int handle_key ()**
  
  Handles a keystroke.

### Friends

- class Fl_Cocoa_Screen_Driver
- class Fl_Screen_Driver

### Additional Inherited Members

#### 32.65.1 Detailed Description

This is the FLTK text input widget. It displays a single line of text and lets the user edit it. Normally it is drawn with an inset box and a white background. The text may contain any characters, and will correctly display any UTF text, using ^X notation for unprintable control characters. It assumes the font can draw any characters of the used scripts, which is true for standard fonts under Windows and Mac OS X. Characters can be input using the keyboard or the character palette/map. Character composition is done using dead keys and/or a compose key as defined by the operating system.

<table>
<thead>
<tr>
<th>Mouse button 1</th>
<th>Moves the cursor to this point. Drag selects characters. Double click selects words. Triple click selects all line. Shift+click extends the selection. When you select text it is automatically copied to the selection buffer.</th>
</tr>
</thead>
</table>

Generated by Doxygen
Mouse button 2 | 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.

Mouse button 3 | Currently acts like button 1.

Backspace | Deletes one character to the left, or deletes the selected region.

Delete | Deletes one character to the right, or deletes the selected region. Combine with Shift for equivalent of ^X (copy+cut).

Enter | May cause the callback, see when().

Table 32.213 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. (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. (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. Copy the selection to the clipboard and delete it. (If there's no selection, Shift-Delete acts like Delete)</td>
</tr>
<tr>
<td>^Z</td>
<td>Command-Z</td>
<td>Undo. 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>Arrow Keys</td>
<td>Arrow Keys</td>
<td>Standard cursor movement. 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. 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. 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. In single line input, moves to start of line. In multiline input, moves to start of top line. Can be combined with Shift to extend selection.</td>
</tr>
<tr>
<td>Ctrl-End</td>
<td>Command-End, Command-PgDn, Ctrl-Right</td>
<td>Move to bottom of document/field. In single line input, moves to end of line. In multiline input, moves to end of last line. Can be combined with Shift to extend selection.</td>
</tr>
</tbody>
</table>
32.65.2 Constructor & Destructor Documentation

### 32.65.2.1 Fl_Input()

```cpp
Fl_Input::Fl_Input (int X, int Y, int W, int H, const char * l = 0)
```

Creates a new Fl_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

32.65.3 Member Function Documentation

#### 32.65.3.1 draw()

```cpp
void Fl_Input::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

#### 32.65.3.2 handle()

```cpp
int Fl_Input::handle (int event ) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

| in | 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_Secret_Input,* and *Fl_Spinner::Fl_Spinner_Input.*

### 32.65.3.3 handle_key()

```cpp
int Fl_Input::handle_key ( ) [protected]
```

Handles a keystroke.

This protected method handles a keystroke in an *Fl_Input* or derived class. It handles compose key sequences and can also be used e.g. in *Fl_Multiline_Input,* *Fl_Float_Input* and several more derived classes. The details are way too complicated to be documented here and can be changed as required. If in doubt, please consult the source code.

Returns

1 if the keystroke is handled by us, 0 if not.

The documentation for this class was generated from the following files:

- Fl_Input.H
- Fl_Input.cxx

### 32.66 Fl_Input_ Class Reference

This class provides a low-overhead text input field.

```
#include <Fl_Input_.H>
```

Inheritance diagram for Fl_Input_

```
Fl_Widget
Fl_Input
Fl_File_Input Fl_Float_Input Fl_Int_Input Fl_Multiline_Input Fl_Output Fl_Secret_Input Fl_Spinner::Fl_Spinner_Input
Fl_Multiline_Output
```

#### Public Member Functions

- int **append** (const char *t, int l=0, char keep_selection=0)
  
  Append text at the end.

- 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** ()

Generated by Doxygen
Deletes the current selection.

- **int cut (int a, int b)**
  
  Deletes all characters between index \texttt{a} and \texttt{b}.

- **int cut (int n)**
  
  Deletes the next \texttt{n} bytes rounded to characters before or after the cursor.

- **\texttt{Fl_Input_} (int, \texttt{int}, \texttt{int}, \texttt{int}, \texttt{const char} \* = \texttt{0})**
  
  Creates a new \texttt{Fl_Input_} widget.

- **unsigned int index (int i) const**
  
  Returns the character at index \texttt{i}.

- **int input_type () const**
  
  Gets the input field type.

- **void input_type (int t)**
  
  Sets the input field type.

- **int insert (const char \* t, int l=0)**
  
  Inserts text at the cursor position.

- **int mark () const**
  
  Gets the current selection mark.

- **int mark (int m)**
  
  Sets the current selection mark.

- **int maximum_size () const**
  
  Gets the maximum length of the input field in characters.

- **void maximum_size (int m)**
  
  Sets the maximum length of the input field in characters.

- **int position () const**
  
  Gets the position of the text cursor.

- **int position (int p)**
  
  Sets the cursor position and mark.

- **int position (int p, int m)**
  
  Sets the index for the cursor and mark.

- **int readonly () const**
  
  Gets the read-only state of the input field.

- **void readonly (int b)**
  
  Sets the read-only state of the input field.

- **int replace (int b, int e, const char \* text, int ilen=0)**
  
  Deletes text from \texttt{b} to \texttt{e} and inserts the new string \texttt{text}.

- **void resize (int, int, int, int)**
  
  Changes the size of the widget.

- **int shortcut () const**
  
  Return the shortcut key associated with this widget.

- **void shortcut (int s)**
  
  Sets the shortcut key associated with this widget.

- **int size () const**
  
  Returns the number of bytes in \texttt{value()}. 

- **void size (int W, int H)**
  
  Sets the width and height of this widget.

- **int static_value (const char \* s)**
  
  Changes the widget text.

- **int static_value (const char \* s, int i)**
  
  Changes the widget text.

- **int tab_nav () const**
  
  Gets whether the Tab key causes focus navigation in multiline input fields or not.
• void tab_nav (int val)
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
• Fl_Color textcolor () const
  Gets the color of the text in the input field.
• void textcolor (Fl_Color n)
  Sets the color of the text in the input field.
• Fl_Font textfont () const
  Gets the font of the text in the input field.
• void textfont (Fl_Font s)
  Sets the font of the text in the input field.
• Fl_Fontsize textsize () const
  Gets the size of the text in the input field.
• void textsize (Fl_Fontsize s)
  Sets the size of the text in the input field.
• int undo ()
  Undoes previous changes to the text buffer.
• const char * value () const
  Returns the text displayed in the widget.
• int value (const char * v)
  Changes the widget text.
• int value (const char *, int)
  Changes the widget text.
• int wrap () const
  Gets the word wrapping state of the input field.
• void wrap (int b)
  Sets the word wrapping state of the input field.
• ~Fl_Input_ ()
  Destroys the widget.

Protected Member Functions

• void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.
• void handle_mouse (int, int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.
• int handletext (int e, int, int, int, int)
  Handles all kinds of text field related events.
• int line_end (int i) const
  Finds the end of a line.
• int line_start (int i) const
  Finds the start of a line.
• int linesPerPage ()
• void maybe_do_callback ()
• int up_down_position (int, int keepmark=0)
  Moves the cursor to the column given by up_down_pos.
• int word_end (int i) const
  Finds the end of a word.
• int word_start (int i) const
  Finds the start of a word.
• int xscroll () const
• int yscroll () const
• void yscroll (int yOffset)
Additional Inherited Members

32.66.1 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_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)

32.66.2 Constructor & Destructor Documentation

32.66.2.1 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>

32.66.2.2 ~Fl_Input()

```c
Fl_Input_::~Fl_Input_ ( )
```

Destroys the widget.

The destructor clears all allocated buffers and removes the widget from the parent Fl_Group.

32.66.3 Member Function Documentation
32.66.3.1 append()

```cpp
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 move the new position or mark.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>t</code></th>
<th>text that will be appended</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>l</code></td>
<td>length of text, or 0 if the string is terminated by <code>nul</code>.</td>
</tr>
<tr>
<td>in</td>
<td><code>keep_selection</code></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

32.66.3.2 copy()

```cpp
int Fl_Input_::copy (  
  int clipboard  )
```

Put the current selection into the clipboard.
This function copies the current selection between `mark()` and `position()` into the specified `clipboard`. This does not replace the old clipboard contents if `position()` and `mark()` are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.

**Parameters**

| `clipboard` | the clipboard destination 0 or 1 |

**Returns**

- 0 if no text is selected, 1 if the selection was copied

See also

- `Fl::copy(const char *, int, int)`

32.66.3.3 copy_cuts()

```cpp
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()`
32.66.3.4 cursor_color() [1/2]

```
Fl_Color Fl_Input_::cursor_color ( ) const [inline]
```

Gets the color of the cursor.

Returns

the current cursor color

32.66.3.5 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 |

32.66.3.6 cut() [1/3]

```
int Fl_Input_::cut ( ) [inline]
```

Deletes the current selection.
This function deletes the currently selected text without storing it in the clipboard. To use the clipboard, you may call copy() first or copy_cuts() after this call.

Returns

0 if no data was copied

32.66.3.7 cut() [2/3]

```
int Fl_Input_::cut ( int a, int b ) [inline]
```

Deletes all characters between index a and b.
This function deletes the currently selected text without storing it in the clipboard. To use the clipboard, you may call copy() first or copy_cuts() after this call.

Parameters

| a,b | range of bytes rounded to full characters and clamped to the buffer |

Returns

0 if no data was copied

32.66.3.8 cut() [3/3]

```
int Fl_Input_::cut ( int n ) [inline]
```

Deletes the next n bytes rounded to characters before or after the cursor.
This function deletes the currently selected text without storing it in the clipboard. To use the clipboard, you may call `copy()` first or `copy_cuts()` after this call.

Parameters

- \( n \) number of bytes rounded to full characters and clamped to the buffer. A negative number will cut characters to the left of the cursor.

Returns

- 0 if no data was copied

### 32.66.3.9 `drawtext()`

```cpp
void Fl_Input_::drawtext ( int X, int Y, int W, int H ) [protected]
```

Draws the text in the passed bounding box. If `damage() & FL_DAMAGE_ALL` is true, this assumes the area has already been erased to `color()`. Otherwise it does minimal update and erases the area itself.

Parameters

- \( X, Y, W, H \) area that must be redrawn

### 32.66.3.10 `handle_mouse()`

```cpp
void Fl_Input_::handle_mouse ( int X, int Y, int , int , int drag = 0 ) [protected]
```

Handles mouse clicks and mouse moves.

**Todo** Add comment and parameters

### 32.66.3.11 `handletext()`

```cpp
int Fl_Input_::handletext ( int event, int X, int Y, int W, int H ) [protected]
```

Handles all kinds of text field related events. This is called by derived classes.

**Todo** Add comment and parameters
32.66.3.12  `index()`

```cpp
unsigned int Fl_Input_::index ( int i ) const
```

Returns the character at index `i`. This function returns the UTF-8 character at `i` as a ucs4 character code.

**Parameters**

- `i`: index into the value field

**Returns**

the character at index `i`

32.66.3.13  `input_type()` [1/2]

```cpp
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.

**Returns**

the current input type

32.66.3.14  `input_type()` [2/2]

```cpp
void Fl_Input_::input_type ( int t ) [inline]
```

Sets the input field type. A `redraw()` is required to reformat the input field.

**Parameters**

- `t`: new input type

32.66.3.15  `insert()`

```cpp
int Fl_Input_::insert ( const char * t, int l = 0 ) [inline]
```

Inserts text at the cursor position. This function inserts the string in `t` at the cursor `position()` and moves the new position and mark to the end of the inserted text.

**Parameters**

- `t`: text that will be inserted
- `l`: length of text, or 0 if the string is terminated by `nul`

**Returns**

0 if no text was inserted
32.6.3.16 line_end()

int Fl_Input_::line_end (int i) const [protected]

Finds the end of a line.
This call calculates the end of a line based on the given index i.

Parameters

\[ \text{in } i \text{ starting index for the search} \]

Returns

end of the line

32.6.3.17 line_start()

int Fl_Input_::line_start (int i) const [protected]

Finds the start of a line.
This call calculates the start of a line based on the given index i.

Parameters

\[ \text{in } i \text{ starting index for the search} \]

Returns

start of the line

32.6.3.18 mark() [1/2]

int Fl_Input_::mark ( ) const [inline]

Gets the current selection mark.

Returns

index into the text

32.6.3.19 mark() [2/2]

int Fl_Input_::mark (int m) [inline]

Sets the current selection mark.
mark(n) is the same as position(position(),n).

Parameters

\[ m \text{ new index of the mark} \]

Returns

0 if the mark did not change
See also

- `position()`, `position(int, int)`

32.66.3.20 maximum_size() [1/2]

```cpp
int Fl_Input_::maximum_size ( ) const [inline]
```

Gets the maximum length of the input field in characters.

See also

- `maximum_size(int)`

32.66.3.21 maximum_size() [2/2]

```cpp
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).

32.66.3.22 position() [1/3]

```cpp
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)`

32.66.3.23 position() [2/3]

```cpp
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)`.  

Generated by Doxygen
32.66.3.24 position() [3/3]

```cpp
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)

32.66.3.25 readonly() [1/2]

```cpp
int Fl_Input_::readonly ( ) const [inline]
```

Gets the read-only state of the input field.

Returns

- non-zero if this widget is read-only

32.66.3.26 readonly() [2/2]

```cpp
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

32.66.3.27 replace()

```cpp
int Fl_Input_::replace ( 
    int b,
    int e,
    const char * text,
    int ilen = 0 )
```

Deletes text from b to e and inserts the new string text. All changes to the text buffer go through this function. It deletes the region between b and e (either one may be less or equal to the other), and then inserts the string text at that point and moves the mark() and position() to the end of the insertion. Does the callback if when() & FL_WHEN_CHANGED and there is a change.
Set b and e equal to not delete anything. Set text to NULL to not insert anything. ilen can be zero or strlen(text), which saves a tiny bit of time if you happen to already know the length of the insertion, or can be used to insert a portion of a string. If ilen is zero, strlen(text) is used instead. b and e are clamped to the 0..size() range, so it is safe to pass any values. b, e, and ilen are used as numbers of bytes (not characters), where b and e count from 0 to size() (end of buffer). If b and/or e don’t point to a valid UTF-8 character boundary, they are adjusted to the previous (b) or the next (e) valid UTF-8 character boundary, resp.. If the current number of characters in the buffer minus deleted characters plus inserted characters in text would overflow the number of allowed characters (maximum_size()), then only the first characters of the string are inserted, so that maximum_size() is not exceeded.
cut() and insert() are just inline functions that call replace().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
<th>beginning index of text to be deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>e</td>
<td>ending index of text to be deleted and insertion position</td>
</tr>
<tr>
<td>in</td>
<td>text</td>
<td>string that will be inserted</td>
</tr>
<tr>
<td>in</td>
<td>ilen</td>
<td>length of text or 0 for nul terminated strings</td>
</tr>
</tbody>
</table>

Returns

0 if nothing changed

Note

If text does not point to a valid UTF-8 character or includes invalid UTF-8 sequences, the text is inserted nevertheless (counting invalid UTF-8 bytes as one character each).

32.66.3.28  resize()

void Fl_Input_::resize {
  int X,
  int Y,
  int W,
  int H ) [virtual]

Changes the size of the widget.
This call updates the text layout so that the cursor is visible.

Parameters

| in  | X,Y,W,H | new size of the widget |

See also

Fl_Widget::resize(int, int, int, int)
Reimplemented from Fl_Widget.

32.66.3.29  shortcut() [1/2]

int Fl_Input_::shortcut ( ) const [inline]

Return the shortcut key associated with this widget.

Returns

shortcut keystroke
See also

`Fl_Button::shortcut()`

### 32.66.3.30 shortcut() [2/2]

```cpp
void Fl_Input_::shortcut ( 
    int s ) [inline]
```

Sets the shortcut key associated with this widget. Pressing the shortcut key gives text editing focus to this widget.

**Parameters**

- **in s** new shortcut keystroke

See also

`Fl_Button::shortcut()`

### 32.66.3.31 size() [1/2]

```cpp
int Fl_Input_::size ( ) const [inline]
```

Returns the number of bytes in `value()`. This may be greater than `strlen(value())` if there are null characters in the text.

**Returns**

- number of bytes in the text

### 32.66.3.32 size() [2/2]

```cpp
void Fl_Input_::size ( 
    int W, 
    int H ) [inline]
```

Sets the width and height of this widget.

**Parameters**

- **in W,H** new width and height

See also

`Fl_Widget::size(int, int)`

### 32.66.3.33 static_value() [1/2]

```cpp
int Fl_Input_::static_value ( 
    const char * str )
```

Changes the widget text. This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the `Fl_Input` is destroyed or `value()` is called again.
### static_value() [2/2]

```cpp
int Fl_Input_::static_value (const char * str, int len)
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.
You can use the len parameter to directly set the length if you know it already or want to put null characters in the text.

**Parameters**

- `str` (in): the new text
- `len` (in): the length of the new text

**Returns**

non-zero if the new value is different than the current one

### tab_nav() [1/2]

```cpp
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`

### tab_nav() [2/2]

```cpp
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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>If val is 1, Tab advances focus (default).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>If val is 0, Tab inserts a tab character (old FLTK behavior).</td>
</tr>
</tbody>
</table>

See also

`tab_nav()`, `Fl::OPTION_ARROW_FOCUS`.

32.66.3.37 **textcolor()** [1/2]

```cpp
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)`

32.66.3.38 **textcolor()** [2/2]

```cpp
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()`

32.66.3.39 **textfont()** [1/2]

```cpp
Fl_Font Fl_Input_::textfont ( ) const [inline]
```

Gets the font of the text in the input field.

Returns

the current Fl_Font index

32.66.3.40 **textfont()** [2/2]

```cpp
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 |

32.66.3.41  textsize() [1/2]

`Fl_Fontsize Fl_Input_::textsize () const [inline]`

Gets the size of the text in the input field.

Returns

the text height in pixels

32.66.3.42  textsize() [2/2]

`void Fl_Input_::textsize ( Fl_Fontsize s ) [inline]`

Sets the size of the text in the input field. The text height defaults to `FL_NORMAL_SIZE`.

Parameters

| in | s | the new font height in pixel units |

32.66.3.43  undo()

`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.

32.66.3.44  up_down_position()

`int Fl_Input_::up_down_position ( int i, int keepmark = 0 ) [protected]`

Moves the cursor to the column given by `up_down_pos`. This function is helpful when implementing up and down cursor movement. It moves the cursor from the beginning of a line to the column indicated by the global variable `up_down_pos` in pixel units.

Parameters

| in | i | index into the beginning of a line of text |
| in | keepmark | if set, move only the cursor, but not the mark |

Returns

index to new cursor position
32.66.3.45 value() [1/3]

const char* Fl_Input_::value ( ) const [inline]
Returns the text displayed in the widget.
This function returns the current value, which is a pointer to the internal buffer and is valid only until the next event
is handled.
Returns
  pointer to an internal buffer - do not free() this

See also
  Fl_Input_::value(const char*)

32.66.3.46 value() [2/3]

int Fl_Input_::value (  
    const char * str )
Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal
buffer. Passing NULL is the same as "".

Parameters
  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()

32.66.3.47 value() [3/3]

int Fl_Input_::value (  
    const char * str,  
    int len )
Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal
buffer. Passing NULL is the same as "".
You can use the length parameter to directly set the length if you know it already or want to put null characters
in the text.

Parameters
  in  str  the new text
  in  len  the length of the new text

Returns
  non-zero if the new value is different than the current one
See also

Fl_Input_::value(const char* str), Fl_Input_::value()

32.66.3.48  word_end()

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

\[ \text{in } i \quad \text{starting index for the search} \]

Returns

end of the word

32.66.3.49  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

\[ \text{in } i \quad \text{starting index for the search} \]

Returns

start of the word, or previous word

32.66.3.50  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.

32.66.3.51  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
32.67  Fl_Input_Choice Class Reference

A combination of the input widget and a menu button.

Inheritance diagram for Fl_Input_Choice:

```
+-----------------+      +-----------------+
| Fl_Widget       |      | Fl_Group        |
|                 +-----------------+      +-----------------+
|                 | Fl_Widget       |      | 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.

Protected Member Functions

- virtual int inp_h () const
  See inp_x() for info.
- virtual int inp_w () const
  See inp_x() for info.
- virtual int inp_x () const
  The methods inp_x(), inp_y(), inp_w() and inp_h() return the desired position and size of the internal Fl_Input widget.
- virtual int inp_y () const
  See inp_x() for info.
- virtual int menu_h () const
  See menu_x() for info.
- virtual int menu_w () const
  See menu_x() for info.
- virtual int menu_x () const
  The methods menu_x(), menu_y(), menu_w() and menu_h() return the desired position and size of the internal Fl_Menu_Button widget.
- virtual int menu_y () const
  See menu_x() for info.

Additional Inherited Members

32.67.1 Detailed Description

A combination of the input widget and a menu button.

![Figure 32.21 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 of Fl_Input_Choice

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

// Fl_Input_Choice callback()
void choice_cb(Fl_Widget *w, void *userdata) {
    // Show info about the picked item
    Fl_Input_Choice *choice = (Fl_Input_Choice*)w;
    printf("*** Choice Callback:\n" );
    printf(" widget's text value='"s'\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();
}
```

Subclassing Example

One can subclass Fl_Input_Choice to override the virtual methods inp_x/y/w/h() and menu_x/y/w/h() to take control of the internal Fl_Input and Fl_Menu_Button widget positioning. In this example, input and menubutton's positions are swapped:

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

class MyInputChoice : public Fl_Input_Choice {
protected:
    virtual int inp_x() const { return x() + Fl::box_dx(box()) + menu_w(); } // override to reposition
    virtual int menu_x() const { return x() + Fl::box_dx(box()); } // override to reposition
public:
    MyInputChoice(int X,int Y,int W,int H,const char*L=0) : Fl_Input_Choice(X,Y,W,H,L) {
        resize(X,Y,W,H); // necessary for ctor to trigger our overrides
    }
};

int main(int argc, char **argv) {
    Fl_Window *win = new Fl_Window(400,300);
    MyInputChoice mychoice = new MyInputChoice(150,40,150,25,"Right Align Input");
    mychoice->add("Aaa");
    mychoice->add("Bbb");
    mychoice->add("Ccc");
    win->end();
    win->resizable(win);
    win->show();
    return Fl::run();
}
```

32.67.2 Constructor & Destructor Documentation

32.67.2.1 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.
32.67.3 Member Function Documentation

32.67.3.1 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:

```c
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Fonts");
Fl_Menu_Button *mb = choice->menubutton(); // use Fl_Input_Choice’s Fl_Menu_Button
mb->add("Helvetica", 0, MyFont_CB, (void*)mydata); // use Fl_Menu_Button’s add() methods
mb->add("Courier", 0, MyFont_CB, (void*)mydata);
mb->add("More..", 0, FontDialog_CB, (void*)mydata);
```

32.67.3.2 inp_x()

virtual int Fl_Input_Choice::inp_x ( ) const [inline], [protected], [virtual]
The methods inp_x(), inp_y(), inp_w() and inp_h() return the desired position and size of the internal Fl_Input widget. These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

32.67.3.3 input()

Fl_Input* Fl_Input_Choice::input ( ) [inline]

Returns a pointer to the internal Fl_Input widget. This can be used to directly access all of the Fl_Input widget's methods.

32.67.3.4 menu_x()

virtual int Fl_Input_Choice::menu_x ( ) const [inline], [protected], [virtual]
The methods menu_x(), menu_y(), menu_w() and menu_h() return the desired position and size of the internal Fl_Menu_Button widget. These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

32.67.3.5 menubutton()

Fl_Menu_Button* Fl_Input_Choice::menubutton ( ) [inline]

Returns a pointer to the internal Fl_Menu_Button widget. This can be used to access any of the methods of the menu button, e.g.

```c
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)");
}
```

32.67.3.6 update_menubutton()

int Fl_Input_Choice::update_menubutton ( )

Updates the menubutton with the string value in Fl_Input. If the string value currently in Fl_Input matches one of the menu items in menubutton(), that menu item will become the current item selected. Call this method after setting value(const char*) if you need the menubutton() to be synchronized with the Fl_Input field.

```
// Add items
```
choice->add(".25");
choice->add(".50");
choice->add("1.0");
choice->add("2.0");
choice->add("4.0");
choice->value("1.0"); // sets Fl_Input to "1.0"
choice->update_menubutton(); // cause menubutton to reflect this value too
// (returns 1 if match was found, 0 if not)

// Verify menubutton's value.
printf("menu button choice index=%d, value=%s\n",
    choice->menubutton()->value(), // would be -1 if update not done
    choice->menubutton()->text()); // would be NULL if update not done

Returns
1 if a matching menuitem was found and value set, 0 if not.

Version
1.4.0

32.67.3.7 value() [1/2]

void Fl_Input_Choice::value (const char * val) [inline]

Sets the Fl_Input text field's contents to val.
Note it is possible to set the value() to one that is not in the menubutton's list of choices.
Setting the value() does NOT affect the menubutton's selection. If that's needed, call update_menubutton() after setting value().

See also
void value(int val), update_menubutton()

32.67.3.8 value() [2/2]

void Fl_Input_Choice::value (int val)

Chooses item# val in the menu, and sets the Fl_Input text field to that value.
Any previous text is cleared.

See also
void value(const char *val)

The documentation for this class was generated from the following files:

- Fl_Input_Choice.H
- Fl_Input_Choice.cxx

32.68 Fl_Int_Input Class Reference

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xabcdef).
#include <Fl_Int_Input.H>

Inheritance diagram for Fl_Int_Input:
**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**

**32.68.1 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 0xaff).

**32.68.2 Constructor & Destructor Documentation**

**32.68.2.1 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`

**32.69 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.

```c
#include <Fl_JPEG_Image.H>
```

Inheritance diagram for `Fl_JPEG_Image`:

```
Fl_Image
    Fl_RGB_Image
        Fl_JPEG_Image
```

Generated by Doxygen
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

32.69.1 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.

32.69.2 Constructor & Destructor Documentation

32.69.2.1 **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 *imagename, const unsigned char *data)**

32.69.2.2 **Fl_JPEG_Image()** [2/2]

Fl_JPEG_Image::Fl_JPEG_Image (const char *name, const unsigned char *data)

The constructor loads the JPEG image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. name_png can be NULL. If a name is given, the image is added to the list of shared images (see: Fl_Shared_Image) and will be available by that name.

The inherited destructor frees all memory and server resources that are used by the image.

Use Fl_Image::fail() to check if Fl_JPEG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.
32.70 Fl_Kdialog_Native_File_Chooser_Driver Class Reference

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>A unique name or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A pointer to the memory location of the JPEG image</td>
</tr>
</tbody>
</table>

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

32.70 Fl_Kdialog_Native_File_Chooser_Driver Class Reference

Inheritance diagram for Fl_Kdialog_Native_File_Chooser_Driver:

![Inheritance Diagram](image)

Friends

- class Fl_Native_File_Chooser

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser_Kdialog.H
- Fl_Native_File_Chooser_Kdialog.cxx

32.71 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

Generated by Doxygen
32.71.1 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.

32.71.2 Member Function Documentation

32.71.2.1 draw()

void Fl_Label::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    Fl_Align align ) const

Draws the label aligned to the given box. Draws a label with arbitrary alignment in an arbitrary box.

32.71.2.2 measure()

void Fl_Label::measure (  
    int & W,  
    int & H ) const

Measures the size of the label.

Parameters

\[ \text{in, out} \ W,H \] : this is the requested size for the label text plus image; on return, this will contain the size needed to fit the label

32.71.3 Member Data Documentation

32.71.3.1 type

uchar Fl_Label::type

type of label.
See also

Fl_Labeltype

The documentation for this struct was generated from the following files:

- Fl_Widget.H
- fl_labeltype.cxx

### 32.72 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:

```plaintext
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

### 32.72.1 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()`.

#### 32.72.2 Constructor & Destructor Documentation

Figure 32.22 Fl_Light_Button

Generated by Doxygen
### 32.72.2.1 Fl_Light_Button()

`Fl_Light_Button::Fl_Light_Button (int X, int Y, int W, int H, const char ∗l = 0)`

Creates a new `Fl_Light_Button` widget using the given position, size, and label string. The default box type is `FL_UP_BOX` and the default down box type `down_box()` is `FL_NO_BOX (0)`. The `selection_color()` sets the color of the "light". Default is `FL_YELLOW`. The default label alignment is `'FL_ALIGN_LEFT|FL_ALIGN_INSIDE'` so the label is drawn inside the button area right of the "light".

**Note**

Do not change the default box types of `Fl_Light_Button`. The box types determine how the button is drawn. If you change the `down_box()` type the drawing behavior is undefined.

#### 32.72.3 Member Function Documentation

##### 32.72.3.1 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`.

##### 32.72.3.2 handle()

```cpp
int Fl_Light_Button::handle (int event ) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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_Light_Button.H
- Fl_Light_Button.cxx

### 32.73 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

### 32.74 Fl_Mac_App_Menu Class Reference

**Static Public Member Functions**

- static void custom_application_menu_items (const Fl_Menu_Item *m)

*Adds custom menu item(s) to the application menu of the system menu bar.*

**Static Public Attributes**

- static const char *about
  *Localizable text for the “About xxx” application menu item.*
- static const char *hide
  *Localizable text for the “Hide xxx” application menu item.*
- static const char *hide_others
  *Localizable text for the “Hide Others” application menu item.*
- static const char *print
  *Localizable text for the “Print Front Window” application menu item.*
- static const char *print_no_titlebar
  *Localizable text for the “Print Front Window” application menu item.*
- static const char *quit

Generated by Doxygen
Localizable text for the "Quit xxx" application menu item.

- static const char * services
  Localizable text for the "Services" application menu item.
- static const char * show
  Localizable text for the "Show All" application menu item.
- static const char * toggle_print_titlebar
  Localizable text for the "Toggle print titlebar" application menu item.

32.74.1 Member Function Documentation

32.74.1.1 custom_application_menu_items()

static void Fl_Mac_App_Menu::custom_application_menu_items (const Fl_Menu_Item * m) [static]

Adds custom menu item(s) to the application menu of the system menu bar. They are positioned after the "Print Front Window / Toggle printing of titlebar" items, or at their place if they were removed with Fl_Mac_App_Menu::print = "".

Parameters

m zero-ending array of Fl_Menu_Item's.

32.74.2 Member Data Documentation

32.74.2.1 print

const char* Fl_Mac_App_Menu::print [static]

Localizable text for the "Print Front Window" application menu item. This menu item and next one won't be displayed if Fl_Mac_App_Menu::print is set to an empty string.

The documentation for this class was generated from the following file:

- mac.H

32.75 Fl_Menu_ Class Reference

Base class of all widgets that have a menu in FLTK.
#include <Fl_Menu_.H>

Inheritance diagram for Fl_Menu_:

```
Fl_Menu_  
|   |   |   |
Fl_Widget Fl_Menu_ Fl_Choice Fl_Menu_Bar Fl_Menu_Button  
|   |   |   |
Fl_Sys_Menu_Bar
```
Public Member Functions

- int add (const char *)
  
  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.

- int add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)
  
  Adds a new menu item.

- int add (const char *, a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  
  See int Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

- void clear ()
  
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

  int clear_submenu (int index)
  
  Clears the specified submenu pointed to by index of all menu items.

- void copy (const Fl_Menu_Item *, void *user_data=0)
  
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- Fl_Boxtype down_box () const
  
  This box type is used to surround the currently-selected items in the menus.

  void down_box (Fl_Boxtype b)
  
  See Fl_Boxtype Fl_Menu_::down_box() const

- Fl_Color down_color () const
  
  For back compatibility, same as selection_color()

  void down_color (unsigned c)
  
  For back compatibility, same as selection_color()

- int find_index (const char *) const
  
  Find the menu item index for a given menu path name, such as "Edit/Copy".

- int find_index (const Fl_Menu_Item *) const
  
  Find the index into the menu array for a given item.

- int find_index (Fl_Callback *) const
  
  Find the index into the menu array for a given callback cb.

- const Fl_Menu_Item * find_item (const char *)
  
  Find the menu item for a given menu path name, such as "Edit/Copy".

- const Fl_Menu_Item * find_item (Fl_Callback *)
  
  Find the menu item for the given callback cb.

- Fl_Menu_ (int, int, int, int, const char *=0)
  
  Creates a new Fl_Menu_ widget using the given position, size, and label string.

- void global ()
  
  Make the shortcuts for this menu work no matter what window has the focus when you type it.

- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)
  
  Inserts a new menu item at the specified index position.

- int insert (int index, const char *, a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  
  See int Fl_Menu_::insert(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

- int item_pathname (char *, name, int namelen, const Fl_Menu_Item *finditem=0) const
  
  Get the menu 'pathname' for the specified menuitem.

- const Fl_Menu_Item * menu () const
  
  Returns a pointer to the array of Fl_Menu_Items.

- void menu (const Fl_Menu_Item *)
  
  Sets the menu array pointer directly.

- const Fl_Menu_Item * menu_end ()
  
  Finishes menu modifications and returns menu().

- int mode (int i) const
  
  Gets the flags of item i.
• void **mode** (int i, int fl)
  
  *Sets the flags of item i.*

• const Fl_Menu_Item * **mvalue** () const

  *Returns a pointer to the last menu item that was picked.*

• const Fl_Menu_Item * **picked** (const Fl_Menu_Item *)

  *When user picks a menu item, call this.*

• void **remove** (int)

  *Deletes item i from the menu.*

• void **replace** (int, const char *)

  *Changes the text of item i.*

• void **setonly** (Fl_Menu_Item *item)

  *Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.*

• void **shortcut** (int i, int s)

  *Changes the shortcut of item i to s.*

• int **size** () const

  *This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.*

• void **size** (int W, int H)

• const Fl_Menu_Item * **test_shortcut** ()

  *Returns the menu item with the entered shortcut (key value).*

• const char * **text** () const

  *Returns the title of the last item chosen.*

• const char * **text** (int i) const

  *Returns the title of item i.*

• Fl_Color **textcolor** () const

  *Get the current color of menu item labels.*

• void **textcolor** (Fl_Color c)

  *Sets the current color of menu item labels.*

• Fl_Font **textfont** () const

  *Gets the current font of menu item labels.*

• void **textfont** (Fl_Font c)

  *Sets the current font of menu item labels.*

• Fl_Fontsize **textsize** () const

  *Gets the font size of menu item labels.*

• void **textsize** (Fl_Fontsize c)

  *Sets the font size of menu item labels.*

• int **value** () const

  *Returns the index into menu() of the last item chosen by the user.*

• int **value** (const Fl_Menu_Item *)

  *The value is the index into menu() of the last item chosen by the user.*

• int **value** (int i)

  *The value is the index into menu() of the last item chosen by the user.*

**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

32.75.1 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)

32.75.2 Constructor & Destructor Documentation

32.75.2.1 Fl_Menu_( )

Fl_Menu_::Fl_Menu_ ( int X, int Y, int W, int H, const char * l = 0 )

Creates a new Fl_Menu_ widget using the given position, size, and label string. menu() is initialized to null.

32.75.3 Member Function Documentation

32.75.3.1 add() [1/2]

int Fl_Menu_::add ( const char * str )

This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string. The passed string is split at any '|' characters and then add(s,0,0,0,0) is done with each section. This is often useful if you are just using the value, and is compatible with Forms and other GL programs. The section strings use the same special characters as described for the long version of add().

No items must be added to a menu during a callback to the same menu.

Parameters

| str | string containing multiple menu labels as described above |

Returns

the index into the menu() array, where the entry was added
32.75.3.2 add() [2/2]

```c
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.

Due to backwards compatibility and historical restrictions we recommend to use either

- static menu arrays that are not extended during runtime or
- dynamic, extendable menu item arrays that are entirely created by using add() or insert().

This ensures that all menu arrays and strings are copied to internal storage and released when required.

**Note**

- If you create menus from static Fl_Menu_Item arrays and add() or insert() more menu items later, then the menu array is copied to local storage but some local (static) strings may appear to "leak memory". This is a known issue and discouraged usage (see description above) but the impact on memory usage should typically be small.

**Detailed Description of Parameters**

**label**

The menu item's label. This argument is required and must not be NULL.

The characters "\&, \", \^", and "_" are treated as special characters in the label string. The "\&" character specifies that the following character is an accelerator and will be underlined. The "\" character is used to escape the next character in the string. Labels starting with the "_" character cause a divider to be placed after that menu item.
A label of the form "File/Quit" will create the submenu "File" with a menu item called "Quit".

The label string is copied to new memory and can be freed. The other arguments (including the shortcut) are copied into the menu item unchanged.

If an item exists already with that name then it is replaced with this new one. Otherwise this new one is added to the end of the correct menu or submenu. The return value is the offset into the array that the new entry was placed at.

**shortcut**

The keyboard shortcut for this menu item.

This parameter is optional, and defaults to 0 to indicate no shortcut.

The shortcut can either be a raw integer value (eg. FL_CTRL+'A') or a string (eg. "^c" or "^97").

Raw integer shortcuts can be a combination of keyboard chars (eg. 'A') and optional keyboard modifiers (see Fl::event_state(), e.g. FL_SHIFT, etc). In addition, FL_COMMAND can be used to denote FL_META under Mac OS X and FL_CTRL under other platforms.

String shortcuts can be specified in one of two ways:

- `[#+^]<ascii_value>` e.g. "97", "^97", "±97", "#97"
- `[#+^]<ascii_char>` e.g. "a", "^a", "±a", "#a"

...where `<ascii_value>` is a decimal value representing an ASCII character (eg. 97 is the ascii code for 'a'), and the optional prefixes enhance the value that follows. Multiple prefixes must appear in the order below.

- `#` - Alt
- `+` - Shift
- `^` - Control

Internally, the text shortcuts are converted to integer values using fl_old_shortcut(const char ∗).

**callback**

The callback to invoke when this menu item is selected.

This parameter is optional, and defaults to 0 for no callback.

**userdata**

The callback's 'user data' that is passed to the callback.
This parameter is optional, and defaults to 0.

flags

These are bit flags to define what kind of menu item this is.

This parameter is optional, and defaults to 0 to define a 'regular' menu item.

These flags can be 'OR'ed together:

- **FL_MENU_INACTIVE** // Deactivate menu item (gray out)
- **FL_MENU_TOGGLE** // Item is a checkbox toggle (shows checkbox for on/off state)
- **FL_MENU_VALUE** // The on/off state for checkbox/radio buttons (if set, state is 'on')
- **FL_MENU_RADIO** // Item is a radio button (one checkbox of many can be on)
- **FL_MENU_INVISIBLE** // Item will not show up (shortcut will work)
- **FL_SUBMENU_POINTER** // Indicates user_data() is a pointer to another menu array
- **FL_SUBMENU** // This item is a submenu to other items
- **FL_MENU_DIVIDER** // Creates divider line below this item. Also ends a group of radio buttons.

**All other bits in 'flags' are reserved and must not be used.**

If FL_SUBMENU is set in an item's flags, then actually two items are added:

- the first item is the menu item (submenu title), as expected, and
- the second item is the submenu terminating item with the label and all other members set to 0.

If you add submenus with the 'path' technique, then the corresponding submenu terminators (maybe more than one) are added as well.

**Todo** Raw integer shortcut needs examples. Dependent on responses to [https://www.fltk.org/newsgroups.php?g=fltk.coredev+v:10086](https://www.fltk.org/newsgroups.php?g=fltk.coredev+v:10086) and results of STR#2344

### 32.75.3.3 clear()

```cpp
void Fl_Menu_::clear ()
```

Same as menu(NULL), set the array pointer to null, indicating a zero-length menu. Menus must not be cleared during a callback to the same menu.

### 32.75.3.4 clear_submenu()

```cpp
int Fl_Menu_::clear_submenu ( int index )
```

Clears the specified submenu pointed to by index of all menu items. This method is useful for clearing a submenu so that it can be re-populated with new items. Example: a "File/Recent Files/..." submenu that shows the last few files that have been opened. The specified index must point to a submenu. The submenu is cleared with remove(). If the menu array was directly set with menu(x), then copy() is done to make a private array.

**Warning**

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

**Example:**

```cpp
int index = menubar->find_index("File/Recent"); // get index of "File/Recent" submenu
if ( index != -1 ) menubar->clear_submenu(index); // clear the submenu
menubar->add("File/Recent/Aaa");
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)`

### 32.75.3.5 copy()

```c
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)`.

### 32.75.3.6 down_box()

```c
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.

### 32.75.3.7 find_index() [1/3]

```c
int Fl_Menu_::find_index (const char * pathname) const
```

Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
This method finds a menu item's index position for the given menu `pathname`, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER).
To get the menu item pointer for a pathname, use `find_item()`

Parameters

| in pathname | The path and name of the menu item to find |

Returns

The index of the matching item, or -1 if not found.

See also

`item_pathname()`

### 32.75.3.8 find_index() [2/3]

```c
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.

Generated by Doxygen
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()`

#### 32.75.3.9 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

| 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

`find_index(const char*)`

#### 32.75.3.10 find_item() [1/2]

```cpp
const Fl_Menu_Item * Fl_Menu_::find_item (  
    const char * pathname )
```

Find the menu item for a given menu pathname, such as "Edit/Copy".

This method finds a menu item in the menu array, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER).

To get the menu item's index, use find_index(const char*)

### Example:

```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);
}
```
```c
if ( ( item = (Fl_Menu_Item*)menubar->find_item("Edit/&Copy") ) != NULL ) {
  item->labelcolor(FL_GREEN);
}
```

### Parameters

| **pathname** | The path and name of the menu item |

### Returns

The item found, or NULL if not found

### See also

find_index(const char*), find_item(Fl_Callback*), item_pathname()

---

### 32.75.3.11 find_item() [2/2]

```c
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_index(const char*)

---

### 32.75.3.12 global()

```c
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!)

### 32.75.3.13 insert()

```c
int Fl_Menu_::insert ( int index,
                      const char * label,
                      int shortcut,
                      Fl_Callback * callback,
                      void * userdata = 0,
                      int flags = 0 )
```
Inserts a new menu item at the specified index position. If index is -1, the menu item is appended; same behavior as add().

To properly insert a menu item, label must be the name of the item (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>in index</td>
<td>The menu array's index position where the new item is inserted. If -1, behavior is the same as add().</td>
</tr>
<tr>
<td>in label</td>
<td>The text label for the menu item. If the label is a menu pathname, index is ignored, and the pathname indicates the position of the new item.</td>
</tr>
<tr>
<td>in shortcut</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>in callback</td>
<td>Optional callback invoked when user clicks the item. Default 0 if none.</td>
</tr>
<tr>
<td>in userdata</td>
<td>Optional user data passed as an argument to the callback. Default 0 if none.</td>
</tr>
<tr>
<td>in flags</td>
<td>Optional flags that control the type of menu item; see add() 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()

### 32.75.3.14 item_pathname()

int Fl_Menu_::item_pathname (char ∗name, int namelen, const Fl_Menu_Item ∗finditem = 0) const

Get the menu 'pathname' for the specified menuitem. If finditem==NULL, mvalue() is used (the most recently picked menuitem).

**Example:**

```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="")
32.75.3.15 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

find_item() - 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
}
```

32.75.3.16 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.

32.75.3.17 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()

32.75.3.18 mode() [1/2]

int Fl_Menu_::mode (int i) const [inline]

Gets the flags of item i.
For a list of the flags, see Fl_Menu_Item.

32.75.3.19 mode() [2/2]

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.

32.75.3.20 mvalue()

const Fl_Menu_Item * Fl_Menu_::mvalue () const [inline]

Returns a pointer to the last menu item that was picked.

32.75.3.21 picked()

const Fl_Menu_Item * Fl_Menu_::picked (const Fl_Menu_Item * v)

When user picks a menu item, call this.
It will do the callback. Unfortunately this also casts away const for the checkboxes, but this was necessary so
non-checkbox menus can really be declared const...

32.75.3.22 remove()

void Fl_Menu_::remove (int i)

Deletes item i from the menu.
If the menu array was directly set with menu(x) then copy() is done to make a private array.
No items must be removed from a menu during a callback to the same menu.

Parameters

i index into menu array
32.75.3.23 replace()

```cpp
type Fl_Menu_::replace (  
    int i,  
    const char * str )
Changes the text of item i.
This is the only way to get slash into an add()'ed menu item. If the menu array was directly set with menu(x) then
    copy() is done to make a private array.

Parameters

<table>
<thead>
<tr>
<th>i</th>
<th>index into menu array</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>new label for menu item at index i</td>
</tr>
</tbody>
</table>
```

32.75.3.24 size()

```cpp
int Fl_Menu_::size ( ) const
This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
This includes the "terminator" item at the end. To copy a menu array you need to copy size()*sizeof(Fl_Menu_Item)
bytes. If the menu is NULL this returns zero (an empty menu will return 1).
```

32.75.3.25 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.
```

32.75.3.26 text() [1/2]

```cpp
const char* Fl_Menu_::text ( ) const [inline]
Returns the title of the last item chosen.
```

32.75.3.27 text() [2/2]

```cpp
const char* Fl_Menu_::text (  
    int i ) const [inline]
Returns the title of item i.
```

32.75.3.28 textcolor()

```cpp
Fl_Color Fl_Menu_::textcolor ( ) const [inline]
Get the current color of menu item labels.
```
32.75.3.29 textfont() [1/2]

```
Fl_Font Fl_Menu_::textfont ( ) const [inline]
```

Gets the current font of menu item labels.

32.75.3.30 textfont() [2/2]

```
void Fl_Menu_::textfont ( Fl_Font c ) [inline]
```

Sets the current font of menu item labels.

32.75.3.31 textsize() [1/2]

```
Fl_Fontsize Fl_Menu_::textsize ( ) const [inline]
```

Gets the font size of menu item labels.

32.75.3.32 textsize() [2/2]

```
void Fl_Menu_::textsize ( Fl_Fontsize c ) [inline]
```

Sets the font size of menu item labels.

32.75.3.33 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.

32.75.3.34 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.

32.75.3.35 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

Generated by Doxygen
32.76 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

```c
#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

#### 32.76.1 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.

```
foo File Edit Checkbox Radio Font Empty Inactive Huge button
```

Figure 32.23 menubar

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 '&amp;' character in the name field of the menu item. If you put a '&amp;' character in a top-level "button" then the shortcut picks it. The '&amp;' 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.

### 32.76.2 Constructor & Destructor Documentation

#### 32.76.2.1 Fl_Menu_Bar()

Fl_Menu_Bar::Fl_Menu_Bar (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * l = 0 )

Creates a new Fl_Menu_Bar widget using the given position, size, and label string. The default boxtype is FL_UP_BOX.

The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu. labelsize(), 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.

### 32.76.3 Member Function Documentation

#### 32.76.3.1 draw()

void Fl_Menu_Bar::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```c
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.

#### 32.76.3.2 handle()

int Fl_Menu_Bar::handle (  
  int event ) [virtual]

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
Parameters

\[ \text{in } \text{event} \text{ the kind of event received} \]

Return values

\[
\begin{array}{c|c}
0 & \text{if the event was not used or understood} \\
1 & \text{if the event was used and can be deleted} \\
\end{array}
\]

See also

Fl_Event

Reimplemented from Fl_Widget.

32.76.3.3 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

32.77 Fl_Menu_Button Class Reference

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.

#include <Fl_Menu_Button.H>

Inheritance diagram for Fl_Menu_Button:

```
Fl_Widget
    ↓
Fl_Menu_
    ↓
Fl_Menu_Button
```

Public Types

- enum popup_buttons {
  POPUP1 = 1, POPUP2, POPUP12, POPUP3, POPUP13, POPUP23, POPUP123
  }

  indicate what mouse buttons pop up the menu.

Public Member Functions

- Fl_Menu_Button (int, int, int, int, const char **=0)
  
  Creates a new Fl_Menu_Button widget using the given position, size, and label string.

- int handle (int)
  
  Handles the specified event.

- const Fl_Menu_Item * popup ()
  
  Act exactly as though the user clicked the button or typed the shortcut key.
Protected Member Functions

• void draw ()

Draws the widget.

Additional Inherited Members

32.77.1 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.

![Figure 32.24 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.

32.77.2 Member Enumeration Documentation

32.77.2.1 popup_buttons

```cpp
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.
32.77 Fl_Menu_Button Class Reference

## Enumerator

<table>
<thead>
<tr>
<th>Name</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>

32.77.3 Constructor & Destructor Documentation

### Fl_Menu_Button()

```cpp
def 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.

32.77.4 Member Function Documentation

### draw()

```cpp
def void Fl_Menu_Button::draw ( ) [protected], [virtual]
```

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead. Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

### handle()

```cpp
def int Fl_Menu_Button::handle (int event) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>the kind of event received</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the event was not used or understood</td>
</tr>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.

32.77.4.3 popup()

const Fl_Menu_Item ∗ Fl_Menu_Button::popup ( )

Act exactly as though the user clicked the button or typed the shortcut key.
The menu appears, it waits for the user to pick an item, and if they pick one it sets value() and does the callback or
sets changed() as described above. The menu item is returned or NULL if the user dismisses the menu.

Note

Since FLTK 1.4.0 Fl_Menu::menu_end() is called before the menu pops up to make sure the menu array is
located in private storage.

See also

Fl_Menu::menu_end()

The documentation for this class was generated from the following files:

- Fl_Menu_Button.H
- Fl_Menu_Button.cxx

32.78 Fl_Menu_Item Struct Reference

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.
#include <Fl_Menu_Item.H>

Public Member Functions

- void activate ( )
  Allows a menu item to be picked.
- int active ( ) const
  Gets whether or not the item can be picked.
- int activevisible ( ) const
  Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.
- int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)
  Adds a menu item.
- int add (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)
  See int add(const char ∗, int shortcut, Fl_Callback *, void *, int)
- long argument ( ) const
  Gets the user_data() argument that is sent to the callback function.
- void argument (long v)
  Sets the user_data() argument that is sent to the callback function.
- Fl_Callback_p callback ( ) const
  Returns the callback function that is set for the menu item.
- void callback (Fl_Callback ∗c)
Sets the menu item's callback function.

- void callback (Fl_Callback *c, void *p)
  Sets the menu item's callback function and userdata() argument.

- void callback (Fl_Callback0 *c)
  Sets the menu item's callback function.

- void callback (Fl_Callback1 *c, long p=0)
  Sets the menu item's callback function and userdata() argument.

- void check ()
  Back compatibility only.

- int checkbox () const
  Returns true if a checkbox will be drawn next to this item.

- int checked () const
  Back compatibility only.

- void clear ()
  Turns the check or radio item "off" for the menu item.

- void deactivate ()
  Prevents a menu item from being picked.

- void do_callback (Fl_Widget *o) const
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.

- void do_callback (Fl_Widget *o, long arg) const
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.

- void do_callback (Fl_Widget *o, void *arg) const
  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.

- void draw (int x, int y, int w, int h, const Fl_Menu_*, int t=0) const
  Draws the menu item in bounding box x,y,w,h, optionally selects the item.

- const Fl_Menu_Item *find_shortcut (int *ip=0, const bool require_alt=false) const
  Search only the top level menu for a shortcut.

- Fl_Menu_Item *first ()
  Returns the first menu item, same as next(0).

- const Fl_Menu_Item *first () const
  Returns the first menu item, same as next(0).

- void hide ()
  Hides an item in the menu.

- void image (Fl_Image &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.

• **Fl_Menu_Item ∗ next (int i=1)**
  
  Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

• **const Fl_Menu_Item ∗ next (int=1) const**
  
  Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items.

• **const Fl_Menu_Item ∗ popup (int X, int Y , const char ∗title=0, const Fl_Menu_Item ∗picked=0, const Fl_Menu_ ∗=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 ∗ testShortcut () const**
  
  This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.

• **void uncheck ()**
  
  Back compatibility only.

• **void ∗ user_data () const**
  
  Gets the user_data() argument that is sent to the callback function.

• **void user_data (void ∗v)**
  
  Sets the user_data() argument that is sent to the callback function.

• **int value () const**
  
  Returns the current value of the check or radio item.

• **int visible () const**
  
  Gets the visibility of an item.
Public Attributes

- **Fl_Callback * callback_**
  menu item callback
- **int flags**
  menu item flags like FL_MENU_TOGGLE, FL_MENU_RADIO
- **Fl_Color labelcolor_**
  menu item text color
- **Fl_Font labelfont_**
  which font for this menu item text
- **Fl_Fontsize labelsize_**
  size of menu item text
- **uchar 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

32.78.1 Detailed Description

The **Fl_Menu_Item** structure defines a single menu item that is used by the **Fl_Menu_** class.

```c
struct Fl_Menu_Item {
  const char* text; // label()
  int shortcut;
  Fl_Callback* callback_;
  void* user_data;
  int flags;
  uchar labeltype;
  uchar labelfont;
  uchar labelsize;
  uchar labelcolor;
};
```

Typically menu items are statically defined; for example:

```c
Fl_Menu_Item popup[] = {
  {"alpha", FL_ALT+'a', the_cb, (void*)1},
  {"beta", FL_ALT+'b', the_cb, (void*)2},
  {"gamma", FL_ALT+'c', the_cb, (void*)3, FL_MENU_DIVIDER},
  {"strange", 0, strange_cb},
  {"charm", 0, charm_cb},
  {"truth", 0, truth_cb},
  {"beauty", 0, beauty_cb},
  {"submenu", 0, 0, 0, FL_SUBMENU},
  {"one"},
  {"two"},
  {"three"},
  {0},
  {"inactive", FL_ALT+'i', 0, 0, FL_MENU_INACTIVE|FL_MENU_DIVIDER},
  {"invisible",FL_ALT+'i', 0, 0, FL_MENU_INVISIBLE},
  {"check", FL_ALT+'c', 0, 0, FL_MENU_TOGGLE|FL_MENU_VALUE},
  {"box", FL_ALT+'b', 0, 0, FL_MENU_TOGGLE},
  {0}};
```

produces:
A submenu title is identified by the bit FL_SUBMENU in the flags field, and ends with a label() that is NULL. You can nest menus to any depth. A pointer to the first item in the submenu can be treated as an Fl_Menu array itself. It is also possible to make separate submenu arrays with FL_SUBMENU_POINTER flags.

You should use the method functions to access structure members and not access them directly to avoid compatibility problems with future releases of FLTK.

Note

Adding menu items with insert(), add(), or any of its overloaded variants copies the entire menu to internal storage. Using the memory of a static menu array after that would access unused (but not released) memory and thus have no effect.

### 32.78.2 Member Function Documentation

#### 32.78.2.1 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)

32.78.2.2 argument() [1/2]

long Fl_Menu_Item::argument ( ) const [inline]
Gets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the stored user-
data() argument to long and returns it as a long value.

32.78.2.3 argument() [2/2]

void Fl_Menu_Item::argument (long v) [inline]
Sets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the given argument 
v to void* and stores it in the menu item's userdata() member. This may not be portable to some machines.

32.78.2.4 callback() [1/5]

Fl_Callback_p Fl_Menu_Item::callback ( ) const [inline]
Returns the callback function that is set for the menu item.
Each item has space for a callback function and an argument for that function. Due to back compatibility, the Fl_Menu_Item itself is not passed to the callback, instead you have to get it by calling ((Fl_Menu_*+)w)->mvalue() where w is the widget argument.

32.78.2.5 callback() [2/5]

void Fl_Menu_Item::callback ( Fl_Callback * c ) [inline]
Sets the menu item's callback function.
This method does not set the userdata() argument.
See also
Fl_Callback_p Fl_MenuItem::callback() const

32.78.2.6 callback() [3/5]

void Fl_Menu_Item::callback ( Fl_Callback * c,
void * p ) [inline]
Sets the menu item's callback function and userdata() argument.
See also
Fl_Callback_p Fl_MenuItem::callback() const
32.78.2.7 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

```cpp
Fl_Callback_p Fl_MenuItem::callback() const
```

32.78.2.8 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. The argument is cast to void* and stored as the userdata() for the menu item's callback function.

See also

```cpp
Fl_Callback_p Fl_MenuItem::callback() const
```

32.78.2.9 check()

```cpp
void Fl_Menu_Item::check () [inline]
```

Back compatibility only.

**Deprecated** Please use `Fl_Menu_Item::set()` instead. This method will be removed in FLTK 1.5.0 or later.

See also

```cpp
set()
```

32.78.2.10 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.

32.78.2.11 checked()

```cpp
int Fl_Menu_Item::checked () const [inline]
```

Back compatibility only.

**Deprecated** Please use `Fl_Menu_Item::value()` instead. This method will be removed in FLTK 1.5.0 or later.

See also

```cpp
value()
```

32.78.2.12 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.
32.78.2.13 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 `userdata()` as its second argument. You must first check that `callback()` is non-zero before calling this.

32.78.2.14 do_callback() [2/3]

```cpp
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`. The `long arg` cast to void* when calling the callback. You must first check that `callback()` is non-zero before calling this.

32.78.2.15 do_callback() [3/3]

```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.

32.78.2.16 findShortcut()

```cpp
const Fl_Menu_Item * Fl_Menu_Item::findShortcut ( int * ip = 0, const bool require_alt = false ) const
```

Search only the top level menu for a shortcut. Either &x in the label or the shortcut fields are used. This tests the current event, which must be an `FL_KEYBOARD` or `FL_SHORTCUT`, against a shortcut value.

Parameters

| ip | returns the index of the item, if ip is not NULL. |
| require_alt | if true: match only if Alt key is pressed. |

Returns

found `Fl_Menu_Item` or NULL

32.78.2.17 image() [1/2]

```cpp
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.

32.78.2.18 image() [2/2]

```cpp
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.

### 32.78.2.19 insert()

```cpp
int Fl_Menu_Item::insert(
    int index,
    const char * mytext,
    int sc,
    Fl_Callback * cb,
    void * data = 0,
    int myflags = 0)
```

Inserts an item at position `index`. If `index` is -1, the item is added the same way as `Fl_Menu_Item::add()`.

If 'mytext' contains any un-escaped front slashes (/), it's assumed a menu pathname is being specified, and the value of `index` will be ignored.

In all other aspects, the behavior of `insert()` is the same as `add()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>insert new items here</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mytext</td>
<td>new label string, details see above</td>
</tr>
<tr>
<td>in</td>
<td>sc</td>
<td>keyboard shortcut for new item</td>
</tr>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function for new item</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>user data for new item</td>
</tr>
<tr>
<td>in</td>
<td>myflags</td>
<td>menu flags as described in <code>Fl_Menu_Item</code></td>
</tr>
</tbody>
</table>

**Returns**

the index into the menu() array, where the entry was added

### 32.78.2.20 label()

```cpp
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.

### 32.78.2.21 labelcolor() [1/2]

```cpp
Fl_Color Fl_Menu_Item::labelcolor ( ) const [inline]
```

Gets the menu item's label color.

This color is passed to the labeltype routine, and is typically the color of the label text. This defaults to FL_BLACK.

If this color is not black fltk will not use overlay bitplanes to draw the menu - this is so that images put in the menu draw correctly.

### 32.78.2.22 labelcolor() [2/2]

```cpp
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

32.78.2.23 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.

32.78.2.24 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.

32.78.2.25 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.

32.78.2.26 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.

32.78.2.27 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.

32.78.2.28 next() [1/2]

Fl_Menu_Item* Fl_Menu_Item::next (int i = 1) [inline]

Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.
There are two calls so that you can advance through const and non-const data.

32.78.2.29 next() [2/2]

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.

Generated by Doxygen
32.78.2.30 popup()

const Fl_Menu_Item * Fl_Menu_Item::popup (  
    int X,  
    int Y,  
    const char * title = 0,  
    const Fl_Menu_Item * picked = 0,  
    const Fl_Menu_Item * menu_button = 0 ) const

This method is called by widgets that want to display menus. The menu stays up until the user picks an item or dismisses it. The selected item (or NULL if none) is returned. This does not do the callbacks or change the state of check or radio items. The menu is positioned so the cursor is centered over the item picked. This will work even if picked is in a submenu. If picked is zero or not in the menu item table the menu is positioned with the cursor in the top-left corner.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>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).</td>
</tr>
<tr>
<td>title</td>
<td>a character string title for the menu. If non-zero a small box appears above the menu with the title in it.</td>
</tr>
<tr>
<td>picked</td>
<td>if this pointer is not NULL, the popup menu will appear so that the picked menu is under the mouse pointer.</td>
</tr>
<tr>
<td>menu_button</td>
<td>is a pointer to an Fl_Menu_ from which the color and boxtypes for the menu are pulled. If NULL then defaults are used.</td>
</tr>
</tbody>
</table>

Returns

a pointer to the menu item selected by the user, or NULL

32.78.2.31 pulldown()

const Fl_Menu_Item * Fl_Menu_Item::pulldown (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const Fl_Menu_Item * initial_item = 0,  
    const Fl_Menu_Item * pbutton = 0,  
    const Fl_Menu_Item * title = 0,  
    int menubar = 0 ) const

Pulldown() is similar to popup(), but a rectangle is provided to position the menu. The menu is made at least W wide, and the picked item initial_item is centered over the rectangle (like Fl_Choice uses). If initial_item is NULL or not found, the menu is aligned just below the rectangle (like a pulldown menu). The title and menubar arguments are used internally by the Fl_Menu_Bar widget.

32.78.2.32 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.

32.78.2.33 set()

void Fl_Menu_Item::set () [inline]
Turns the check or radio item "on" for the menu item.
Note that this does not turn off any adjacent radio items like setonly() does.

32.78.2.34 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_::setonly(Fl_Menu_Item*) instead.

32.78.2.35 shortcut()

```cpp
void Fl_Menu_Item::shortcut ( int s ) [inline]
```

Sets exactly what key combination will trigger the menu item.
The value is a logical 'or' of a key and a set of shift flags, for instance FL_ALT+’a’ or FL_ALT+FL_F+10 or just ’a’. A value of zero disables the shortcut.
The key can be any value returned by Fl::event_key(), but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by Fl::event_state(). If the bit is on that shift key must be pushed.
Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a “don’t care” setting).

32.78.2.36 size()

```cpp
int Fl_Menu_Item::size ( ) const
```

Size of the menu starting from this menu item.
This method counts all menu items starting with this menu item, including all menu items in the same (sub)menu level, all nested submenus, and the terminating empty (0) menu item.
It does not count menu items referred to by FL_SUBMENU_POINTER menu items (except the single menu item with FL_SUBMENU_POINTER).
All menu items counted are consecutive in memory (one array).

**Example:**
```cpp
schemechoice = new Fl_Choice(X+125,Y,140,25,"FLTK Scheme");
schemechoice->add("none");
schemechoice->add("plastic");
schemechoice->add("gtk+");
schemechoice->add("gleam");
printf("schemechoice->menu()->size() = %d\n", schemechoice->menu()->size());
```

**Output:**
schemechoice->menu()->size() = 5

32.78.2.37 submenu()

```cpp
int Fl_Menu_Item::submenu ( ) const [inline]
```

Returns true if either FL_SUBMENU or FL_SUBMENU_POINTER is on in the flags.
FL_SUBMENU indicates an embedded submenu that goes from the next item through the next one with a NULL label(). FL_SUBMENU_POINTER indicates that user_data() is a pointer to another menu array.

32.78.2.38 test_shortcut()

```cpp
const Fl_Menu_Item * Fl_Menu_Item::test_shortcut ( ) const
```

This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.
If the current event matches one of the items shortcut, that item is returned. If the keystroke does not match any shortcuts then NULL is returned. This only matches the shortcut() fields, not the letters in the title preceeded by ’
uncheck()

void Fl_Menu_Item::uncheck() [inline]

Back compatibility only.

Deprecated Please use Fl_Menu_Item::clear() instead. This method will be removed in FLTK 1.5.0 or later.

See also

clear()

value()

int Fl_Menu_Item::value() const [inline]

Returns the current value of the check or radio item.
This is zero (0) if the menu item is not checked and non-zero otherwise.

Since

1.4.0 this method returns 1 if the item is checked but you should not rely on a particular value, only zero or non-zero.

Note

The returned value for a checked menu item was FL_MENU_VALUE (4) before FLTK 1.4.0.

The documentation for this struct was generated from the following files:

- Fl_Menu_Item.H
- Fl_Menu.cxx
- Fl_Menu_.cxx
- Fl_Menu_add.cxx

32.79 Fl_Menu_Window Class Reference

The Fl_Menu_Window widget is a window type used for menus.

#include <Fl_Menu_Window.H>

Inheritance diagram for Fl_Menu_Window:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Window
    ↓
Fl_Single_Window
    ↓
Fl_Menu_Window
```

Public Member Functions

- Fl_Menu_Window (int W, int H, const char *l=0)
  
  Creates a new Fl_Menu_Window widget using the given size, and label string.

- Fl_Menu_Window (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Menu_Window widget using the given position, size, and label string.

- ~Fl_Menu_Window ()
  
  Destroys the window and all of its children.
Additional Inherited Members

32.79.1 Detailed Description

The Fl_Menu_Window widget is a window type used for menus. By default the window is drawn in the hardware overlay planes if they are available so that the menu don't force the rest of the window to redraw. The documentation for this class was generated from the following files:

- Fl_Menu_Window.H
- Fl_Menu_Window.cxx

32.80 Fl_Multi_Browser Class Reference

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

#include <Fl_Multi_Browser.H>

Inheritance diagram for Fl_Multi_Browser:

```
Fl_Widget
  |
  V
Fl_Group
  |
  V
Fl_Browser
  |
  V
Fl_Browser_
    |
    V
Fl_Multi_Browser
```

Public Member Functions

- Fl_Multi_Browser (int X, int Y, int W, int H, const char *L=0)
  Creates a new Fl_Multi_Browser widget using the given position, size, and label string.

Additional Inherited Members

32.80.1 Detailed Description

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

The user interface is Macintosh style: clicking an item turns off all the others and selects that one, dragging selects all the items the mouse moves over, and ctrl + click (Cmd+click on the Mac OS platform) toggles the items. Shift + click extends the selection until the clicked item. This is different from how forms did it. Normally the callback is done when the user releases the mouse, but you can change this with when(). See Fl_Browser for methods to add and remove lines from the browser.

32.80.2 Constructor & Destructor Documentation
32.80.2.1 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_ART_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

32.81 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_Menu_Item ∗)
  
  This method is used to associate a Fl_Multi_Label with a Fl_Menu_Item.
- void label (Fl_Widget ∗)
  
  This method is used to associate a Fl_Multi_Label with a Fl_Widget.

Public Attributes

- const char ∗labela
  
  Holds the "leftmost" of the two elements in the composite label.
- const char ∗labelb
  
  Holds the "rightmost" of the two elements in the composite label.
- uchar typea
  
  Holds the "type" of labela.
- uchar typeb
  
  Holds the "type" of labelb.

32.81.1 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 32.27 Menu items with icons using Fl_Multi_Label
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 multi label to the menu item
ml->label(item);
```

See also
Fl_Label and Fl_Labeltype and examples/howto-menu-with-images.cxx

### 32.81.2 Member Data Documentation

#### 32.81.2.1 labela

const char* Fl_Multi_Label::labela

Holds the "leftmost" of the two elements in the composite label.

Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

#### 32.81.2.2 labelb

const char* Fl_Multi_Label::labelb

Holds the "rightmost" of the two elements in the composite label.

Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

#### 32.81.2.3 typea

uchar Fl_Multi_Label::typea

Holds the "type" of labela.

Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.

#### 32.81.2.4 typeb

uchar Fl_Multi_Label::typeb

Holds the "type" of labelb.

Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.

The documentation for this struct was generated from the following files:

- Fl_Multi_Label.H
- Fl_Multi_Label.cxx

### 32.82 Fl_Multiline_Input Class Reference

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

Generated by Doxygen
#include <Fl_Multiline_Input.H>

Inheritance diagram for Fl_Multiline_Input:

```
Fl_Widget
   |        |
   |        |
Fl_Input_  
   |   |
Fl_Input
   |   |
Fl_Multiline_Input
```

Public Member Functions

- **Fl_Multiline_Input** (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Multiline_Input widget using the given position, size, and label string.

Additional Inherited Members

### 32.82.1 Detailed Description

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

This is for editing multiline text.

This is far from the nirvana of text editors, and is probably only good for small bits of text, 10 lines at most. Note that this widget does not support scrollbars or per-character color control.

If you are presenting large amounts of text and need scrollbars or full color control of characters, you probably want Fl_Text_Editor instead.

In FLTK 1.3.x, the default behavior of the 'Tab' key was changed to support consistent focus navigation. To get the older FLTK 1.1.x behavior, set Fl_Input_::tab_nav() to 0. Newer programs should consider using Fl_Text_Editor.

### 32.82.2 Constructor & Destructor Documentation

#### 32.82.2.1 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

### 32.83 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_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**

### 32.83.1 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.

### 32.83.2 Constructor & Destructor Documentation

#### 32.83.2.1 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

### 32.84 Fl_Native_FILE_CHOOSER Class Reference

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

```c
#include <Fl_Native_FILE_CHOOSER.H>
```
Public Types

- `enum Option { NO_OPTIONS = 0x0000, SAVEAS_CONFIRM = 0x0001, NEW_FOLDER = 0x0002, PREVIEW = 0x0004, USE_FILTER_EXT = 0x0008 }

- `enum Type { BROWSE_FILE = 0, BROWSE_DIRECTORY, BROWSE_MULTI_FILE, BROWSE_MULTI_DIRECTORY, BROWSE_SAVE_FILE, BROWSE_SAVE_DIRECTORY }

Public Member Functions

- `int count () const
  Returns the number of filenames (or directory names) the user selected.

- `const char ∗ directory () const
  Returns the current preset directory() value.

- `void directory (const char ∗val)
  Preset the directory the browser will show when opened.

- `const char ∗ errmsg () const
  Returns a system dependent error message for the last method that failed.

- `const char ∗ filename () const
  Return the filename the user chose.

- `const char ∗ filename (int i) const
  Return one of the filenames the user selected.

- `const char ∗ filter () const
  Returns the filter string last set.

- `void filter (const char ∗f)
  Sets the filename filters used for browsing.

- `int filter_value () const
  Returns which filter value was last selected by the user.

- `void filter_value (int i)
  Sets which filter will be initially selected.

- `int filters () const
  Gets how many filters were available, not including "All Files".

- `Fl_Native_File_Chooser (int val=BROWSE_FILE)
  The constructor.

- `int options () const
  Gets the platform specific Fl_Native_File_Chooser::Option flags.

- `void options (int o)
  Sets the platform specific chooser options to val.

- `const char ∗ preset_file () const
  Get the preset filename.

- `void preset_file (const char ∗f)
  Sets the default filename for the chooser.

- `int show ()
  Post the chooser's dialog.

- `const char ∗ title () const
  Get the title of the file chooser's dialog window.

- `void title (const char ∗t)
  Set the title of the file chooser's dialog window.

- `int type () const
  Gets the current Fl_Native_File_Chooser::Type of browser.

- `void type (int t)
  Sets the current Fl_Native_File_Chooser::Type of browser.

- `~Fl_Native_File_Chooser ()
  Destructor.
Static Public Attributes

- static const char * file_exists_message = "File exists. Are you sure you want to overwrite?"
  
  Localizable message.

32.84.1 Detailed Description

This class lets an FLTK application easily and consistently access the operating system's native file chooser. Some operating systems have very complex and specific file choosers that many users want access to specifically, instead of FLTK's default file chooser(s). In cases where there is no native file browser, FLTK's own file browser is used instead.

To use this widget, use the following include in your code:

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

The following example shows how to pick a single file:

```
// Create and post the local native file chooser
#include <FL/Fl_Native_File_Chooser.H>
[..]
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE_FILE);
fnfc.filter("Text\t*.txt\n" "C Files\t*.c,.cxx,h,.c");
fnfc.directory("/var/tmp");  // default directory to use

// Show native chooser
switch ( fnfc.show() ) {
  case -1: printf("ERROR: %s\n", fnfc.errmsg()); break; // ERROR
  case 1: printf("CANCEL\n"); break; // CANCEL
  default: printf("PICKED: %s\n", fnfc.filename()); break; // FILE CHOSEN
}
```

The Fl_Native_File_Chooser widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the fl_fopen() or fl_open() utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).

Platform Specific Caveats

- Under 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.
32.84.2 Member Enumeration Documentation

32.84.2.1 Option

enum Fl_Native_File_Chooser::Option

 Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OPTIONS</td>
<td>no options enabled</td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Show native ‘Save As’ overwrite confirm dialog.</td>
</tr>
<tr>
<td>NEW_FOLDER</td>
<td>Show ‘New Folder’ icon (if supported)</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>enable preview mode (if supported)</td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter pilots the output file extension (if supported)</td>
</tr>
</tbody>
</table>

32.84.2.2 Type

enum Fl_Native_File_Chooser::Type

 Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWSE_FILE</td>
<td>browse files (lets user choose one file)</td>
</tr>
<tr>
<td>BROWSE_DIRECTORY</td>
<td>browse directories (lets user choose one directory)</td>
</tr>
<tr>
<td>BROWSE_MULTI_FILE</td>
<td>browse files (lets user choose multiple files)</td>
</tr>
<tr>
<td>BROWSE_MULTI_DIRECTORY</td>
<td>browse directories (lets user choose multiple directories)</td>
</tr>
</tbody>
</table>
### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWSE_SAVE_FILE</td>
<td>browse to save a file</td>
</tr>
<tr>
<td>BROWSE_SAVE_DIRECTORY</td>
<td>browse to save a directory</td>
</tr>
</tbody>
</table>

### 32.84.3 Constructor & Destructor Documentation

#### 32.84.3.1 Fl_Native_File_Chooser()

```
Fl_Native_File_Chooser::Fl_Native_File_Chooser ( int val = BROWSE_FILE )
```

The constructor.
Internally allocates the native widgets. Optional `val` presets the type of browser this will be, which can also be changed with `type()`.

#### 32.84.3.2 ∼Fl_Native_File_Chooser()

```
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ( )
```

Destructor.
Deallocates any resources allocated to this widget.

### 32.84.4 Member Function Documentation

#### 32.84.4.1 count()

```
int Fl_Native_File_Chooser::count ( ) const
```

Returns the number of filenames (or directory names) the user selected.

**Example:**
```
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));
  }
}
```

#### 32.84.4.2 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.

#### 32.84.4.3 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.

#### 32.84.4.4 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).
32.84.4.5 filename() [2/2]

const char * Fl_Native_File_Chooser::filename ( int i ) const

Return one of the filenames the user selected.
Use count() to determine how many filenames the user selected.

Example:
if ( fnfc->show() == 0 ) {
  // Print all filenames user selected
  for (int n=0; n<fnfc->count(); n++ ) {
    printf("%d) '%s'
", n, fnfc->filename(n));
  }
}

32.84.4.6 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.

32.84.4.7 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 a \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.

32.84.4.8 filter_value() [1/2]

int Fl_Native_File_Chooser::filter_value ( ) const

Returns which filter value was last selected by the user.
This is only valid if the chooser returns success.

32.84.4.9 filter_value() [2/2]

void Fl_Native_File_Chooser::filter_value ( int i )

Sets which filter will be initially selected.
The first filter is indexed as 0. If filter_value()==filters(), then "All Files" was chosen. If filter_value() > filters(), then
a custom filter was set.

32.84.4.10 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.
### 32.84.11 preset_file()

```cpp
void Fl_Native_File_Chooser::preset_file (
    const char * f )
```

Sets the default filename for the chooser.

Use `directory()` to set the default directory. Mainly used to preset the filename for save dialogs, and on most
platforms can be used for opening files as well.

### 32.84.12 show()

```cpp
int Fl_Native_File_Chooser::show ( )
```

Post the chooser's dialog.

Blocks until dialog has been completed or cancelled.

Returns

- 0 – user picked a file
- 1 – user cancelled
- -1 – failed; `errmsg()` has reason

### 32.84.13 title() [1/2]

```cpp
const char * Fl_Native_File_Chooser::title ( ) const
```

Get the title of the file chooser's dialog window.

Return value may be NULL if no title was set.

### 32.84.14 title() [2/2]

```cpp
void Fl_Native_File_Chooser::title ( 
    const char * t )
```

Set the title of the file chooser's dialog window.

Can be NULL if no title desired. The default title varies according to the platform, so you are advised to set the title
explicitly.

The documentation for this class was generated from the following files:

- `Fl_Native_File_Chooser.H`
- `Fl_Native_File_Chooser.cxx`
- `Fl_Native_File_Chooser_GTK.cxx`

### 32.85 Fl_Nice_Slider Class Reference

Inheritance diagram for Fl_Nice_Slider:
32.86 Fl_Output Class Reference

This widget displays a piece of text.

```cpp
#include <Fl_Output.H>
```

Inheritance diagram for Fl_Output:

```
Fl_Widget
   ▼
    Fl_Nice_Slider
       ▼
        Fl_Slider
           ▼
            Fl_Valuator
               ▼
                Fl_Widget
                   ▼
                    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

32.86.1 Detailed Description

This widget displays a piece of text.

When you set the `value()` of `Fl_Output` it does a `strcpy()` to its own storage, which is useful for program-generated values. The user may select portions of the text using the mouse and paste the contents into other fields or programs.
There is a single subclass, Fl_Multiline_Output, which allows you to display multiple lines of text. Fl_Multiline_Output does not provide scroll bars. If a more complete text editing widget is needed, use Fl_Text_Display instead.

The text may contain any characters except \0, and will correctly display anything, using \^X notation for unprintable control characters and \nnn notation for unprintable characters with the high bit set. It assumes the font can draw any characters in the ISO-Latin1 character set.

### 32.86.2 Constructor & Destructor Documentation

#### 32.86.2.1 Fl_Output()

```c
Fl_Output::Fl_Output (  
    int X, 
    int Y, 
    int W, 
    int H,  
    const char ∗ l = 0 )
```

Creates a new Fl_Output widget using the given position, size, and label string.

The default boxtype is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

The documentation for this class was generated from the following files:

- Fl_Output.H
- Fl_Input.cxx

### 32.87 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.

```c
#include <Fl_Overlay_Window.H>
```

Inheritance diagram for Fl_Overlay_Window:
Public Member Functions

- virtual Fl_Overlay_Window * as_overlay_window ()
  Return non-null if this is an Fl_Overlay_Window object.
- int can_do_overlay ()
  Returns non-zero if there's hardware overlay support.
- virtual void draw_overlay ()=0
  You must subclass Fl_Overlay_Window and provide this method.
- void flush ()
  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)
  Same as Fl_Window::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

32.87.1 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.
32.87.2 Constructor & Destructor Documentation

32.87.2.1 Fl_Overlay_Window()

Fl_Overlay_Window::Fl_Overlay_Window (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ l = 0 ) [protected]

Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.  
If the positions (x,y) are not given, then the window manager will choose them.

32.87.3 Member Function Documentation

32.87.3.1 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>.

32.87.3.2 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.

32.87.3.3 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().

32.87.3.4 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 func-

tion.  
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.
32.87.3.5 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

32.88 Fl_Pack Class Reference

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

#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

32.88.1 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()

32.88.2 Constructor & Destructor Documentation

32.88.2.1 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>
# Fl_Pack Class Reference

Represents page-structured drawing surfaces.

```cpp
#include <Fl_Pack.H>
```

## Inheritance diagram for Fl_Pack

```
Fl_Pack
  |     |
  v     v
Fl_Surface_Device
  |     |
  v     v
Fl_Widget_Surface
  |     |
  v     v
Fl_Surface_Device
  |     |
  v     v
Fl_Paged_Device
  |     |
  v     v
Fl_PostScript_File_Device
  |     |
  v     v
Fl_Printer
```

## Classes

- `struct page_format`
  
  width, height and name of a page format
Public Types

- enum Page_Format {
  A0 = 0, A1, A2, A3, A4, A5, A6, A7, A8, A9, B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, C5E, DLE, EXECUTIVE, FOLIO, LEDGER, LEGAL, LETTER, TABLOID, ENVELOPE, MEDIA = 0x1000
}

Possible page formats.

- enum Page_Layer { PORTRAIT = 0, LANDSCAPE = 0x100, REVERSED = 0x200, ORIENTATION = 0x300
}

Possible page layouts.

Public Member Functions

- virtual int begin_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  Begins a print job.

- virtual int begin_page (void)
  Begins a new printed page.

- virtual void end_job (void)
  To be called at the end of a print job.

- virtual int end_page (void)
  To be called at the end of each page.

- virtual void margins (int *left, int *top, int *right, int *bottom)
  Computes the dimensions of margins that lie between the printable page area and the full page.

- void print_widget (Fl_Widget *widget, int delta_x=0, int delta_y=0)
  Synonym of draw(Fl_Widget*, int, int)

- void print_window (Fl_Window *win, int x_offset=0, int y_offset=0)
  Synonym of draw_decorated_window(Fl_Window*, int, int)

- virtual void rotate (float angle)
  Rotates the graphics operations relatively to paper.

- virtual void scale (float scale_x, float scale_y=0.)
  Changes the scaling of page coordinates.

- int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)
  Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).

- int start_page ()
  Synonym of begin_page().

- virtual ~Fl_Paged_Device ()
  The destructor.

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

32.89.1 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.

32.89.2 Member Enumeration Documentation

32.89.2.1 Page_Format

enum Fl_Paged_Device::Page_Format
Possible page formats.
All paper formats with pre-defined width and height. The Fl_Paged_Device::page_formats array gives these widths and heights.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>A0 format.</td>
</tr>
<tr>
<td>A1</td>
<td>A1 format.</td>
</tr>
<tr>
<td>A2</td>
<td>A2 format.</td>
</tr>
<tr>
<td>A3</td>
<td>A3 format.</td>
</tr>
<tr>
<td>A4</td>
<td>A4 format.</td>
</tr>
<tr>
<td>A5</td>
<td>A5 format.</td>
</tr>
<tr>
<td>A6</td>
<td>A6 format.</td>
</tr>
<tr>
<td>A7</td>
<td>A7 format.</td>
</tr>
<tr>
<td>A8</td>
<td>A8 format.</td>
</tr>
<tr>
<td>A9</td>
<td>A9 format.</td>
</tr>
<tr>
<td>B0</td>
<td>B0 format.</td>
</tr>
<tr>
<td>B1</td>
<td>B1 format.</td>
</tr>
<tr>
<td>B2</td>
<td>B2 format.</td>
</tr>
<tr>
<td>B3</td>
<td>B3 format.</td>
</tr>
<tr>
<td>B4</td>
<td>B4 format.</td>
</tr>
<tr>
<td>B5</td>
<td>B5 format.</td>
</tr>
<tr>
<td>B6</td>
<td>B6 format.</td>
</tr>
<tr>
<td>B7</td>
<td>B7 format.</td>
</tr>
<tr>
<td>B8</td>
<td>B8 format.</td>
</tr>
<tr>
<td>B9</td>
<td>B9 format.</td>
</tr>
<tr>
<td>B10</td>
<td>B10 format.</td>
</tr>
<tr>
<td>EXECUTIVE</td>
<td>Executive format.</td>
</tr>
<tr>
<td>FOLIO</td>
<td>Folio format.</td>
</tr>
<tr>
<td>LEDGER</td>
<td>Ledger format.</td>
</tr>
<tr>
<td>LEGAL</td>
<td>Legal format.</td>
</tr>
<tr>
<td>LETTER</td>
<td>Letter format.</td>
</tr>
<tr>
<td>TABLOID</td>
<td>Tabloid format.</td>
</tr>
</tbody>
</table>

32.89.2.2 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>

#### 32.89.3 Member Function Documentation

##### 32.89.3.1 begin_job()

```cpp
int Fl_Paged_Device::begin_job (  
    int pagecount = 0,  
    int * frompage = NULL,  
    int * topage = NULL,  
    char ** perr_message = NULL) [virtual]
```

Begins a print job.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>pagecount</th>
<th>the total number of pages of the job (or 0 if you don't know the number of pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>frompage</td>
<td>if non-null, *frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>topage</td>
<td>if non-null, *topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td>out</td>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]'d after use.</td>
</tr>
</tbody>
</table>

**Returns**

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented in Fl_Printer, and Fl_PostScript_File_Device.

##### 32.89.3.2 begin_page()

```cpp
int Fl_Paged_Device::begin_page (  
    void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area. This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

**Note**

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

**Returns**

0 if OK, non-zero if any error

Reimplemented in Fl_Printer, and Fl_PostScript_File_Device.
### 32.89.3.3 end_page()

```cpp
int Fl_Paged_Device::end_page()
{
    void ) [virtual]
```

To be called at the end of each page. This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

**Note**

* end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

**Returns**

0 if OK, non-zero if any error.

Reimplemented in Fl_Printer, and Fl_PostScript_File_Device.

### 32.89.3.4 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_Printer, and Fl_PostScript_File_Device.

### 32.89.3.5 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. |

Reimplemented in Fl_Printer, and Fl_PostScript_File_Device.

### 32.89.3.6 scale()

```cpp
void Fl_Paged_Device::scale ( float scale_x,
                             float scale_y = 0. ) [virtual]
```

Generated by Doxygen
Changes the scaling of page coordinates. This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don't combine their effects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scale_x</code></td>
<td>Horizontal dimensions of plot are multiplied by this quantity.</td>
</tr>
<tr>
<td><code>scale_y</code></td>
<td>Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code>. Thus, <code>scale(factor)</code>; is equivalent to <code>scale(factor, factor)</code>;</td>
</tr>
</tbody>
</table>

Reimplemented in `Fl_Printer`, and `Fl_PostScript_File_Device`.

### 32.89.3.7 start_job()

```cpp
int Fl_Paged_Device::start_job ( int pagecount = 0, int * frompage = NULL, int * topage = NULL, char ** perr_message = NULL ) [inline]
```

Syntonym of `begin_job(int pagecount, int *frompage, int *topage, char **perr_message)`.

For API compatibility with FLTK 1.3.x

### 32.89.3.8 start_page()

```cpp
int Fl_Paged_Device::start_page ( ) [inline]
```

Syntonym 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`

### 32.90 Fl_Pixmap Class Reference

The `Fl_Pixmap` class supports caching and drawing of colormap ( pixmap) images, including transparency.

```cpp
#include <Fl_Pixmap.H>
```

Inheritance diagram for `Fl_Pixmap`:

```
            Fl_Pixmap
              |            |
              v            v
         Fl_GIF_Image     Fl_XPM_Image

```

Public Member Functions

- virtual void `color_average (Fl_Color c, float i)`
  
  The `color_average()` method averages the colors in the image with the provided FLTK color value.

- `Fl_Image * copy ()` const

    Creates a resized copy of the image.
• virtual void desaturate ()

The desaturate() method converts an image to grayscale.

• void draw (int X, int Y)
• virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)

Draws the image to the current drawing surface with a bounding box.

• Fl_Pixmap (char * const * D)

The constructors create a new pixmap from the specified XPM data.

• Fl_Pixmap (const char * const * D)

The constructors create a new pixmap from the specified XPM data.

• Fl_Pixmap (const uchar * const * D)

The constructors create a new pixmap from the specified XPM data.

• Fl_Pixmap (uchar * const * D)

The constructors create a new pixmap from the specified XPM data.

• virtual void label (Fl_Menu_Item * m)

This method is an obsolete way to set the image attribute of a menu item.

• virtual void label (Fl_Widget * w)

This method is an obsolete way to set the image attribute of a widget or menu item.

• 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

32.90.1 Detailed Description

The Fl_Pixmap class supports caching and drawing of colormap ( pixmap) images, including transparency.

32.90.2 Constructor & Destructor Documentation

32.90.2.1 Fl_Pixmap() [1/4]

Fl_Pixmap::Fl_Pixmap (char *const * D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.
32.90.2.2 Fl_Pixmap() [2/4]

Fl_Pixmap::Fl_Pixmap (uchar *const * D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.

32.90.2.3 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.

32.90.2.4 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.

32.90.3 Member Function Documentation

32.90.3.1 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 provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory. Reimplemented from Fl_Image.

32.90.3.2 copy()

Fl_Image * Fl_Pixmap::copy (int W, int H) const [virtual]
Creates a resized copy of the image. The new image should be released when you are done with it. Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling(). For the new image the following equations are true:

- w() == data_w() == \( W \)
- h() == data_h() == \( H \)
Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of `FL_RGB_Img`, `FL_SVG_Img`, `FL_Pixmap`, `FL_Bitmap`, `FL_Tiled_Img`, and `FL_Shared_Img`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `FL_GIF_Img` is an object of class `FL_Pixmap`.

### Parameters

| in | W,H | Requested width and height of the new image |

**Note**

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from `FL_Image` or any subclass your overridden methods of `FL_Image::copy() const` and `FL_Image::copy(int, int) const` must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from `FL_Image`.

#### 32.90.3.3 desaturate()

```cpp
void Fl_Pixmap::desaturate () [virtual]
```

The `desaturate()` method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from `FL_Image`.

#### 32.90.3.4 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`.

#### 32.90.3.5 label() [1/2]

```cpp
void Fl_Pixmap::label (  
   Fl_Menu_Item ∗ m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

**Deprecated** Please use `Fl_Menu_Item::image()` instead.

Reimplemented from `FL_Image`. 

---

Generated by Doxygen
32.90.3.6 label() [2/2]

```cpp
void Fl_Pixmap::label ( Fl_Widget ∗ widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

**Deprecated** Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Reimplemented from `Fl_Image`.

32.90.3.7 uncache()

```cpp
void Fl_Pixmap::uncache ( ) [virtual]
```

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented from `Fl_Image`.

The documentation for this class was generated from the following files:

- Fl_Pixmap.H
- Fl_Pixmap.cxx

### 32.91 Fl_Plugin Class Reference

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

```cpp
#include <Fl_Plugin.H>
```

Inheritance diagram for Fl_Plugin:

```
Fl_Plugin
    |          |
    |          |
    Fl_Device_Plugin
```

#### 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.

### 32.91.1 Detailed Description

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

**Fl_Plugin** and **Fl_Plugin_Manager** provide a small and simple solution for linking C++ classes at run-time, or optionally linking modules at compile time without the need to change the main application.

**Fl_Plugin_Manager** uses static initialisation to create the plugin interface early during startup. Plugins are stored in a temporary database, organized in classes.

Plugins should derive a new class from **Fl_Plugin** as a base:

```cpp
class My_Plugin : public Fl_Plugin {
public:
    My_Plugin() : Fl_Plugin("effects", "blur") { }
    void do_something(...) {};
};
My_Plugin blur_plugin();
```

Plugins can be put into modules and either linked before distribution, or loaded from dynamically linkable files. An **Fl_Plugin_Manager** is used to list and access all currently loaded plugins.

```cpp
Fl_Plugin_Manager mgr("effects");
int i, n = mgr.plugins();
for (i=0; i<n; i++) {
    My_Plugin *pin = (My_Plugin*)mgr.plugin(i);
    pin->do_something();
}
```

Generated by Doxygen
32.91.2 Constructor & Destructor Documentation

32.91.2.1 Fl_Plugin()

```cpp
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

32.92 Fl_PLUGIN_Manager Class Reference

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

```cpp
#include <Fl_Plugin.H>
```

Inheritance diagram for Fl_Plugin_Manager:

```
Fl_Preferences
   Fl_Plugin_Manager
```

Public Member Functions

- `Fl_Preferences::ID addPlugin (const char * name, Fl_Plugin * plugin)`
  
  This function adds a new plugin to the database.

- `Fl_Plugin_Manager (const char * klass)`
  
  Manage all plugins belonging to one class.

- `Fl_Plugin * plugin (const char * name)`
  
  Return the address of a plugin by name.

- `Fl_Plugin * plugin (int index)`
  
  Return the address of a plugin by index.

- `int plugins ()`
  
  Return the number of plugins in the klass.

- `~Fl_Plugin_Manager ()`
  
  Remove the plugin manager.

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.
32.92.1 Detailed Description

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

See also

`Fl_Plugin`

32.92.2 Constructor & Destructor Documentation

32.92.2.1 \~`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.

32.92.3 Member Function Documentation

32.92.3.1 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.

32.92.3.2 load()

`int Fl_Plugin_Manager::load ( const char * filename ) [static]`

Load a module from disk.
A module must be a dynamically linkable file for the given operating system. When loading a module, its `+init` function will be called which in turn calls the constructor of all statically initialized `Fl_Plugin` classes and adds them to the database.

32.92.3.3 removePlugin()

`void Fl_Plugin_Manager::removePlugin ( Fl_Preferences::ID id ) [static]`

Remove any plugin.
There is no need to call this function explicitly. Every `Fl_Plugin` destructor will call this function at destruction time.

The documentation for this class was generated from the following files:

- `Fl_Plugin.H`
- `Fl_Preferences.cxx`

32.93 FL_PNG_Image Class Reference

The `FL_PNG_Image` class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

```
#include <FL_PNG_Image.H>
```

Inheritance diagram for FL_PNG_Image:

```plaintext
Generated by Doxygen
```
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.

Additional Inherited Members

32.93.1 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.

32.93.2 Constructor & Destructor Documentation

32.93.2.1 `Fl_PNG_Image()` [1/2]

```cpp
Fl_PNG_Image::~Fl_PNG_Image (const char *filename)
```

The constructor loads the named PNG image from the given png filename.

The destructor frees all memory and server resources that are used by the image.

Use `Fl_Image::fail()` to check if `Fl_PNG_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>Name of PNG file to read</th>
</tr>
</thead>
</table>

32.93.2.2 `Fl_PNG_Image()` [2/2]

```cpp
Fl_PNG_Image::Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int maxsize)
```

Constructor that reads a PNG image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. name_png can be NULL. If a name is given, the image is added to the list of shared images (see: Fl_Shared_Image) and will be available by that name.

Parameters

| name_png | A name given to this image or NULL |
| buffer   | Pointer to the start of the PNG image in memory |
32.94 Fl_PNM_Image Class Reference

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

#include <Fl_PNM_Image.H>

Inheritance diagram for Fl_PNM_Image:

- Fl_PNM_Image
- Fl_RGB_Image
- Fl_Image

### Public Member Functions

- **Fl_PNM_Image (const char *filename)**
  
  The constructor loads the named PNM image.

### Additional Inherited Members

32.94.1 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.

32.94.2 Constructor & Destructor Documentation

32.94.2.1 Fl_PNM_Image()

```cpp
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:
32.95 Fl_Positioner Class Reference

This class is provided for Forms compatibility.

```cpp
#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(yminimum(), 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()  
  Draws the widget.

• void draw(int, int, int, int)  

• int handle(int, int, int, int)  

Additional Inherited Members

32.95.1 Detailed Description

This class is provided for Forms compatibility. It provides 2D input. It would be useful if this could be put atop another widget so that the crosshairs are on top, but this is not implemented. The color of the crosshairs is selection_color().

![Figure 32.30 Fl_Positioner](image_url)

32.95.2 Constructor & Destructor Documentation

32.95.2.1 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.

32.95.3 Member Function Documentation

32.95.3.1 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
32.96  **Fl_PostScript_File_Device**  Class Reference

To send graphical output to a PostScript file.

```cpp
#include <Fl_PostScript.H>
```

Inheritance diagram for Fl_PostScript_File_Device:

```
Fl_Surface_Device
   `-- Fl_Widget_Surface
       `-- Fl_Paged_Device
           `-- Fl_PostScript_File_Device
```

**Public Member Functions**

- `int begin_job (FILE *ps_output, int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)`

  *Beginns the session where all graphics requests will go to FILE pointer.*

- `int begin_job (int pagecount, int *from, int *to, char **perr_message)`
Don't use with this class.

- **int begin_job (int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)**

  Begins the session where all graphics requests will go to a local PostScript file.

- **int begin_page (void)**

  Begins a new printed page.

- **void close_command (Fl_PostScript_Close_Command cmd)**

  Sets the function end_job() calls to close the file().

- **virtual void end_current ()**

  FLTK calls this each time a surface ceases to be the current drawing surface.

- **void end_job (void)**

  Completes all PostScript output.

- **int end_page (void)**

  To be called at the end of each page.

- **FILE * file ()**

  Returns the underlying FILE* receiving all PostScript data.

- **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.

- **virtual void set_current ()**

  Make this surface the current drawing surface.

- **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().

- **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().

- **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

32.96.1 Detailed Description

To send graphical output to a PostScript file.

This class is used exactly as the Fl_Printer class except for the begin_job() call, two variants of which are usable and allow to specify what page format and layout are desired.

Processing of text: Text uses vectorial fonts under the X11 + Pango and the Wayland platforms. With other platforms, only text restricted to the Latin alphabet (and a few other characters listed in the table below) and to FLTK standard fonts is vectorized. All other unicode characters or all other fonts (FL_FREE_FONT and above) are output as a bitmap. FLTK standard fonts are output using the corresponding PostScript standard fonts. The Latin alphabet means all unicode characters between U+0020 and U+017F, or, in other words, the ASCII, Latin-1 Supplement and Latin Extended-A charts.

Processing of transparent Fl_RGB_Image objects: Under the X11 + Pango and the Wayland platforms, these objects are output with their exact transparency. With other platforms, these objects are drawn blended to white color. Class Fl_EPS_File_Surface's constructor allows to set another background color for blending.

Extra characters supported by standard PostScript fonts

<table>
<thead>
<tr>
<th>Char</th>
<th>Code-point</th>
<th>Name</th>
<th>Char</th>
<th>Code-point</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>U+0192</td>
<td>florin</td>
<td>&quot;</td>
<td>U+201A</td>
<td>quotescrb</td>
</tr>
<tr>
<td>^</td>
<td>U+02C6</td>
<td>circumflex</td>
<td>&quot;</td>
<td>U+201C</td>
<td>quotedl</td>
</tr>
<tr>
<td>`</td>
<td>U+02C7</td>
<td>caron</td>
<td>&quot;</td>
<td>U+201D</td>
<td>quotedbl</td>
</tr>
<tr>
<td>v</td>
<td>U+02D8</td>
<td>breve</td>
<td>&quot;</td>
<td>U+201E</td>
<td>quotedblb</td>
</tr>
<tr>
<td>'</td>
<td>U+02D9</td>
<td>dotaccent</td>
<td>†</td>
<td>U+2020</td>
<td>dagger</td>
</tr>
<tr>
<td>U+02DA</td>
<td>ring</td>
<td>‡</td>
<td>U+2021</td>
<td>daggerdb</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>U+02DB</td>
<td>ogonek</td>
<td>*</td>
<td>U+2022</td>
<td>bullet</td>
</tr>
<tr>
<td>,</td>
<td>U+02DC</td>
<td>tilde</td>
<td>...</td>
<td>U+2026</td>
<td>ellipse</td>
</tr>
<tr>
<td>”</td>
<td>U+02DD</td>
<td>hungarumlaut</td>
<td>%</td>
<td>U+2030</td>
<td>perthousand</td>
</tr>
<tr>
<td>-</td>
<td>U+2013</td>
<td>endash</td>
<td>′</td>
<td>U+2039</td>
<td>guilingsl</td>
</tr>
<tr>
<td>-</td>
<td>U+2014</td>
<td>emdash</td>
<td>′</td>
<td>U+203A</td>
<td>guilingsr</td>
</tr>
<tr>
<td>'</td>
<td>U+2018</td>
<td>quoteleft</td>
<td>′</td>
<td>U+2044</td>
<td>fraction</td>
</tr>
<tr>
<td>'</td>
<td>U+2019</td>
<td>quoteright</td>
<td>€</td>
<td>U+20AC</td>
<td>Euro</td>
</tr>
</tbody>
</table>

32.96.2 Member Function Documentation

32.96.2.1 begin_job() [1/2]

```c
int Fl_PostScript_File_Device::begin_job (FILE * ps_output,
    int pagecount = 0,
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to FILE pointer.

This member function prevents end_job() from closing ps_output, so the user can check with ferror(ps_output) for output errors.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps_output</td>
<td>A writable FILE pointer that will receive PostScript output and that should not be closed until after end_job() has been called.</td>
</tr>
<tr>
<td>pagecount</td>
<td>The total number of pages to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td>format</td>
<td>Desired page format.</td>
</tr>
<tr>
<td>layout</td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

always 0.

### 32.96.2.2 begin_job() (2/2)

```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 to select an output PostScript file. This member function makes end_job() close the resulting PostScript file and display an alert message with fl_alert() in case of any output error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecount</td>
<td>The total number of pages to be created. Use 0 if this number is unknown when this function is called.</td>
</tr>
<tr>
<td>format</td>
<td>Desired page format.</td>
</tr>
<tr>
<td>layout</td>
<td>Desired page layout.</td>
</tr>
</tbody>
</table>

Returns

0 if OK, 1 if user cancelled the file dialog, 2 if fopen failed on user-selected output file.

### 32.96.2.3 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. This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

Generated by Doxygen
32.96.2.4  end_current()

virtual void Fl_PostScript_File_Device::end_current ( ) [virtual]
FLTK calls this each time a surface ceases to be the current drawing surface.
This member function is mostly of interest to developers of new Fl_Surface_Device derived classes. It allows to
perform surface-specific operations necessary when this surface ceases to be current. Each implementation should
end with a call to Fl_Surface_Device::end_current().
Reimplemented from Fl_Surface_Device.

32.96.2.5  end_job()

void Fl_PostScript_File_Device::end_job ( ) [virtual]
Completes all PostScript output.
This also closes with fclose() the underlying file() unless close_command() was used to set another function.
Reimplemented from Fl_Paged_Device.

32.96.2.6  end_page()

int Fl_PostScript_File_Device::end_page ( ) [virtual]
To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page()
open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to
Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.
Reimplemented from Fl_Paged_Device.

32.96.2.7  margins()

void Fl_PostScript_File_Device::margins ( int*out left,
                        int*out top,
                        int*out right,
                        int*out 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>out</td>
<td>If non-null, +left is set to the left margin size.</td>
</tr>
<tr>
<td>out</td>
<td>If non-null, +top is set to the top margin size.</td>
</tr>
<tr>
<td>out</td>
<td>If non-null, +right is set to the right margin size.</td>
</tr>
<tr>
<td>out</td>
<td>If non-null, +bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.
32.96.2.8 origin() [1/2]

void Fl_PostScript_File_Device::origin (  
      int * x,  
      int * y ) [virtual]

Computes the coordinates of the current origin of graphics functions.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>x,y</td>
<td></td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.

32.96.2.9 origin() [2/2]

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>x,y</td>
<td></td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Widget_Surface.

32.96.2.10 printable_rect()

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.

32.96.2.11 rotate()

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>angle</strong></td>
<td></td>
<td>Rotation angle in counter-clockwise degrees.</td>
</tr>
</tbody>
</table>
Reimplemented from Fl_Paged_Device.

### 32.96.2.12 scale()

```cpp
void Fl_PostScript_File_Device::scale {
    float scale_x,
    float scale_y = 0. } [virtual]
```

Changes the scaling of page coordinates. This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don’t combine their effects.

**Parameters**

- `scale←_x` Horizontal dimensions of plot are multiplied by this quantity.
- `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.

### 32.96.2.13 set_current()

```cpp
virtual void Fl_PostScript_File_Device::set_current { 
    void ) [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current()` / `Fl_Surface_Device::pop_current()`.

**Note**

It’s recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device() -> set_current();`. Don’t do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current()` / `Fl_Surface_Device::pop_current()`.

Reimplemented from Fl_Surface_Device.

### 32.96.2.14 start_job() [1/2]

```cpp
int Fl_PostScript_File_Device::start_job {
    FILE * ps_output,
    int pagecount = 0,
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT ) [inline]
```

Synonym of `begin_job()`.
For API compatibility with FLTK 1.3.x
32.96.2.15  start_job() [2/2]

```c
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

32.96.2.16  translate()

```c
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

32.97  Fl_Preferences Class Reference

Fl_Preferences store user settings between application starts.

```c
#include <Fl_Preferences.H>
```

Inheritance diagram for Fl_Preferences:

```
Fl_Preferences
| Fl_Plugin_Manager
```

Classes
- struct Entry
- class Name

"Name" provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.
- class Node
- class RootNode

Public Types
- typedef void * ID

Every Fl_Preferences-Group has a unique ID.
- enum Root {
  UNKNOWN_ROOT_TYPE = -1 , SYSTEM = 0 , USER , MEMORY ,  
  ROOT_MASK = 0x00FF , CORE = 0x0100 , C_LOCALE = 0x1000 , SYSTEM_L = SYSTEM | C_LOCALE ,  
  USER_L = USER | C_LOCALE , CORE_SYSTEM_L = CORE | SYSTEM_L , CORE_USER_L = CORE | USER_L ,  
  CORE_SYSTEM = CORE | SYSTEM ,  
  CORE_USER = CORE | USER }

Define the scope of the preferences.
Public Member Functions

- char clear ()
  Delete all groups and all entries.

- char delete_all_entries ()
  Delete all entries.

- char delete_all_groups ()
  Delete all groups.

- char delete_entry (const char *entry)
  Deletes a single name/value pair.

- char delete_group (const char *group)
  Deletes a group.

- int dirty ()
  Check if there were changes to the database that need to be written to disk.

- int entries ()
  Returns the number of entries (name/value pairs) in a group.

- const char * entry (int index)
  Returns the name of an entry.

- char entry_exists (const char *key)
  Returns non-zero if an entry with this name exists.

- Root filename (char *buffer, size_t buffer_size)
  Return the file name and path to the preference file.

- Fl_Preferences (const char *path, const char *vendor, const char *application)
  Use this constructor to create or read a preference file at an arbitrary position in the file system.

- Fl_Preferences (const Fl_Preferences &)
  Create another reference to a Preferences group.

- Fl_Preferences (Fl_Preferences &parent, const char *group)
  Generate or read a new group of entries within another group.

- Fl_Preferences (Fl_Preferences &parent, int groupIndex)
  Open a child group using a given index.

- Fl_Preferences (Fl_Preferences *parent, const char *group)
  Create or access a group of preferences using a name.

- Fl_Preferences (Fl_Preferences *parent, int groupIndex)

- Fl_Preferences (ID id)
  Create a new dataset access point using a dataset ID.

- Fl_Preferences (Root root, const char *vendor, const char *application)
  The constructor creates a group that manages key/value pairs and child groups.

- int flush ()
  Writes preferences to disk if they were modified.

- char get (const char *entry, char *)&value, const char *defaultValue)
  Reads an entry from the group.

- char get (const char *entry, char *value, const char *defaultValue, int maxSize)
  Reads an entry from the group.

- char get (const char *entry, double &value, double defaultValue)
  Reads an entry from the group.

- char get (const char *entry, float &value, float defaultValue)
  Reads an entry from the group.

- char get (const char *entry, int &value, int defaultValue)
  Reads an entry from the group.

- char get (const char *entry, void *)&value, const void *defaultValue, int defaultSize)
  Reads an entry from the group.
**Static Public Member Functions**

- static unsigned int file_access ()
  
  *Return the current file access permissions for the FLTK preferences system.*

- static void file_access (unsigned int flags)
  
  *Tell the FLTK preferences system which files in the file system it may read, create, or write.*

- static Root filename (char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application)
  
  *Determine the file name and path to preferences that would be opened with these parameters.*

- static const char * new_UUID ()
  
  *Returns a UUID as generated by the system.*

- static char remove (ID id_)
  
  *Remove the group with this ID from a database.*
Static Public Attributes

- static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK
  
  Set this to give FLTK and applications permission to read, write, and create preference files.

- static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK
  
  Set this to allow FLTK and applications to read preference files.

- static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK
  
  Set this to allow FLTK and applications to create and write preference files.

- static const unsigned int APP_OK = SYSTEM_OK | USER_OK
  
  Set this if it is OK for applications to read, create, and write any kind of preference files.

- static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK
  
  Set this if it is OK for FLTK to read, create, or write preference files.

- static const unsigned int CORE_READ_OK = 0x0010
  
  Set this if it is OK for FLTK to read preference files.

- static const unsigned int CORE_WRITE_OK = 0x0020
  
  Set this if it is OK for FLTK to create or write preference files.

- static const unsigned int NONE = 0x0000
  
  Set this if no call to Fl_Preferences shall access the file system.

- static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK
  
  Set this if it is OK for applications to read, create, and write system wide preference files.

- static const unsigned int SYSTEM_READ_OK = 0x0004
  
  Set this if it is OK for applications to read system wide preference files.

- static const unsigned int SYSTEM_WRITE_OK = 0x0008
  
  Set this if it is OK for applications to create and write system wide preference files.

- static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK
  
  Set this if it is OK for applications to read, create, and write user preference files.

- static const unsigned int USER_READ_OK = 0x0001
  
  Set this if it is OK for applications to read user preference files.

- static const unsigned int USER_WRITE_OK = 0x0002
  
  Set this if it is OK for applications to create and write user preference files.

Protected Attributes

- Node * node
- RootNode * rootNode

Friends

- class Node
- class RootNode

32.97.1 Detailed Description

Fl_Preferences store user settings between application starts. Fl_Preferences are similar to the Registry on Windows and Preferences on MacOS, providing a simple method to store customisable user settings between app launches, for instance the previous window position or a history of previously used documents.

Preferences are organized in a hierarchy of groups. Every group can contain more groups and any number of key/value pairs. Keys can be text strings containing ASCII letters, digits, periods, and underscores. Forward slashes in a key name are treated as subgroups, i.e. the key 'window/width' would actually refer to the key 'width' inside the group 'window'.

Keys usually have a unique name within their group. Duplicate keys are possible though and can be accessed using the index based functions.
A value can be an UTF-8 string. Control characters and UTF-8 sequences are stored as octal values. Long strings are wrapped at the line ending and will be reassembled when reading the file back. Several methods allow setting and getting numerical values and binary data. Preferences are stored in text files that can be edited manually if needed. The file format is easy to read and relatively forgiving. Preference files are the same on all platforms. User comments in preference files are preserved. Filenames are unique for each application by using a vendor/application naming scheme. The user must provide default values for all entries to ensure proper operation should preferences be corrupted or not yet exist. FLTK preferences are not meant to replace a fully features database. No merging of data takes place. If several instances of an app access the same database at the same time, only the most recent changes will persist.

Preferences should not be used to store document data. The .prefs file should be kept small for performance reasons. One application can have multiple preference files. Extensive binary data however should be stored in separate files: see `FL_Preferences::get_userdata_path()`.

FL_Preferences are not thread-safe. They can temporarily change the locale on some platforms during read and write access, which also changes it temporarily in other threads of the same app.

Typically a preferences database is read at startup, and then reopened and written at app shutdown:

```c
int appWindowWidth, appWindowHeight;

void launch() {
    FL_Preferences app(Fl_Preferences::USER_L, "matthiasm.com", "hello");
    // 'app' constructor will be called, reading data from .prefs file
    FL_Preferences window(app, "window");
    window.get("height", appWindowHeight, 600);
    // 'app' destructor will be called. This will write data to the
    // .prefs file if any preferences were changed or added
}

void quit() {
    FL_Preferences app(Fl_Preferences::USER_L, "matthiasm.com", "hello");
    FL_Preferences window(app, "window");
    window.set("width", appWindowWidth);
    window.set("height", appWindowHeight);
}
```

See also

`FL_Preferences::Fl_Preferences(Fl_Preferences *parent, const char *group)` for more details on memory mapped preferences.

Note

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see `FL_Preferences::Fl_Preferences(Root, const char *, const char *)` for details.

Starting with FLTK 1.4, preference files should be created with `SYSTEM_L` or `USER_L` to be interchangeable between computers with differing locale settings. The legacy modes, `LOCAL` and `SYSTEM`, will read and write floating point values using the decimal point of the current locale. As a result, a fp-value would be written ‘3,1415’ on a German machine, and would be read back as ‘3.0’ on a US machine because the comma would not be recognized as an alternative decimal point.

### 32.97.2 Member Typedef Documentation

#### 32.97.2.1 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.
32.97.3  Member Enumeration Documentation

32.97.3.1  Root

enum Fl_Preferences::Root

Define the scope of the preferences.

Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN_ROOT_TYPE</td>
<td>Returned if storage could not be determined.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Preferences are used system-wide. Deprecated, see SYSTEM_L.</td>
</tr>
<tr>
<td>USER</td>
<td>Preferences apply only to the current user. Deprecated, see USER_L.</td>
</tr>
<tr>
<td>MEMORY</td>
<td>Returned if querying memory mapped preferences.</td>
</tr>
<tr>
<td>ROOT_MASK</td>
<td>Mask for the values above.</td>
</tr>
<tr>
<td>CORE</td>
<td>OR'd by FLTK to read and write core library preferences and options.</td>
</tr>
<tr>
<td>C_LOCALE</td>
<td>This flag should always be set, it makes sure that floating point values are written correctly independently of the current locale.</td>
</tr>
<tr>
<td>SYSTEM_L</td>
<td>Preferences are used system-wide, locale independent.</td>
</tr>
<tr>
<td>USER_L</td>
<td>Preferences apply only to the current user, locale independent.</td>
</tr>
<tr>
<td>CORE_SYSTEM_L</td>
<td>Same as CORE</td>
</tr>
<tr>
<td>CORE_USER_L</td>
<td>Same as CORE</td>
</tr>
<tr>
<td>CORE_SYSTEM</td>
<td>Deprecated, same as CORE</td>
</tr>
<tr>
<td>CORE_USER</td>
<td>Deprecated, same as CORE</td>
</tr>
</tbody>
</table>

32.97.4  Constructor & Destructor Documentation

32.97.4.1  Fl_Preferences() [1/7]

Fl_Preferences::Fl_Preferences ( 
  Root root, 
  const char * vendor, 
  const char * application )

The constructor creates a group that manages key/value pairs and child groups. Preferences can be stored per user using the root type Fl_Preferences::USER_L, or stored system-wide using Fl_Preferences::SYSTEM_L.

Groups and key/value pairs can be read and written randomly. Reading undefined values will return the default value. Writing undefined values will create all required groups and key/value pairs.

This constructor creates the base instance for all following entries and reads the database from disk into memory if it exists. The vendor argument is a unique text string identifying the development team or vendor of an application. A domain name or an EMail address (replacing the @ with a .) are great unique names, e.g. "research.matthiasm.com" or "fluid.fltk.org". The application argument can be the working title or final name of your application. Both vendor and application must be valid UNIX path segments as they become parts of the preference file path and may contain forward slashes to create deeper file structures.

Note

On Windows, the directory is constructed by querying the Common AppData or AppData key of the Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders registry entry. The filename and path is then constructed as $(query)/$(vendor)/$(application).prefs. If the query call fails, data will be stored in RAM only. It will be lost when the app exits.
In FLTK versions before 1.4.0, if querying the registry failed, preferences would be written to C:\FLTK\$(vendor)\$(application).prefs.

Note

On Linux, the USER directory is constructed by reading $HOME. If $HOME is not set or not pointing to an existing directory, FLTK will check the path member of the passwd struct returned by getpwuid(getuid()). If all attempts fail, data will be stored in RAM only and be lost when the app exits.

The SYSTEM preference filename is hardcoded as /etc/fltk/$(vendor)/$(application).prefs. For backward compatibility, the old USER.prefs file naming scheme $(directory)/.fltk/$(vendor)/$(application).prefs is checked first. If that file does not exist, the environment variable $XDG_CONFIG_HOME is read as a base directory. If $XDG_CONFIG_HOME not set, the base directory defaults to $HOME/.config/. The user preferences will be stored in $(directory)/$(vendor)/$(application).prefs. The user data path will be $(directory)/$(vendor)/$(application)/. In FLTK versions before 1.4.0, if $HOME was not set, the USER path would be empty, generating $(vendor)/$(application).prefs, which was used relative to the current working directory.

Note

On macOS, the USER directory is constructed by reading $HOME. If $HOME is not set or not pointing to an existing directory, we check the path returned by NSHomeDirectory(), and finally checking the path member of the passwd struct returned by getpwuid(getuid()). If all attempts fail, data will be stored in RAM only and be lost when the app exits. The filename and path is then constructed as $(directory)/Library/Preferences/$(vendor)/$(application).prefs. The SYSTEM directory is hardcoded as /Library/Preferences/$(vendor)/$(application).prefs. In FLTK versions before 1.4.0, if $HOME was not set, the USER path would be NULL, generating <null>/Library/Preferences/$(vendor)/$(application).prefs, which would silently fail to create a preference file.

Parameters

| in | root | can be USER_L or SYSTEM_L for user specific or system wide preferences |
| in | vendor | unique text describing the company or author of this file, must be a valid filepath segment |
| in | application | unique text describing the application, must be a valid filepath segment |

See also

Fl_Preferences(Fl_Preferences *parent, const char *group) with parent set to NULL

32.97.4.2 Fl_Preferences() [2/7]

Fl_Preferences::Fl_Preferences (    const char * path,    const char * vendor,    const char * application )

Use this constructor to create or read a preference file at an arbitrary position in the file system. The file name is generated in the form $(path)/$(application).prefs. If application is NULL, path is taken literally as the file path and name.

Parameters

| in | path | path to the directory that contains the preference file |
| in | vendor | unique text describing the company or author of this file, must be a valid filepath segment |
| in | application | unique text describing the application, must be a valid filepath segment |
32.97.4.3 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>in</th>
<th>parent</th>
<th>reference object for the new group</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>group</td>
<td>name of the group to access (may contain '/')</td>
</tr>
</tbody>
</table>

32.97.4.4 Fl_PREFERENCES() [4/7]

Fl_PREFERENCES::Fl_PREFERENCES (  
    Fl_PREFERENCES ∗ parent,  
    const char ∗ group )

Create or access a group of preferences using a name. 
Parent should point to a previously created parent preferences group to create a preferences hierarchy. 
If parent is set to NULL, an unnamed database will be accessed that exists only in local memory and is not 
associated with a file on disk. The root type of this database is set to Fl_PREFERENCES::MEMORY.

- the memory database is not shared among multiple instances of the same app
- memory databases are not thread safe
- all data will be lost when the app quits

void some_function() {  
    Fl_PREFERENCES guide( NULL, "Guide" );  
    guide.set("answer", 42);  
}

void other_function() {  
    int x;  
    Fl_PREFERENCES guide( NULL, "Guide" );  
    guide.get("answer", x, -1);  
}

FLTK uses the memory database to manage plugins. See Fl_Plugin.

Parameters

| in  | parent | the parameter parent is a pointer to the parent group. If parent is NULL, the new 
|     |        | preferences item refers to an application internal database ("runtime prefs") which exists only 
|     |        | once, and remains in RAM only until the application quits. This database is used to manage 
|     |        | plugins and other data indexes by strings. Runtime prefs are not thread-safe. |
| in  | group  | a group name that is used as a key into the database |

See also

    Fl_PREFERENCES( Fl_PREFERENCES&, const char ∗ group )

32.97.4.5 Fl_PREFERENCES() [5/7]

Fl_PREFERENCES::Fl_PREFERENCES (  

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>parent</code></td>
<td>reference object for the new group</td>
</tr>
<tr>
<td>in <code>groupIndex</code></td>
<td>zero based index into child groups</td>
</tr>
</tbody>
</table>

### 32.97.4.6 Fl_Preferences() [6/7]

```
Fl_Preferences::Fl_Preferences (  
    Fl_Preferences * parent,  
    int groupIndex )
```

See also

```
Fl_Preferences( Fl_Preferences&, int groupIndex )
```

### 32.97.4.7 Fl_Preferences() [7/7]

```
Fl_Preferences::Fl_Preferences (  
    Fl_Preferences::ID id )
```

Create a new dataset access point using a dataset ID. ID's are a great way to remember shortcuts to database entries that are deeply nested in a preferences database, as long as the database root is not deleted. An ID can be retrieved from any `Fl_Preferences` dataset, and can then be used to create multiple new references to the same dataset. ID's can be very helpful when put into the `user_data()` field of widget callbacks.

### 32.97.4.8 ~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 preference file and deletes all internal databases. The destructor does not remove any data from the database. It merely deletes your reference to the database.

### 32.97.5 Member Function Documentation

#### 32.97.5.1 delete_entry()

```
char Fl_Preferences::delete_entry (  
    const char * key )
```

Deletes a single name/value pair. This function removes the entry `key` from the database.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>key</code></td>
<td>name of entry to delete</td>
</tr>
</tbody>
</table>
32.97.5.2 delete_group()

char Fl_Preferences::delete_group (const char * group )

Deletes a group.
Removes a group and all keys and groups within that group from the database.

Parameters

| in | group | name of the group to delete |

Returns

0 if call failed

32.97.5.3 dirty()

int Fl_Preferences::dirty ( )

Check if there were changes to the database that need to be written to disk.

Returns

1 if the database will be written to disk by flush or destructor.

0 if the database is unchanged since the last write operation.

-1 if there is an internal database error.

32.97.5.4 entries()

int Fl_Preferences::entries ( )

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

32.97.5.5 entry()

const char * Fl_Preferences::entry ( int index )

Returns the name of an entry.
There is no guaranteed order of entry names. The index must be within the range given by entries().

Parameters

| in | index | number indexing the requested entry |
Returns

pointer to value cstring

### 32.97.5.6 entry_exists()

```cpp
cchar Fl_Preferences::entry_exists {
    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

### 32.97.5.7 file_access() [1/2]

```cpp
unsigned int Fl_Preferences::file_access ( ) [static]
```

Return the current file access permissions for the FLTK preferences system.

See also

Fl_Preferences::file_access(unsigned int)

### 32.97.5.8 file_access() [2/2]

```cpp
void Fl_Preferences::file_access {
    unsigned int flags ) [static]
```

Tell the FLTK preferences system which files in the file system it may read, create, or write.

The FLTK core library will try to read or even create or write preference files when calling Fl::option(), Fl_File_Chooser, the printing panel, and possibly some other internal functions. If your application wants to keep FLTK from touching the file system, call this function before making any other FLTK calls:

// neither FLTK nor the app may read, create, or write preference files
Fl_Preferences::file_access( Fl_PREFERENCES::NONE );

or

// 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:

Fl_Preferences::file_access( Fl_PREFERENCES::file_access() &~ Fl_PREFERENCES::SYSTEM_WRITE );

If preferences are created using a filename (instead of Fl_PREFERENCES::USER or Fl_PREFERENCES::SYSTEM), file access is handled as if the Fl_PREFERENCES::USER flag was set.

See also

Fl_PREFERENCES::NONE and others for a list of flags.

Fl_PREFERENCES::file_access()

### 32.97.5.9 filename() [1/2]

```cpp
Fl_Preferences::Root Fl_Preferences::filename {
    char * buffer,
    size_t buffer_size )
```
Return the file name and path to the preference file.
If the preferences have not changed or have not been flushed, the file or directory may not have been created yet.

Parameters

<table>
<thead>
<tr>
<th>out buffer</th>
<th>write the resulting path into this buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in buffer_size</td>
<td>size of the buffer in bytes</td>
</tr>
</tbody>
</table>

Returns

the root type at creation type, or MEMORY for runtime prefs, it does not return CORE or LOCALE flags.

32.97.5.10  filename() [2/2]

```cpp
Fl_Preferences::Root Fl_Preferences::filename (char * buffer, size_t buffer_size, Root root, const char * vendor, const char * application) [static]
```

Determine the file name and path to preferences that would be openend with these parameters. Find the possible location of a preference file on disk without touching any of the pathname components. This can be used to check if a preference file already exists.

Parameters

<table>
<thead>
<tr>
<th>out buffer</th>
<th>write the resulting path into this buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in buffer_size</td>
<td>size of the buffer in bytes</td>
</tr>
<tr>
<td>in root</td>
<td>can be USER_L or SYSTEM_L for user specific or system wide preferences</td>
</tr>
<tr>
<td>in vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>in application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>

Returns

the input root value, or Fl_Preferences::UNKNOWN_ROOT_TYPE if the path could not be determined.

See also

Fl_Preferences( Root root, const char *vendor, const char *application )

32.97.5.11  flush()

```cpp
int Fl_Preferences::flush ()
```

Writes preferences to disk if they were modified. This method can be used to verify that writing a preference file went well. Deleting the base preferences object will also write the contents of the database to disk.

Returns

-1 if anything went wrong, i.e. file could not be opened, permissions blocked writing, etc.
0 if the file was written to disk. This does not check if the disk ran out of space and the file is truncated.
1 if no data was written to the database and no write attempt to disk was made.
32.97.5.12  get() [1/8]

char Fl_Preferences::get (  
    const char * key,  
    char * & text,  
    const char * defaultValue )

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). get() allocates memory of sufficient size to hold the value. The buffer must be free’d by the developer using 'free(value)'.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>text</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

32.97.5.13  get() [2/8]

char Fl_Preferences::get (  
    const char * key,  
    char * text,  
    const char * defaultValue,  
    int maxSize )

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). 'maxSize' is the maximum length of text that will be read. The text buffer must allow for one additional byte for a trailing zero.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>text</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>maxSize</td>
<td>maximum length of value plus one byte for a trailing zero</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

32.97.5.14  get() [3/8]

char Fl_Preferences::get (  
    const char * key,  
    double & value,  
    double defaultValue )

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

32.97.5.15 get() [4/8]

```cpp
char Fl_Preferences::get ( 
    const char * key, 
    float & value, 
    float defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

32.97.5.16 get() [5/8]

```cpp
char Fl_Preferences::get ( 
    const char * key, 
    int & value, 
    int defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).
Returns

0 if the default value was used

3.97.5.17 get() [6/8]
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

3.97.5.18 get() [7/8]
char Fl_Preferences::get (  
    const char * key,  
    void * data,  
    const void * defaultValue,  
    int defaultSize,  
    int * maxSize )

Reads a binary entry from the group, encoded in hexadecimal blocks.
A binary (not hex) default value can be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). maxSize is the maximum length of text that will be read and returns the actual number of bytes read.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>value returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
<tr>
<td>in,out</td>
<td>maxSize</td>
<td>maximum length of value and actual number of bytes set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used
32.97.5.19  get() [8/8]
char Fl_Preferences::get (  
    const char * key,  
    void * data,  
    const void * defaultValue,  
    int defaultSize,  
    int maxSize )

Reads a binary entry from the group, encoded in hexadecimal blocks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>value returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value</td>
</tr>
<tr>
<td>in</td>
<td>defaultSize</td>
<td>size of default value array</td>
</tr>
<tr>
<td>in</td>
<td>maxSize</td>
<td>maximum length of value, to receive the number of bytes read, use the function below instead.</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

See also

Fl_Preferences::get( const char *key, void *data, const void *defaultValue, int defaultSize, int *maxSize )

32.97.5.20  get_userdata_path()
char Fl_Preferences::get_userdata_path (  
    char * path,  
    int pathlen )

Creates a path that is related to the preference file and that is usable for additional application data.

This function creates a directory that is named after the preferences database without the .prefs extension and located in the same directory. It then fills the given buffer with the complete path name.

There is no way to verify that the path name fit into the buffer. If the name is too long, it will be clipped.

This function can be used with direct paths that don't end in .prefs . getUserDataPath() will remove any extension and end the path with a /. If the file name has no extension, getUserDataPath() will append .data/ to the path name.

Example:
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( path, FL_PATH_MAX );
creates the preferences database in the directory (User 'matt' on Linux):
/Users/matt/.fltk/matthiasm.com/test.prefs
..and returns the userdata path:
/Users/matt/.fltk/matthiasm.com/test/

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>path</th>
<th>buffer for user data path</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pathlen</td>
<td>size of path buffer (should be at least FL_PATH_MAX )</td>
</tr>
</tbody>
</table>
Returns

1 if there is no filename \((path\) will be unmodified)
1 if \(pathlen\) is 0 \((path\) will be unmodified)
1 if a path was created successfully, \(path\) will contain the path name ending in a '/'
0 if path was not created for some reason; \(path\) will contain the pathname that could not be created

See also

\texttt{Fl\_Preferences::Fl\_Preferences(Root, const char*, const char*)}

### 32.97.5.21 \texttt{group()}

\texttt{const char * Fl\_Preferences::group (int num\_group)}

Returns the name of the \(N\)th \((num\_group)\) group.
There is no guaranteed order of group names. The index must be within the range given by \texttt{groups()}.  

Parameters

\begin{verbatim}
| in  | num_group | number indexing the requested group |
\end{verbatim}

Returns

'C' string pointer to the group name

### 32.97.5.22 \texttt{group_exists()}

\texttt{char Fl\_Preferences::group\_exists (const char * key)}

Returns non-zero if a group with this name exists.
Group names are relative to the \texttt{Fl\_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

\begin{verbatim}
| in  | key | name of group that is searched for |
\end{verbatim}

Returns

0 if no group by that name was found

### 32.97.5.23 \texttt{groups()}

\texttt{int Fl\_Preferences::groups ( )}

Returns the number of groups that are contained within a group.

Returns

0 for no groups at all
32.97.5.24 new_UUID()

const char * Fl_Preferences::new_UUID ( ) [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!

32.97.5.25 set() [1/7]

cchar 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 preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>text</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

32.97.5.26 set() [2/7]

cchar Fl_Preferences::set (  
    const char * key,  
    const void * data,  
    int dsize )
Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>dsize</td>
<td>size of data array</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

32.97.5.27 set() [3/7]

cchar Fl_Preferences::set (
const char * key,
    double value )

Sets an entry (name/value pair). 
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

32.97.5.28  set() [4/7]

char Fl_Preferences::set ( 
    const char * key,
    double value,
    int precision )

Sets an entry (name/value pair). 
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
<td>number of decimal digits to represent value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

32.97.5.29  set() [5/7]

char Fl_Preferences::set ( 
    const char * key,
    float value )

Sets an entry (name/value pair). 
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed
32.97.5.30  set() [6/7]

```c
char Fl_Preferences::set ( const char * key, float value, int precision )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
<td>number of decimal digits to represent value</td>
</tr>
</tbody>
</table>

**Returns**

0 if setting the value failed

32.97.5.31  set() [7/7]

```c
char Fl_Preferences::set ( const char * key, int value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value</td>
</tr>
</tbody>
</table>

**Returns**

0 if setting the value failed

32.97.5.32  size()

```c
int Fl_Preferences::size ( const char * key )
```

Returns the size of the value part of an entry.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
</table>
32.97.6 Member Data Documentation

32.97.6.1 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.

32.97.6.2 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.

32.97.6.3 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

32.98 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 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"**
  
  [this text may be customized at run-time]

- **static const char * dialog_from = "From:"**
  
  [this text may be customized at run-time]

- **static const char * dialog_to = "To:"**
  
  [this text may be customized at run-time]

- **static const char * dialog_properties = "Properties..."**
  
  [this text may be customized at run-time]

- **static const char * dialog_copyNo = "# Copies:"**
  
  [this text may be customized at run-time]

- **static const char * dialog_print_button = "Print"**
  
  [this text may be customized at run-time]

- **static const char * dialog_cancel_button = "Cancel"**
  
  [this text may be customized at run-time]


Additional Inherited Members

32.98.1 Detailed Description

OS-independent print support.

**Fl_Printer** allows to use all drawing, color, text, image, and clip FLTK functions, and to have them operate on printed page(s). There are two main, non exclusive, ways to use it:

- Print any widget (standard, custom, **Fl_Window**, **Fl_Gl_Window**) as it appears on screen, with optional translation, scaling and rotation. This is done by calling `print_widget()`, `print_window()` or `print_window_part()`.

- Use a series of FLTK graphics commands (e.g., font, text, lines, colors, clip, image) to compose a page appropriately shaped for printing.

In both cases, begin by `begin_job()`, `begin_page()`, `printable_rect()` and `origin()` calls and finish by `end_page()` and `end_job()` calls.

Example of use: print a widget centered in a page

```c++
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>

int width, height;
Fl_Widget *widget = ... // a widget we want printed
Fl_Printer *printer = new Fl_Printer();
if (printer->begin_job(1) == 0) {
  printer->begin_page();
  printer->printable_rect(&width, &height);
  fl_color(FL_BLACK);
  fl_line_style(FL_SOLID, 2);
  fl_rect(0, 0, width, height);
  fl_font(FL_COURIER, 12);
  time_t now; time(&now); fl_draw(ctime(&now), 0, fl_height());
  printer->origin(width/2, height/2);
  printer->print_widget(widget, -widget->w()/2, -widget->h()/2);
  printer->end_page();
  printer->end_job();
}
delete printer;
```

Platform specifics

- X11 and Wayland platforms:
  - FLTK expresses all graphics data using (Level 2) PostScript and sends that to the selected printer. See class **Fl_PostScript_File_Device** for a description of how text and transparent images appear in print.
  - If the GTK library is available at run-time, class **Fl_Printer** runs GTK's printer dialog which allows to set printer, paper size and orientation.
  - Under the KDE desktop, **Fl_Printer** runs the `kdialog` command to create KDE-styled file dialogs if that command is available at run-time, unless FLTK was built with CMake and option `OPTION_USE_KDIALOG` turned off. In that case, **Fl_Printer** attempts to run the GTK dialog.
  - If the GTK library is not available, or if `FL::option(FL::OPTION_PRINTER_USES_GTK)` has been turned off, class **Fl_Printer** runs FLTK's print dialog.
- Unless it has been previously changed, the default paper size is A4. To change that, press the "Properties" button of the "Print" dialog window opened by an `Fl_Printer::begin_job()` call. This opens a "Printer Properties" window where it's possible to select the adequate paper size. Finally press the "Save" button therein to assign the chosen paper size to the chosen printer for this and all further print operations.

- Use the static public attributes of this class to set the print dialog to other languages than English. For example, the "Printer:" dialog item `Fl_Printer::dialog_printer` can be set to French with:
  ```c++
  Fl_Printer::dialog_printer = "Imprimante;";
  ```
  before creation of the `Fl_Printer` object.

- Use `Fl_PostScript_File_Device::file_chooser_title` to customize the title of the file chooser dialog that opens when using the "Print To File" option of the print dialog.

- Windows platform: Transparent `Fl_RGB_Images` don't print with exact transparency on most printers (a workaround is to use `print_window_part()`). `Fl_RGB_Images` don't `rotate()` well.

- Mac OS X platform: all graphics requests print as on display and accept rotation and scaling.

### 32.98.2 Member Function Documentation

#### 32.98.2.1 begin_job()

```c++
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>In</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecount</td>
<td>the total number of pages of the job (or 0 if you don't know the number of pages)</td>
</tr>
<tr>
<td>frompage</td>
<td>if non-null, *frompage is set to the first page the user wants printed</td>
</tr>
<tr>
<td>topage</td>
<td>if non-null, *topage is set to the last page the user wants printed</td>
</tr>
<tr>
<td>perr_message</td>
<td>if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]ed 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`.

#### 32.98.2.2 begin_page()

```c++
int Fl_Printer::begin_page (  
    void ) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.

This function also makes this surface the current drawing surface with `Fl_Surface_Device::push_current()`.

**Note**

`begin_page()` calls `Fl_Surface_Device::push_current()` and leaves this device as the active surface. If any calls between `begin_page()` and `end_page()` open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to `Fl_Surface_Device::pop_current()` and a call to `Fl_Surface_Device::push_current()`, or the content of the dialog box will be rendered to the printer instead of the screen.
Returns
0 if OK, non-zero if any error
Reimplemented from `Fl_Paged_Device`.

### 32.98.2.3 end_page()

```cpp
int Fl_Printer::end_page ( void ) [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with `Fl_Surface_Device::pop_current()`.

Note
end_page() calls `Fl_Surface_Device::pop_current()`. If any calls between `begin_page()` and `end_page()` open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to `Fl_Surface_Device::pop_current()` and a call to `Fl_Surface_Device::push_current()`.

Returns
0 if OK, non-zero if any error.
Reimplemented from `Fl_Paged_Device`.

### 32.98.2.4 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
- **out left** If non-null, `*left` is set to the left margin size.
- **out top** If non-null, `*top` is set to the top margin size.
- **out right** If non-null, `*right` is set to the right margin size.
- **out bottom** If non-null, `*bottom` is set to the bottom margin size.

Reimplemented from `Fl_Paged_Device`.

### 32.98.2.5 origin() [1/2]

```cpp
void Fl_Printer::origin ( int * x, int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters
- **out x,y** If non-null, `*x` and `*y` are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from `Fl_Widget_Surface`.
### 32.98.2.6 origin()

```c
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**

- **x**, **y**
  Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

### 32.98.2.7 printable_rect()

```c
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.

### 32.98.2.8 rotate()

```c
void Fl_Printer::rotate ( float angle ) [virtual]
```

Rotates the graphics operations relatively to paper. The rotation is centered on the current graphics origin. Successive `rotate()` calls don't combine their effects.

**Parameters**

- **angle**
  Rotation angle in counter-clockwise degrees.

Reimplemented from Fl_Paged_Device.

### 32.98.2.9 scale()

```c
void Fl_Printer::scale ( float scale_x, float scale_y = 0. ) [virtual]
```

Changes the scaling of page coordinates. This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don't combine their effects.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scale_x</code></td>
<td>Horizontal dimensions of plot are multiplied by this quantity.</td>
</tr>
<tr>
<td><code>scale_y</code></td>
<td>Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code>. Thus, <code>scale(factor)</code> is equivalent to <code>scale(factor, factor)</code>;</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Paged_Device`.

### 32.98.2.10 `set_current()`

```cpp
void Fl_Printer::set_current ( )
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current()` / `Fl_Surface_Device::pop_current()`.

Note

- It's recommended to use this function only as follows:
  - The current drawing surface is the display;
  - make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
  - draw to that surface;
  - make the display current again with `Fl_Display_Device::display_device()->set_current()`. Don’t do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current()` / `Fl_Surface_Device::pop_current()`.

Reimplemented from `Fl_Surface_Device`.

### 32.98.2.11 `translate()`

```cpp
void Fl_Printer::translate ( int x, int y )
```

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`

### 32.99 `Fl_Progress Class Reference`

Displays a progress bar for the user.

```cpp
#include <Fl_Progress.H>
```

Inheritance diagram for `Fl_Progress`:

```
Fl_Widget
    |
    ⬇
Fl_Progress
```
Public Member Functions

- **FL_Progress** (int x, int y, int w, int h, const char *l=0)
  
  The constructor creates the progress bar using the position, size, and label.

- **float** maximum () const
  
  Gets the maximum value in the progress widget.

- **void** maximum (float v)
  
  Sets the maximum value in the progress widget.

- **float** minimum () const
  
  Gets the minimum value in the progress widget.

- **void** minimum (float v)
  
  Sets the minimum value in the progress widget.

- **float** value () const
  
  Gets the current value in the progress widget.

- **void** value (float v)
  
  Sets the current value in the progress widget.

Protected Member Functions

- virtual void **draw** ()
  
  Draws the progress bar.

Additional Inherited Members

32.99.1 Detailed Description

Displays a progress bar for the user.

32.99.2 Constructor & Destructor Documentation

32.99.2.1 **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.

32.99.3 Member Function Documentation

32.99.3.1 **maximum() [1/2]**

float FL_Progress::maximum () const [inline]

Gets the maximum value in the progress widget.
32.99.3.2 maximum() [2/2]

```cpp
void Fl_Progress::maximum ( 
    float v ) [inline]
```

Sets the maximum value in the progress widget.

32.99.3.3 minimum() [1/2]

```cpp
float Fl_Progress::minimum ( ) const [inline]
```

Gets the minimum value in the progress widget.

32.99.3.4 minimum() [2/2]

```cpp
void Fl_Progress::minimum ( 
    float v ) [inline]
```

Sets the minimum value in the progress widget.

32.99.3.5 value() [1/2]

```cpp
float Fl_Progress::value ( ) const [inline]
```

Gets the current value in the progress widget.

32.99.3.6 value() [2/2]

```cpp
void Fl_Progress::value ( 
    float v ) [inline]
```

Sets the current value in the progress widget.

The documentation for this class was generated from the following files:

- Fl_Progress.H
- Fl_Progress.cxx

32.100 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:

```
    Fl_Widget
     |          |
     v          v
    Fl_Button
     |          |
     v          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

32.100.1 Constructor & Destructor Documentation

32.100.1.1 Fl_Radio_Button()

Fl_Radio_Button::Fl_Radio_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗ L = 0 )

The constructor creates the button using the given position, size, and label.  
The Button type() is set to FL_RADIO_BUTTON.

Parameters

<table>
<thead>
<tr>
<th>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

32.101 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

32.102 Fl_Radio_Round_Button Class Reference

Inheritance diagram for Fl_Radio_Round_Button:
Public Member Functions

- **Fl_Radio_Round_Button** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new `Fl_Radio_Button` widget using the given position, size, and label string.

Additional Inherited Members

32.102.1 Constructor & Destructor Documentation

32.102.1.1 Fl_Radio_Round_Button()

`Fl_Radio_Round_Button::Fl_Radio_Round_Button`

```cpp
int X,
int Y,
int W,
int H,
const char *L = 0)
```

Creates a new `Fl_Radio_Button` widget using the given position, size, and label string.
The button `type()` is set to `FL_RADIO_BUTTON`.

Parameters

<table>
<thead>
<tr>
<th>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_Round_Button.H
- Fl_Round_Button.cxx

32.103 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 \((x = y = w = h = 0)\).
• **Fl_Rect** (const Fl_Widget &widget)
  
  *This constructor creates a rectangle based on a widget’s position and size.*

• **Fl_Rect** (const Fl_Widget *const widget)
  
  *This constructor creates a rectangle based on a widget’s position and size.*

• **Fl_Rect** (int W, int H)
  
  *This constructor creates a rectangle with \( x = y = 0 \) and the given width and height.*

• **Fl_Rect** (int X, int Y, int W, int H)
  
  *This constructor creates a rectangle with the given \( x,y \) coordinates and the given width and height.*

• 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)*

### 32.103.1 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 fl_measure(), fl_text_extents(), fl_rect(), fl_rectf(), and maybe more.

### 32.103.2 Member Function Documentation

#### 32.103.2.1 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.

#### 32.103.2.2 r()

```
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
32.104 Fl_Scroll::Fl_Region_LRTB Struct Reference

A local struct to manage a region defined by left/right/top/bottom.
#include <Fl_Scroll.H>

Public Attributes

- int b
  (b)ottom "y" position, aka y2
- int l
  (l)eft "x" position, aka x1
- int r
  (r)ight "x" position, aka x2
- int t
  (t)op "y" position, aka y1

32.104.1 Detailed Description

A local struct to manage a region defined by left/right/top/bottom.
The documentation for this struct was generated from the following file:

- Fl_Scroll.H

32.105 Fl_Scroll::Fl_Region_XYWH Struct Reference

A local struct to manage a region defined by xywh.
#include <Fl_Scroll.H>

Public Attributes

- int h
- int w
- int x
- int y

32.105.1 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

32.106 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 *l=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

32.106.1 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.

32.106.2 Constructor & Destructor Documentation

32.106.2.1 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.

32.106.3 Member Function Documentation

32.106.3.1 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

|       |   event   | the kind of event received |

Return values

| 0     | if the event was not used or understood |
| 1     | if the event was used and can be deleted |
See also

Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Repeat_Button.H
- Fl_Repeat_Button.hxx

### 32.107 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](image)

**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**

### 32.107.1 Detailed Description

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.

A carriage-return symbol is drawn next to the button label.

![Fl_Return_Button](image)

**Figure 32.31 Fl_Return_Button**

### 32.107.2 Constructor & Destructor Documentation
32.107.2.1 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.

32.107.3 Member Function Documentation

32.107.3.1 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.

32.107.3.2 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

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Button.

The documentation for this class was generated from the following files:

- Fl_Return_Button.H
- Fl_Return_Button.cxx
32.108  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.

#include <Fl_Image.H>

Inheritance diagram for Fl_RGB_Image:

Fl_RGB_Image
  ↓
Fl_Image
  ↓
Fl_BMP_Image Fl_JPEG_Image Fl_PNG_Image Fl_PNM_Image Fl_SVG_Image

Public Member Functions

- virtual Fl_SVG_Image * as_svg_image ()
  
  Returns whether an image is an Fl_SVG_Image or not.
- virtual void color_average (Fl_Color c, float i)
  
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- virtual Fl_Image * copy (int W, int H) const
  
  Creates a resized copy of the image.
- virtual void desaturate ()
  
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  
  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap * pxm, Fl_Color bg=FL_GRAY)
  
  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar * bits, int W, int H, int D=3, int LD=0)
  
  The constructor creates a new image from the specified data.
- virtual void label (Fl_Menu_Item * m)
  
  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget * w)
  
  This method is an obsolete way to set the image attribute of a widget or menu item.
- 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 size_t max_size ()
  
  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)
  
  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Generated by Doxygen
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

32.108.1 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.

32.108.2 Constructor & Destructor Documentation

32.108.2.1 Fl_RGB_Image()

Fl_RGB_Image::Fl_RGB_Image (  
  const uchar * bits,  
  int W,  
  int H,  
  int D = 3,  
  int LD = 0 )

The constructor creates a new image from the specified data. The data array bits must contain sufficient data to provide W * H * D image bytes and optional line padding, see LD.
W and H are the width and height of the image in pixels, resp. D is the image depth and can be:

- D=1: each uchar in bits[] is a grayscale pixel value
- D=2: each uchar pair in bits[] is a grayscale + alpha pixel value
- D=3: each uchar triplet in bits[] is an R/G/B pixel value
- D=4: each uchar quad in bits[] is an R/G/B/A pixel value

LD specifies the line data size of the array, see Fl_Image::ld(int). If LD is zero, then W * D is assumed, otherwise LD must be greater than or equal to W * D to account for (unused) extra data per line (padding).
The caller is responsible that the image data array bits persists as long as the image is used. This constructor sets Fl_RGB_Image::alloc_array to 0. To have the image object control the deallocation of the data array bits, set alloc_array to non-zero after construction.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bits</th>
<th>The image data array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W</td>
<td>The width of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>H</td>
<td>The height of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>The image depth, or 'number of channels' (default=3).</td>
</tr>
<tr>
<td>in</td>
<td>LD</td>
<td>Line data size (default=0).</td>
</tr>
</tbody>
</table>
32.108.2.2 Fl_RGB_Image() [2/2]

Fl_RGB_Image::Fl_RGB_Image (  
    const Fl_Pixmap * pxm,  
    Fl_Color bg = FL_GRAY  
)

The constructor creates a new RGBA image from the specified Fl_Pixmap.
The RGBA image is built fully opaque except for the transparent area of the pixmap that is assigned the bg color with full transparency.
This constructor creates a new internal data array and sets Fl_RGB_Image::alloc_array to 1 so the data array is deleted when the image is destroyed.

32.108.3 Member Function Documentation

32.108.3.1 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.

32.108.3.2 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 provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.
Reimplemented in Fl_SVG_Image.

32.108.3.3 copy()

Fl_Image * Fl_RGB_Image::copy (  
    int W,  
    int H ) const [virtual]

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- \( w() == \text{data} \_w() == W \)
- \( h() == \text{data} \_h() == H \)
Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of `FL_RGB_Image`, `FL_SVG_Image`, `FL_Pixmap`, `FL_Bitmap`, `FL_Tiled_Image`, and `FL_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `FL_GIF_Image` is an object of class `FL_Pixmap`.

**Parameters**

| in | W,H | Requested width and height of the new image |

**Note**

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from `FL_Image` or any subclass your overridden methods of `'FL_Image::copy() const'` and `'FL_Image::copy(int, int) const'` must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from `FL_Image`.
Reimplemented in `FL_SVG_Image`.

### 32.108.3.4 desaturate()

```cpp
void Fl_RGB_Image::desaturate ( ) [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from `FL_Image`.
Reimplemented in `FL_SVG_Image`.

### 32.108.3.5 draw()

```cpp
void Fl_RGB_Image::draw ( int X, int Y, int W, int H, int cx = 0, int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments `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`.
Reimplemented in `FL_SVG_Image`.

### 32.108.3.6 label() [1/2]

```cpp
void Fl_RGB_Image::label ( Fl_Menu_Item ∗ m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

**Deprecated** Please use `Fl_Menu_Item::image()` instead.

Reimplemented from `FL_Image`.

---

Generated by Doxygen
32.108.3.7 label() [2/2]

```c
void Fl_RGB_Image::label ( 
  Fl_Widget * widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

**Deprecated** Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

32.108.3.8 max_size() [1/2]

```c
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

```c
void Fl_RGB_Image::max_size(size_t)
```

32.108.3.9 max_size() [2/2]

```c
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 \( w() \times h() \times d() \). 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.

32.108.3.10 normalize()

```c
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.

32.108.3.11 uncache()

```c
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.

32.108.4 Member Data Documentation

32.108.4.1 array

```c
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
32.109 Fl_Roller Class Reference

The Fl_Roller widget is a "dolly" control commonly used to move 3D objects.

#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

32.109.1 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.

![Fl_Roller](image.png)

Figure 32.32 Fl_Roller

32.109.2 Constructor & Destructor Documentation

32.109.2.1 Fl_Roller()

Fl_Roller::Fl_Roller (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * L = 0 )

Generated by Doxygen
32.109.3 Member Function Documentation

32.109.3.1 draw()

```cpp
void Fl_Roller::draw ( ) [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

32.109.3.2 handle()

```cpp
int Fl_Roller::handle ( int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

- `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

32.110 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:
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

32.110.1 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()**.

![Figure 32.33 Fl_Round_Button](image)

The **Fl_Round_Button** subclass displays the "on" state by turning on a light, rather than drawing pushed in. The shape of the "light" is initially set to FL_ROUND_DOWN_BOX. The color of the light when on is controlled with **selection_color()**, which defaults to FL_FOREGROUND_COLOR.

32.110.2 Constructor & Destructor Documentation

32.110.2.1 **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.

![Figure 32.34 Fl_Round_Button](image)

The **Fl_Round_Button** subclass displays the "ON" state by turning on a light, rather than drawing pushed in. The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark.
The shape of the "light" is set with **down_box()** and its default value is FL_ROUND_DOWN_BOX.
The color of the light when on is controlled with **selection_color()**, which defaults to FL_FOREGROUND_COLOR (usually black).
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X, Y, W, H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Round_Button.H
- Fl_Round_Button.cxx

### 32.111 Fl_Round_Clock Class Reference

A clock widget of type FL_ROUND_CLOCK.

```c
#include <Fl_Round_Clock.H>
```

Inheritance diagram for Fl_Round_Clock:

```
Fl_Widget
    ^
   /|
  Fl_Clock_Output
    |
  Fl_Clock
    |
Fl_Round_Clock
```

### Public Member Functions

- `Fl_Round_Clock (int X, int Y, int W, int H, const char *L = 0)`
  
  Creates the clock widget, setting his type and box.

### Additional Inherited Members

### 32.111.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.

Has no box.

### 32.111.2 Constructor & Destructor Documentation

#### 32.111.2.1 Fl_Round_Clock()

```c
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 box type is FL_NO_BOX.

This constructor is the same as `Fl_Clock(FL_ROUND_CLOCK, X, Y, W, H, L)`.
See also

\[ \text{Fl}\_\text{Clock(uchar, int, int, int, int, const char *)} \]

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Round_Clock.H
- Fl_Clock.cxx

### 32.112 **Fl_Scroll** Class Reference

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

```c
#include <Fl_Scroll.H>
```

Inheritance diagram for Fl_Scroll:

```
Fl_Widget
   |   |
   Fl_Group
   |   |
   Fl_Scroll
```

#### Classes

- struct **Fl_Region_LRTB**
  
  A local struct to manage a region defined by left/right/top/bottom.

- struct **Fl_Region_XYWH**
  
  A local struct to manage a region defined by xywh.

- struct **Fl_Scrollbar_Data**
  
  A local struct to manage a scrollbar's xywh region and tab values.

- struct **ScrollInfo**
  
  Structure to manage scrollbar and widget interior sizes.

#### Public Types

- enum

  \[
  \text{HORIZONTAL} = 1, \text{VERTICAL} = 2, \text{BOTH} = 3, \text{ALWAYS\_ON} = 4, \\
  \text{HORIZONTAL\_ALWAYS} = 5, \text{VERTICAL\_ALWAYS} = 6, \text{BOTH\_ALWAYS} = 7
  \]

#### Public Member Functions

- void **clear** ()
  
  Clear all but the scrollbars...

- virtual int **delete_child** (int n)
  
  Removes the widget at index from the group and deletes it.

- **Fl_Scroll** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Scroll widget using the given position, size, and label string.

- int **handle** (int)
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 &) const
  
  Returns the bounding box for the interior of the scrolling area, inside the scrollbars.

- void draw ()
  
  Draws the widget.

- void fix_scrollbar_order ()
  
  Insure the scrollbars are the last children.

- void recalc_scrollbars (ScrollInfo &si) const
  
  Calculate visibility/size/position of scrollbars, find children's bounding box.

Additional Inherited Members

32.112.1 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 32.35 Fl_Scroll](image)

If all of the child widgets are packed together into a solid rectangle then you want to set box() to FL_NO_BOX or one of the _FRAME types. This will result in the best output. However, if the child widgets are a sparse arrangement you must set box() to a real _BOX type. This can result in some blinking during redrawing, but that can be solved by using a Fl_Double_Window.

The Fl_Scroll widget calculates the bounding box of all its children by using their widget positions and sizes (x, y, w, h). Outside labels are not considered. If you need outside labels of any widgets or free space outside of this bounding box you can add a tiny invisible Fl_Box at the relevant corner(s) of the Fl_Scroll widget, for instance:

Generated by Doxygen
By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll. Use `Fl_Scroll::type()` to change this as follows:

- 0 - No scrollbars
- Fl_Scroll::HORIZONTAL - Only a horizontal scrollbar.
- Fl_Scroll::VERTICAL - Only a vertical scrollbar.
- Fl_Scroll::BOTH - The default is both scrollbars.
- Fl_Scroll::HORIZONTAL_ALWAYS - Horizontal scrollbar always on, vertical always off.
- Fl_Scroll::VERTICAL_ALWAYS - Vertical scrollbar always on, horizontal always off.
- Fl_Scroll::BOTH_ALWAYS - Both always on.

Use `scrollbar.align(int)` (see void `Fl_Widget::align(Fl_Align)`) to change what side the scrollbars are drawn on.

If the FL_ALIGN_LEFT bit is on, the vertical scrollbar is on the left. If the FL_ALIGN_TOP bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.

This widget can also be used to pan around a single child widget "canvas". This child widget should be of your own class, with a `draw()` method that draws the contents. The scrolling is done by changing the `x()` and `y()` of the widget, so this child must use the `x()` and `y()` to position its drawing. To speed up drawing it should test `fl_not_clipped(int x,int y,int w,int h)` to find out if a particular area of the widget must be drawn.

Another very useful child is a single `Fl_Pack`, which is itself a group that packs its children together and changes size to surround them. Filling the `Fl_Pack` with `Fl_Tabs` groups (and then putting normal widgets inside those) gives you a very powerful scrolling list of individually-openable panels.

Fluid lets you create these, but you can only lay out objects that fit inside the `Fl_Scroll` without scrolling. Be sure to leave space for the scrollbars, as Fluid won’t show these either.

You cannot use `Fl_Window` as a child of this since the clipping is not conveyed to it when drawn, and it will draw over the scrollbars and neighboring objects.

### 32.112.2 Constructor & Destructor Documentation

#### 32.112.2.1 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.

### 32.112.3 Member Function Documentation
32.112.3.1  bbox()

```c
void Fl_Scroll::bbox (  
            int & X,
            int & Y,
            int & W,
            int & H ) const  [protected]
```

Returns the bounding box for the interior of the scrolling area, inside the scrollbars. This method does not change the scrollbars or their visibility. First the scrollbar positions and visibility are calculated as they should be, according to the positions and sizes of the children. Then the bounding box is calculated. You may need to call `redraw()` to make sure the widget gets updated.

See also

`re.calc_scrollbars()`

32.112.3.2  delete_child()

```c
int Fl_Scroll::delete_child (  
            int index  )  [virtual]
```

Removes the widget at `index` from the group and deletes it. This method does nothing if `index` is out of bounds or if `Fl_Group::child(index)` is one of the scrollbars.

Parameters

| in | index | index of child to be removed |

Returns

success (0) or error code

Return values

| 0  | success          |
| 1  | index out of range |
| 2  | widget not allowed to be removed (see note) |

See also

`Fl_Group::delete_child()`

Since

FLTK 1.4.0

Reimplemented from `Fl_Group`.

32.112.3.3  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:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Group.

32.112.3.4 handle()

int Fl_Scroll::handle (int event) [virtual]
Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

Fl_Event
Reimplemented from Fl_Group.

32.112.3.5 recalc_scrollbars()

void Fl_Scroll::recalc_scrollbars (ScrollInfo & si) const [protected]
Calculate visibility/size/position of scrollbars, find children's bounding box.
The si parameter will be filled with data from the calculations. Derived classes can make use of this call to figure out the scrolling area eg. during resize() handling.
This method does not change the scrollbars or their visibility. It calculates the scrollbar positions and visibility as they should be, according to the positions and sizes of the children.
You may need to call redraw() to make sure the widget gets updated.

Parameters

| in,out | si  | – ScrollInfo structure, filled with data |

See also

bbox()

32.112.3.6 resize()

void Fl_Scroll::resize (int X, int Y, int W, int H) [virtual]
```
Resizes the Fl_Scroll widget and moves its children if necessary. The Fl_Scroll widget first resizes itself, and then it moves all its children if (and only if) the Fl_Scroll widget has been moved. The children are moved by the same amount as the Fl_Scroll widget has been moved, hence all children keep their relative positions.

Note

Fl_Scroll::resize() does not call Fl_Group::resize(), and child widgets are not resized.

Since children of an Fl_Scroll are not resized, the resizable() widget is ignored (if it is set). The scrollbars are moved to their proper positions, as given by Fl_Scroll::scrollbar.align(), and switched on or off as necessary.

Note

Due to current (FLTK 1.3.x) implementation constraints some of this may effectively be postponed until the Fl_Scroll is drawn the next time. This may change in a future release.

See also

Fl_Group::resizable()
Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Group.

32.112.3.7 scroll_to()

void Fl_Scroll::scroll_to (  
  int X,  
  int Y )

Moves the contents of the scroll group to a new position. This is like moving the scrollbars of the Fl_Scroll around. For instance:

Fl_Scroll scroll (10,10,200,200);
Fl_Box b1 ( 10, 10,50,50,"b1"); // relative (x,y) = (0,0)
Fl_Box b2 ( 60, 60,50,50,"b2"); // relative (x,y) = (50,50)
Fl_Box b3 ( 60,110,50,50,"b3"); // relative (x,y) = (50,100)
// populate scroll with more children ...
scroll.end();
scroll.scroll_to(50,100);
will move the logical origin of the internal scroll area to (-50,-100) relative to the origin of the Fl_Scroll (10,10), i.e. Fl_Box b3 will be visible in the top left corner of the scroll area.

32.112.3.8 scrollbar_size() [1/2]

int Fl_Scroll::scrollbar_size ( ) const [inline]

Gets the current size of the scrollbars' troughs, in pixels. If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

32.112.3.9 scrollbar_size() [2/2]

void Fl_Scroll::scrollbar_size (  
  int newSize ) [inline]

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets’ scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.

Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newSize</td>
<td>Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

See also

Fl::scrollbar_size()

32.112.3.10 xposition()

int Fl_Scroll::xposition ( ) const [inline]

Gets the current horizontal scrolling position.

32.112.3.11 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

32.113 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

32.113.1 Detailed Description

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar. Clicking on the arrows move up/left and down/right by linesize(). Scrollbars also accept FL_SHORTCUT events: the arrows move by linesize(), and vertical scrollbars take Page Up/Down (they move by the page size minus linesize()) and Home/End (they jump to the top or bottom).

Scrollbars have step(1) preset (they always return integers). If desired you can set the step() to non-integer values. You will then have to use casts to get at the floating-point versions of value() from Fl_Slider.

![Figure 32.36 Fl_Scrollbar](image)

32.113.2 Constructor & Destructor Documentation

32.113.2.1 Fl_Scrollbar()

Fl_Scrollbar::Fl_Scrollbar ( int X, int Y, int W, int H, const char * L = 0 )
Creates a new Fl_Scrollbar widget with given position, size, and label. You need to do type(FL_HORIZONTAL) if you want a horizontal scrollbar.
32.113.3 Member Function Documentation

32.113.3.1 draw()

```cpp
void Fl_Scrollbar::draw ( ) [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:
```cpp
FL_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();              // calls Fl_Scrollbar::draw()
```
Implements `Fl_Widget`.

32.113.3.2 handle()

```cpp
int Fl_Scrollbar::handle ( int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th>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`.

32.113.3.3 linesize()

```cpp
void Fl_Scrollbar::linesize ( int i ) [inline]
```

This number controls how big the steps are that the arrow keys do.
In addition page up/down move by the size last sent to `value()` minus one `linesize()`. The default is 16.

32.113.3.4 value() [1/3]

```cpp
int Fl_Scrollbar::value ( ) const [inline]
```

Gets the integer value (position) of the slider in the scrollbar.
You can get the floating point value with `Fl_Slider::value()`.
32.113.3.5  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)

32.113.3.6  value() [3/3]

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>position, first line displayed</td>
</tr>
<tr>
<td>windowSize</td>
<td>number of lines displayed</td>
</tr>
<tr>
<td>first_line</td>
<td>number of first line</td>
</tr>
<tr>
<td>total_lines</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

You should call this every time your window changes size, your data changes size, or your scroll position changes (even if in response to a callback from this scrollbar). All necessary calls to redraw() are done. Calls Fl_Slider::scrollvalue(int pos, int size, int first, int total).

The documentation for this class was generated from the following files:

   • Fl_Scrollbar.H
   • Fl_Scrollbar.cxx

32.114  Fl_Scroll::Fl_Scrollbar_Data Struct Reference

A local struct to manage a scrollbar's xywh region and tab values.
#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

32.114.1 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

32.115 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

32.115.1 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.

32.115.2 Constructor & Destructor Documentation

32.115.2.1 Fl_Secret_Input()

Fl_Secret_Input::Fl_Secret_Input ( int X,
  int Y,
  int W,
  int H,
  const char * l = 0 )
32.115.3 Member Function Documentation

32.115.3.1 handle()

```cpp
int Fl_Secret_Input::handle (int event) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

- **in event** the kind of event received

Return values

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted

See also

- Fl_Event

Reimplemented from Fl_Input.
The documentation for this class was generated from the following files:

- Fl_Secret_Input.H
- Fl_Input.cxx

32.116 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.

```cpp
#include <Fl_Select_Browser.H>
```

Inheritance diagram for Fl_Select_Browser:
Public Member Functions

- **Fl_Select_Browser** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new **Fl_Select_Browser** widget using the given position, size, and label string.

Additional Inherited Members

32.116.1 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.

32.116.2 Constructor & Destructor Documentation

32.116.2.1 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**

32.117 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_Shared_Image"
      "Fl_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 provided FLTK color value.

- **Fl_Image** * copy () const

- virtual **Fl_Image** * copy (int W, int H) const
  
  Creates a resized copy of the image.

- virtual void **desaturate** ()
The desaturate() method converts an image to grayscale.

- void draw (int X, int Y)
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  Draws the image to the current drawing surface with a bounding box.

- const Fl_Image * image () const
  Returns a pointer to the internal Fl_Image object.

- const char * name ()
  Returns the filename of the shared image.

- int original ()
  Returns whether this is an original image.

- int refcount ()
  Returns the number of references of this shared image.

- void release ()
  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 image 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)
  Finds 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

32.117.1 Detailed Description

This class supports caching, loading, and drawing of image files. Most applications will also want to link against the fltk_images library and call the fl_register_images() function to support standard image formats such as BMP, GIF, JPEG, PNG, and SVG (unless the library was built with the option removing SVG support).

Images can be requested (loaded) with Fl_Shared_Image::get(), find(), and some other methods. All images are cached in an internal list of shared images and should be released when they are no longer needed. A refcount is used to determine if a released image is to be destroyed with delete.

See also

- fl_register_image()
- Fl_Shared_Image::get()
- Fl_Shared_Image::find()
- Fl_Shared_Image::release()

32.117.2 Constructor & Destructor Documentation

32.117.2.1 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.
32.117.2.2 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.

32.117.2.3 ~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.

32.117.3 Member Function Documentation

32.117.3.1 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().

32.117.3.2 add_handler()

void Fl_Shared_Image::add_handler ( Fl_Shared_Handler f ) [static]

Adds a shared image handler, which is basically a test function for adding new image formats.
This function will be called when an Fl_Shared_Image is to be loaded (for instance with Fl_Shared_Image::get()) and the image type is not known to FLTK.
All registered image handlers will be called in the order of registration. You should always call fl_register_images() before adding your own handlers - unless you need to override a known image file type which should be rare.
See also

    Fl_Shared_Handler for more information of the function you need to define.

32.117.3.3 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.
32.117.3.4 color_average()

```c
void Fl_Shared_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]
```

The `color_average()` method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from `Fl_Image`.

32.117.3.5 compare()

```c
int Fl_Shared_Image::compare (  
    Fl_Shared_Image ** i0,  
    Fl_Shared_Image ** i1 ) [static], [protected]
```

Compares two shared images.
The order of comparison is:

1. Image name, usually the filename used to load it
2. Image width
3. Image height

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>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the images match</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Image i0 is less than image i1</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>Image i0 is greater than image i1</td>
</tr>
</tbody>
</table>

32.117.3.6 copy()

```c
Fl_Image * Fl_Shared_Image::copy (  
    int W,  
    int H ) const [virtual]
```
Creates a resized copy of the image. The new image should be released when you are done with it.

Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().

For the new image the following equations are true:

- \( w() == \text{data}_w() == W \)
- \( h() == \text{data}_h() == H \)

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Im, Fl_SVG_Im, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Im, and Fl_Shared_Im. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Im is an object of class Fl_Pixmap.

Parameters

| in  | \( W,H \) | Requested width and height of the new image |

Note

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

### 32.117.3.7 desaturate()

void Fl_Shared_Im::desaturate () [virtual]

The desaturate() method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from Fl_Image.

### 32.117.3.8 draw()

void Fl_Shared_Im::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.
32.117.3.9 find()

```
Fl_Shared_Image * Fl_Shared_Image::find (  
    const char * name,  
    int W = 0,  
    int H = 0 ) [static]
```

Finds a shared image from its name and size specifications. This uses a binary search in the image cache. If the image name exists with the exact width \( W \) and height \( H \), then it is returned. If \( W = 0 \) and the image name exists with another size, then the original image with that name is returned. In either case the refcount of the returned image is increased. The found image should be released with Fl_Shared_Image::release() when no longer needed.

32.117.3.10 get() [1/2]

```
Fl_Shared_Image * Fl_Shared_Image::get (  
    const char * name,  
    int W = 0,  
    int H = 0 ) [static]
```

Find or load an image that can be shared by multiple widgets. If the image exists with the requested size, this image will be returned. If the image exists, but only with another size, then a new copy with the requested size (width \( W \) and height \( H \)) will be created as a resized copy of the original image. The new image is added to the internal list of shared images. If the image does not yet exist, then a new image of the proper dimension is created from the filename name. The original image from filename name is always added to the list of shared images in its original size. If the requested size differs, then the resized copy with width \( W \) and height \( H \) is also added to the list of shared images.

Note

- If the sizes differ, then two images are created as mentioned above. This is intentional so the original image is cached and preserved. If you request the same image with another size later, then the original image will be found, copied, resized, and returned.

Shared JPEG and PNG images can also be created from memory by using their named memory access constructor. You should release() the image when you're done with it.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>name of the image</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W,H )</td>
<td>desired size</td>
</tr>
</tbody>
</table>

See also

- Fl_Shared_Image::find(const char *name, int W, int H)
- Fl_Shared_Image::release()
- Fl_JPEG_Image::Fl_JPEG_Image(const char *name, const unsigned char *data)
- Fl_PNG_Image::Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int maxsize)

32.117.3.11 get() [2/2]

```
Fl_Shared_Image * Fl_Shared_Image::get (  
    Fl_RGB_Image * rgb,  
    int own_it = 1 ) [static]
```

Builds a shared image from a pre-existing Fl_RGB_Image.

Parameters

| in | rgb | an Fl_RGB_Image used to build a new shared image. |

Generated by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>own</code></td>
<td>1 if the shared image should delete <code>rgb</code> when it is itself deleted, 0 otherwise</td>
</tr>
</tbody>
</table>

### Version

1.3.4

### 32.117.3.12 image()

```cpp
const Fl_Image* Fl_Shared_Image::image ( ) const [inline]
```

Returns a pointer to the internal `Fl_Image` object.
The output is a pointer to the internal image (`Fl_Image` or subclass) which can be used to inspect or copy the image.

**Do not try to modify the image!** You can copy the image though if you want or need to change any attributes, size etc. If all you need to do is to resize the image you should use `Fl_Shared_Image::copy(int, int)` instead.

**Note**
The internal image (pointer) is protected for good reasons, e.g. to prevent access to the image so it can't be modified by user code. **DO NOT** cast away the 'const' attribute to modify the image.

User code should rarely need this method. Use with caution.

**Returns**

- `const Fl_Image* image`, the internal `Fl_Image`

Since

1.4.0

### 32.117.3.13 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

### 32.117.3.14 refcount()

```cpp
int Fl_Shared_Image::refcount ( ) [inline]
```

Returns the number of references of this shared image.
When reference is below 1, the image is deleted.
### 32.117.3.15 release()

```cpp
void Fl_Shared_Image::release ( ) [virtual]
```

Releases and possibly destroys (if refcount <= 0) a shared image. In the latter case, it will reorganize the shared image array so that no hole will occur. Reimplemented from `Fl_Image`.

### 32.117.3.16 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`

### 32.118 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          v
Fl_Valuator
    |          |
    v          v
Fl_Counter
    |          |
    v          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

### 32.118.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

![Figure 32.37 Fl_Simple_Counter](image)

The documentation for this class was generated from the following files:

- `Fl_Simple_Counter.H`
- `Fl_Counter.cxx`
This is a continuous text scroll widget for logging and debugging output, much like a terminal.

#include <Fl_Simple_Terminal.H>

Inheritance diagram for Fl_Simple_Terminal:

```
Fl_Widget
   Fl_Group
   Fl_Text_Display
   Fl_Simple_Terminal
```

Public Member Functions

- bool ansi () const
  Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.

- 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.

- void append (const char *s, int len=-1)
  Appends new string 's' to terminal.

- void clear ()
  Clears the terminal's screen and history.

- int current_style_index () const
  Get the style table index used as the current drawing color/font/weight/size for new text.

- void current_style_index (int)
  Set 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.

- int history_lines () const
  Get the maximum number of terminal history lines last set by history_lines(int).

- void history_lines (int)
  Sets the maximum number of lines for the terminal history.

- int normal_style_index () const
  Gets the style table index used by the ANSI terminal reset sequence "\033[0m".

- 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".

- 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.

- bool stay_at_bottom () const
  Gets the current value of the stay_at_bottom(bool) flag.

- 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.

- const Fl_Text_Display::Style_Table_Entry * style_table () const

Generated by Doxygen
Return the current style table being used.

- 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.

- int **style_table_size** () const
  
  Return the current style table's size (in bytes).

- const char **∗** **text** () const
  
  Returns entire text content of the terminal as a single string.

- void **text**(const char **∗** s, int len=-1)
  
  Replaces the terminal with new text content in string 's'.

- void **vprintf**(const char **∗** fmt, va_list ap)
  
  Appends printf formatted messages to the terminal.

- ~Fl_Simple_Termin**al** ()
  
  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

#### 32.119.1 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));
  Fl::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[32m"
  G_win->end();
  G_win->resizable(G_win);
  G_win->show();
  Fl::add_timeout(0.5, tick_cb);
  return Fl::run();
} //END
```

32.119.1.1 Style Tables For Color/Font/Fontsize Control

Internally this widget derives from Fl_Text_Display, and therefore inherits some of its idiosyncrasies. 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[3mThis text uses the 1st entry in the style table\n\033[1mThis text uses the 2nd entry in the style table\n\033[2mThis text uses the 3rd entry in the style table\n```

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.

32.119.2 Member Function Documentation

Generated by Doxygen
32.119.2.1 ansi() [1/2]

```cpp
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)
```

32.119.2.2 ansi() [2/2]

```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() / textsize() 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:

<table>
<thead>
<tr>
<th>ANSI Sequence</th>
<th>Color Name</th>
<th>Font Name</th>
<th>Font Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;\033[0m&quot;</td>
<td>&quot;Normal&quot;</td>
<td>FL_COURIER, 14</td>
<td>Resets to default color/font/weight/size</td>
</tr>
<tr>
<td>&quot;\033[30m&quot;</td>
<td>Medium Black</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[31m&quot;</td>
<td>Medium Red</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[32m&quot;</td>
<td>Medium Green</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[33m&quot;</td>
<td>Medium Yellow</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[34m&quot;</td>
<td>Medium Blue</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[35m&quot;</td>
<td>Medium Magenta</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[36m&quot;</td>
<td>Medium Cyan</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[37m&quot;</td>
<td>Medium White</td>
<td>FL_COURIER, 14</td>
<td>The color when &quot;\033[0m&quot; reset is used</td>
</tr>
<tr>
<td>&quot;\033[40m&quot;</td>
<td>Bright Black</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[41m&quot;</td>
<td>Bright Red</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[42m&quot;</td>
<td>Bright Green</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[43m&quot;</td>
<td>Bright Yellow</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[44m&quot;</td>
<td>Bright Blue</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[45m&quot;</td>
<td>Bright Magenta</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[46m&quot;</td>
<td>Bright Cyan</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>&quot;\033[47m&quot;</td>
<td>Bright White</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
</tbody>
</table>

Here's example code demonstrating the use of ANSI codes to select the built-in colors, and how it looks in the terminal:

![Figure 32.38 Fl_Simple_Terminal built-in ANSI sequences](image)

**Figure 32.38** 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()
32.119.2.3 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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s</code></td>
<td>string to append.</td>
</tr>
<tr>
<td><code>len</code></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()`

32.119.2.4 clear()

```cpp
void Fl_Simple_Terminal::clear ( )
```

Clears the terminal's screen and history. Cursor moves to top of window.

32.119.2.5 current_style_index() [1/2]

```cpp
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)`

32.119.2.6 current_style_index() [2/2]

```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[4m"  
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'.

32.119.2.7 draw()

```cpp
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.

32.119.2.8  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.

32.119.2.9  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.

32.119.2.10  history_lines() [1/2]

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)

32.119.2.11  history_lines() [2/2]

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. |

32.119.2.12  normal_style_index() [1/2]

int Fl_Simple_Terminal::normal_style_index ( ) const
Gets the style table index used by the ANSI terminal reset sequence "033[0m".
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)
32.119.2.13 normal_style_index() [2/2]

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).

32.119.2.14 printf()

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:
#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.
The 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. |

32.119.2.15 remove_lines()

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 |
32.119.2.16 stay_at_bottom() [1/2]

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)

32.119.2.17 stay_at_bottom() [2/2]

void Fl_Simple_Terminal::stay_at_bottom ( bool val )

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'.

32.119.2.18 style_table() [1/2]

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)

32.119.2.19 style_table() [2/2]

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):

```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, sizeof(mystyle), 0); // use our custom style table

// Now write to terminal, with ANSI that uses our style table
tty->printf("\033[0mNormal Text\033[1mRed Courier Text
\033[2mWhite Helvetica\033[0mBack to normal.");

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>stable</th>
<th>- 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)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>stable_size</td>
<td>- the sizeof() the style table (in bytes). Set this to 0 if <code>stable</code> is NULL.</td>
</tr>
<tr>
<td>normal_style_index</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>

32.119.2.20 style_table_size()

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).

32.119.2.21 text() [1/2]

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.

32.119.2.22 text() [2/2]

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()

32.119.2.23 vprintf()

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>fmt</th>
<th>is a printf format string for the message text.</th>
</tr>
</thead>
<tbody>
<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

32.120 Fl_Single_Window Class Reference

This is the same as Fl_Window.

#include <Fl_Single_Window.H>

Inheritance diagram for Fl_Single_Window:

```
+---+                  +---+      +---+      +---+      +---+
| Fl_Widget |               | Fl_Group |               | Fl_Window |               | Fl_Menu_Window |
    |         |               |           |            |            |            |              |
    | Fl_Single_Window |         |            |            |            |            |              |
    |                   | Fl_Window |            |            |            |            |              |
    |                   |           |            |            |            |            |              |
    +---+               +---+      +---+      +---+      +---+
     |                    | Fl_Group |          | Fl_Window | Fl_Menu_Window |
     +---+               +---+      +---+      +---+      +---+
```

Public Member Functions

- **Fl_Single_Window** (int W, int H, const char *l=0)
  Creates a new Fl_Single_Window widget using the given size, and label (title) string.

- **Fl_Single_Window** (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.

- **void show ()**
  Puts the window on the screen.

- **void show (int a, char **b)**
  Same as Fl_Window::show(int a, char **b)
Additional Inherited Members

32.120.1 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.

32.120.2 Member Function Documentation

32.120.2.1 show()

    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

32.121 Fl_Slider Class Reference

The Fl_Slider widget contains a sliding knob inside a box.
#include <Fl_Slider.H>
Inheritance diagram for Fl_Slider:
Public Member Functions

- void bounds (double a, double b)
  Sets the minimum (a) and maximum (b) values for the valuator widget.
- FL_Slider (int X, int Y, int W, int H, const char *L=0)
  Creates a new FL_Slider widget using the given position, size, and label string.
- FL_Slider (uchar t, int X, int Y, int W, int H, const char *L)
  Creates a new FL_Slider widget using the given type, position, size, and label string.
- int handle (int)
  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 ()
  Draws the widget.
- void draw (int, int, int, int)
- int handle (int, int, int, int, int)

Additional Inherited Members

32.121.1 Detailed Description

The FL_Slider widget contains a sliding knob inside a box. It is often used as a scrollbar. Moving the box all the way to the top/left sets it to the minimum(), and to the bottom/right to the maximum(). The minimum() may be greater than the maximum() to reverse the slider direction. Use void Fl_Widget::type(int) to set how the slider is drawn, which can be one of the following:

- FL_VERTICAL - Draws a vertical slider (this is the default).
- FL_HORIZONTAL - Draws a horizontal slider.
- FL_VERT_FILL_SLIDER - Draws a filled vertical slider, useful as a progress or value meter.
- FL_HOR_FILL_SLIDER - Draws a filled horizontal slider, useful as a progress or value meter.
- FL_VERT_NICE_SLIDER - Draws a vertical slider with a nice looking control knob.
- FL_HOR_NICE_SLIDER - Draws a horizontal slider with a nice looking control knob.

Figure 32.39 Fl_Slider
### 32.121.2 Constructor & Destructor Documentation

#### 32.121.2.1 Fl_Slider()

```cpp
class Fl_Slider{
    int x,
    int y,
    int w,
    int h,
    const char * l = 0
public:
    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.

### 32.121.3 Member Function Documentation

#### 32.121.3.1 bounds()

```cpp
void Fl_Slider::bounds (double a, double b)
```

Sets the minimum (a) and maximum (b) values for the valuator widget. If at least one of the values is changed, a partial redraw is asked.

#### 32.121.3.2 draw()

```cpp
void Fl_Slider::draw () [protected], [virtual]
```

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method `from within your own draw() method`, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

Reimplemented in Fl_Value_Slider.

#### 32.121.3.3 handle()

```cpp
int Fl_Slider::handle (int event) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**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

<table>
<thead>
<tr>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<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.

### 32.121.3.4 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>position of first line displayed</td>
</tr>
<tr>
<td>size</td>
<td>size of window in lines</td>
</tr>
<tr>
<td>first</td>
<td>number of first line</td>
</tr>
<tr>
<td>total</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

Returns Fl_Valuator::value(p)

### 32.121.3.5 slider_size()

```cpp
void Fl_Slider::slider_size (double v)
```

Set the dimensions of the moving piece of slider.

This is the fraction of the size of the entire widget. If you set this to 1 then the slider cannot move. The default value is .08.

For the "fill" sliders this is the size of the area around the end that causes a drag effect rather than causing the slider to jump to the mouse.

The documentation for this class was generated from the following files:

- Fl_Slider.H
- Fl_Slider.cxx

### 32.122 Fl_Spinner Class Reference

This widget is a combination of a numerical input widget and repeat buttons.

```cpp
#include <Fl_Spinner.H>
```

Inheritance diagram for Fl_Spinner:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Spinner
```

Generated by Doxygen
Classes

- class Fl_Spinner_Input

Public Member Functions

- Fl_Color color () const
  Returns the background color of the spinner widget's input field.
- void color (Fl_Color v)
  Sets the background color of the spinner widget's input field.
- Fl_Spinner (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Spinner widget using the given position, size, and label string.
- const char ∗ format () const
  Returns the format string for the value.
- void format (const char ∗f)
  Sets the format string for the value.
- int handle (int event)
  Handles the specified event.
- double maximum () const
  Gets the maximum value of the widget.
- void maximum (double m)
  Sets the maximum value of the widget.
- int maximum_size () const
  Returns the maximum width of the input field.
- void maximum_size (int m)
  Sets the maximum width of the input field.
- double minimum () const
  Gets the minimum value of the widget.
- void minimum (double m)
  Sets the minimum value of the widget.
- void range (double a, double b)
  Sets the minimum and maximum values for the widget.
- void resize (int X, int Y, int W, int H)
  Resizes the Fl_Group widget and all of its children.
- Fl_Color selection_color () const
  Returns the selection color of the spinner widget's input field.
- void selection_color (Fl_Color val)
  Sets the selection color of the spinner widget's input field.
- double step () const
  Gets the amount to change the value when the user clicks a button.
- void step (double s)
  Sets or returns the amount to change the value when the user clicks a button.
- Fl_Color textcolor () const
  Gets the color of the text in the input field.
- void textcolor (Fl_Color c)
  Sets the color of the text in the input field.
- Fl_Font textfont () const
  Gets the font of the text in the input field.
- void textfont (Fl_Font f)
  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const
  Gets the size of the text in the input field.
### Detailed Description

This widget is a combination of a numerical input widget and repeat buttons. The user can either type into the input area or use the buttons to change the value.

![Figure 32.40 Fl_Spinner widget](image)

### Constructor & Destructor Documentation

#### Fl_Spinner()

```cpp
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.

### Member Function Documentation

- **void textsize (Fl_Fontsize s)**
  
  Sets the size of the text in the input field.

- **uchar type () const**
  
  Gets the numeric representation in the input field.

- **void type (uchar v)**
  
  Sets the numeric representation in the input field.

- **double value () const**
  
  Gets the current value of the widget.

- **void value (double v)**
  
  Sets the current value of the input widget.

- **int wrap () const**
  
  Gets the wrap mode of the `Fl_Spinner` widget.

- **void wrap (int set)**
  
  Sets whether the spinner wraps around at upper and lower bounds.

### Protected Attributes

- `Fl_Repea_Button down_button_`
- `Fl_Spinner_Input input_`
- `Fl_Repea_Button up_button_`

### Additional Inherited Members
32.122.3.1 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**

- `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.

32.122.3.2 resize()

```cpp
void Fl_Spinner::resize ( int X, int Y, int W, int H ) [virtual]
```

Resizes the Fl_Group widget and all of its children. The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

**See also**

- Fl_Group::resizable(Fl_Widget*)
- Fl_Group::resizable()
- Fl_Widget::resize(int,int,int)

Reimplemented from Fl_Group.

32.122.3.3 step [1/2]

```cpp
double Fl_Spinner::step ( ) const [inline]
```

Gets the amount to change the value when the user clicks a button.

**See also**

- Fl_Spinner::step(double)
32.122.3.4 step() [2/2]

```cpp
void Fl_Spinner::step (double s)
```

Sets or returns the amount to change the value when the user clicks a button. Before setting step to a non-integer value, the spinner `type()` should be changed to floating point.

See also

```cpp
double Fl_Spinner::step () const
```

32.122.3.5 type() [1/2]

```cpp
uchar Fl_Spinner::type () const [inline]
```

Gets the numeric representation in the input field.

See also

```cpp
Fl_Spinner::type(uchar)
```

32.122.3.6 type() [2/2]

```cpp
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.

32.122.3.7 value()

```cpp
void Fl_Spinner::value (double v) [inline]
```

Sets the current value of the input widget. Before setting value to a non-integer value, the spinner `type()` should be changed to floating point.

32.122.3.8 wrap() [1/2]

```cpp
int Fl_Spinner::wrap ( ) const [inline]
```

Gets the wrap mode of the `Fl_Spinner` widget.

See also

```cpp
void wrap(int)
```

Since

1.4.0
### 32.122.3.9 `wrap()` [2/2]

```c
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**

- `set` non-zero sets wrap mode, zero resets wrap mode

**Since**

1.4.0

The documentation for this class was generated from the following files:

- `Fl_Spinner.H`
- `Fl_Spinner.cxx`

### 32.123 `Fl_Spinner::Fl_Spinner_Input` Class Reference

Inheritance diagram for `Fl_Spinner::Fl_Spinner_Input`:

- `Fl_Spinner::Fl_Spinner_Input`
- `Fl_Input`
- `Fl_Input_`
- `Fl_Widget`

#### Public Member Functions

- `Fl_Spinner_Input (int X, int Y, int W, int H)`
- `int handle (int event)`

Handles events of `Fl_Spinner`'s embedded input widget.

#### Additional Inherited Members

### 32.123.1 Member Function Documentation
32.123.1.1 handle()

```cpp
int Fl_Spinner::Fl_Spinner_Input::handle (int event) [virtual]
```

Handles events of Fl_Spinner's embedded input widget.
Works like Fl_Input::handle() but ignores FL_Up and FL_Down keys so they can be handled by the parent widget (Fl_Spinner).
Reimplemented from Fl_Input.
The documentation for this class was generated from the following files:

- Fl_Spinner.H
- Fl_Spinner.cxx

32.124 Fl_String Class Reference

Fl_String is the basic string class for FLTK.

```cpp
#include <Fl_String.H>
```

Public Member Functions

- int capacity () const
- void debug (const char ∗info) const
- Fl_String (const char ∗str)
- Fl_String (const char ∗str, int size)
- Fl_String (const Fl_String &in)
- void hexdump (const char ∗info) const
- Fl_String & operator= (const char ∗in)
- Fl_String & operator= (const Fl_String &in)
- int size () const
- int slen () const
- const char ∗ value () const
- void value (const char ∗str)
- void value (const char ∗str, int slen)

32.124.1 Detailed Description

Fl_String is the basic string class for FLTK.
In this version Fl_String can be used to store strings, copy strings, and move strings. There are no string manipulation methods yet.
Fl_String can hold the value of an Fl_Input widget including null/bytes if the constructor Fl_String(const char ∗str, int size)

is used.
Assignment and copy constructors copy the string value such that the source string can be freed immediately after
the assignment.
The string value() can be an empty string " " or NULL.
If value() is not NULL it is guaranteed that the string is terminated by a trailing null byte even if the string contains embedded null bytes.
The method size() returns the full string size, whether the string contains embedded null bytes or not. The special method slen() returns 0 if value() is NULL, otherwise the same as strlen() would do.
Examples:

```cpp
FL_String np(NULL);
printf(" np : value = \%p, size = \%d, slen = \%d\n", np.value(), np.size(), np.slen());
FL_String empty("") ;
printf(" empty : value = \%p, size = \%d\n", empty.value(), empty.size());
FL_String fltk("FLTK");
Fl_Input if(0, 0, 0, 0);
if.value("FLTK");
i.value("abc\ndef", 7);
FL_String str(i.value(), i.size());
printf(" str : strlen = \%ld, size = \%d, capacity = \%ld\n",
       str.len(), str.size(), str.capacity());
```

Output:

np : value = (nil), size = 0, slen = 0
empty : value = 0x562840befbf0, size = 0
str : strlen = 3, size = 7, capacity = 15
Since 1.4.0

Todo Complete documentation of class Fl_String

The documentation for this class was generated from the following files:

- Fl_String.H
- Fl_String.cxx

32.125 Fl_Surface_Device Class Reference

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

#include <Fl_Device.H>

Inheritance diagram for Fl_Surface_Device:

```
Fl_Surface_Device
|   
|   Fl_Display_Device Fl_Widget_Surface
|   Fl_Copy_Surface Fl_EPS_File_Surface Fl_Image_Surface Fl_Paged_Device Fl_SVG_File_Surface
|   Fl_PostScript_File_Device Fl_Printer
```

Public Member Functions

- Fl_Graphics_Driver * driver ()
  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()
  Is this surface the current drawing surface?
- virtual void set_current (void)
  Make this surface the current drawing surface.
- virtual ~Fl_Surface_Device ()
  The destructor.

Static Public Member Functions

- static Fl_Surface_Device * pop_current ()
  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)
  Pushes new_current on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()
  The current drawing surface.

Protected Member Functions

- void driver (Fl_Graphics_Driver *graphics_driver)
  Sets the graphics driver of this drawing surface.
- virtual void end_current ()
  FLTK calls this each time a surface ceases to be the current drawing surface.
- Fl_Surface_Device (Fl_Graphics_Driver *graphics_driver)
  Constructor that sets the graphics driver to use for the created surface.
### Detailed Description

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

Any FLTK application has at any time a current drawing surface to which all drawing requests are directed. The current surface is given by `FL_Surface_Device::surface()`. When `main()` begins running, the current drawing surface has been set to the computer's display, an instance of the `FL_Display_Device` class.

A drawing surface other than the computer's display, is typically used as follows:

1. Create `surface`, an object from a particular `FL_Surface_Device` derived class (e.g., `FL_Copy_Surface`, `FL_Printer`).

2. Call `FL_Surface_Device::push_current(surface)` to redirect all graphics requests to `surface` which becomes the new current drawing surface (not necessary with classes `FL_Printer` / `FL_PostScript_File_Device` because it is done by `FL_Paged_Device::begin_page()`).

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()`; (not necessary with classes `FL_Printer` / `FL_PostScript_File_Device`).

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.

### Member Function Documentation

#### end_current()

```cpp
template<class T>
virtual void Fl_Surface_Device::end_current ( ) [inline], [protected], [virtual]
```

FLTK calls this each time a surface ceases to be the current drawing surface.

This member function is mostly of interest to developers of new `FL_Surface_Device` derived classes. It allows to perform surface-specific operations necessary when this surface ceases to be current. Each implementation should end with a call to `FL_Surface_Device::end_current()`.

Reimplemented in `FL_PostScript_File_Device`.

#### pop_current()

```cpp
Fl_Surface_Device * Fl_Surface_Device::pop_current ( ) [static]
```

Removes the top element from the current drawing surface stack, and makes the new top element current.

Returns

A pointer to the new current drawing surface.

See also

`FL_Surface_Device::push_current(FL_Surface_Device *)`
32.125.2.3  push_current()

```cpp
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.

Any call to `push_current()` must be matched by a subsequent call to `Fl_Surface_Device::pop_current()`. The max height of this stack is 16.

Version

1.4.0

32.125.2.4  set_current()

```cpp
void Fl_Surface_Device::set_current (  
  void ) [virtual]
```

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

Note

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device()->set_current()`. Don't do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current()`.

Reimplemented in `Fl_Printer`, `Fl_PostScript_File_Device`, `Fl_Image_Surface`, and `Fl_Copy_Surface`.

32.125.2.5  surface()

```cpp
static Fl_Surface_Device* Fl_Surface_Device::surface ( ) [inline], [static]
```

The current drawing surface.

In other words, the `Fl_Surface_Device` object that currently receives all graphics requests.

Note

It’s possible to transiently remove the GUI scaling factor in place in the current drawing surface with `fl_override_scale()`.

The documentation for this class was generated from the following files:

- `Fl_Device.H`
- `Fl_Device.cxx`

Generated by Doxygen
32.126  Fl_SVG_File_Surface Class Reference

A drawing surface producing a Scalable Vector Graphics (SVG) file.

#include <Fl_SVG_File_Surface.H>

Inheritance diagram for Fl_SVG_File_Surface:

```
Fl_Surface_Device
   ↓
Fl_Widget_Surface
   ↓
Fl_SVG_File_Surface
```

Public Member Functions

- int close ()
  Closes the FILE pointer where SVG data is output.

- FILE * file ()
  Returns the underlying FILE pointer.

- Fl_SVG_File_Surface (int width, int height, FILE * svg, int(*closef)(FILE *))=NULL)
  Constructor of the SVG drawing surface.

- virtual void origin (int *x, int *y)
  Computes the coordinates of the current origin of graphics functions.

- virtual void origin (int x, int y)
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- 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

32.126.1 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:
```
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);
}
```
32.126 Fl_SVG_File_Surface Class Reference

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.

32.126.2 Constructor & Destructor Documentation

32.126.2.1 Fl_SVG_File_Surface()

Fl_SVG_File_Surface::Fl_SVG_File_Surface (  
  int width,  
  int height,  
  FILE * svg,  
  int(*)(FILE *) closef = NULL )

Constructor of the SVG drawing surface.

Parameters

| width, height | Width and height of the graphics area in FLTK drawing units |
| svg           | 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(). |
| closef        | If not NULL, the destructor and close() will call closef(svg) after all SVG data has been sent. If NULL, fclose(svg) is called instead. This allows to close the FILE pointer by, e.g., pclose, or, using a function such as "int keep_open(FILE*){return 0;}", to keep it open after completion of all output to svg. Function closef should return non zero to indicate an error. |

32.126.2.2 ~Fl_SVG_File_Surface()

Fl_SVG_File_Surface::~Fl_SVG_File_Surface { }

Destructor.

The underlying FILE pointer is processed as by close().

32.126.3 Member Function Documentation

32.126.3.1 close()

int Fl_SVG_File_Surface::close ( )

Closes the FILE pointer where SVG data is output.

The underlying FILE is closed by function fclose() unless another function was set at object’s construction time. The only operation possible after this on the Fl_SVG_File_Surface object is its destruction.

Returns

The value returned by the closing function call.

32.126.3.2 origin() [1/2]

virtual void Fl_SVG_File_Surface::origin (  
  int * x,  
  int * y ) [virtual]
Computes the coordinates of the current origin of graphics functions.

Parameters

<table>
<thead>
<tr>
<th>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.

### 32.126.3.3 origin() [2/2]

```cpp
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

<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.

### 32.126.3.4 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.

### 32.126.3.5 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

### 32.127 Fl_SVG_Image Class Reference

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images.
#include <Fl_SVG_Image.H>

Inheritance diagram for Fl_SVG_Image:

```
Fl_SVG_Image
|    |
|___|
|   |
|___|
|   |
|___|
|   |
|___|
Fl_RGB_Image
Fl_SVG_Image
```

Public Member Functions

- 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 provided FLTK color value.
- Fl_SVG_Image * copy () const
  Creates a resized copy of the image.
- virtual 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 desaturate ()
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
  Draws the image to the current drawing surface with a bounding box.
- Fl_SVG_Image (const char *filename, int cx=0, int cy=0)
  The constructor loads the SVG image from the given .svg/.svgz filename or in-memory data.
- virtual 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.

Additional Inherited Members

32.127.1 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). 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

Generated by Doxygen
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>

const char *svg_data = "<svg viewBox="0 0 200 200" version = "1.1">
  <rect x="25" y="50" width="150" height="100" fill="black"/>
  <transform>rotate(45 100 100)</transform>
</svg>";

int main(int argc, char **argv) { 
  Fl_SVG_Image *svg = new Fl_SVG_Image(0, svg_data); // create SVG object
  Fl_Window *win = new Fl_Window(720, 486, "svg test");
  Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
  box->image(svg); // assign svg object to Fl_Box
  win->end();
  win->show(argc,argv);
  return(Fl::run());
}
```

Example of displaying an svg image from a file:
```cpp
#include <errno.h> // errno
#include <string.h> // strerror
#include <FL/Fl.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.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:
```cpp
#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) {} 
  void resize(int x, int y, int w, int h) {
    image()->scale(w, h, 1, 1); // p3 = proportional, p4 = can_expand
    Fl_Box::resize(x, y, w, h);
  }
};

int main(int argc, char **argv) {
  Fl_SVG_Image *svg = new Fl_SVG_Image("/path/to/image.svg");
  box->image(svg);
  box->scale(svg->w(), svg->h());
  win->end();
  win->show(argc, argv);
  return(Fl::run());
}
```

32.127.2 Constructor & Destructor Documentation
32.127.2.1 Fl_SVG_Image()

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.

32.127.3 Member Function Documentation

32.127.3.1 as_svg_image()

virtual Fl_SVG_Image* Fl_SVG_Image::as_svg_image ( ) [inline], [virtual]

Returns whether an image is an Fl_SVG_Image or not. This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of Fl_SVG_Image or NULL if not. Reimplemented from Fl_RGB_Image.

32.127.3.2 color_average()

void Fl_SVG_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]

The color_average() method averages the colors in the image with the provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory. Reimplemented from Fl_RGB_Image.

32.127.3.3 copy()

Fl_Image * Fl_SVG_Image::copy (  
    int W,  
    int H ) const [virtual]

Creates a resized copy of the image. The new image should be released when you are done with it. Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().

For the new image the following equations are true:

- w() == data_w() == W
- h() == data_h() == H
Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of `Fl_RGB_Image`, `Fl_SVG_Image`, `Fl_Pixmap`, `Fl_Bitmap`, `Fl_Tiled_Image`, and `Fl_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `Fl_GIF_Image` is an object of class `Fl_Pixmap`.

Parameters

in | W,H | Requested width and height of the new image

Note

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from `Fl_Image` or any subclass your overridden methods of `Fl_Image::copy() const` and `Fl_Image::copy(int, int) const` must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from `Fl_RGB_Image`.

### 32.127.3.4 desaturate()

```cpp
void Fl_SVG_Image::desaturate ( ) [virtual]
```

The desaturate() method converts an image to grayscale. If the image contains an alpha channel (depth = 4), the alpha channel is preserved. An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from `Fl_RGB_Image`.

### 32.127.3.5 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`.

### 32.127.3.6 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`.

### 32.127.3.7 resize()

```cpp
void Fl_SVG_Image::resize ( int width, int height )
```

Generated by Doxygen
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.

32.127.4 Member Data Documentation

32.127.4.1 proportional

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

32.128 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.
#include <Fl_Sys_Menu_Bar.H>

Inheritance diagram for Fl_Sys_Menu_Bar:

```
Fl_Widget
   |
   Fl_Menu
   |
   Fl_Menu_Bar
   |
   Fl_Sys_Menu_Bar
```

Public Types

- enum window_menu_style_enum { no_window_menu = 0, tabbing_mode_none, tabbing_mode_automatic, tabbing_mode_preferred }
  Possible styles of the Window menu in the system menu bar.

Public Member Functions

- int add (const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  Adds a new menu item.
- int add (const char *label, int shortcut, Fl_Callback *, void *user_data=0, int flags=0)
  Add a new menu item to the system menu bar.
- int add (const char *str)
  Forms-compatible procedure to add items to the system menu bar.
- void clear ()
  Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)
  Clears the specified submenu pointed to by index of all menu items.
- Fl_Sys_Menu_Bar (int x, int y, int w, int h, const char *l=0)

Generated by Doxygen
The constructor.

- int insert (int index, const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  Insert a new menu item.
- int insert (int index, const char *label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)
  insert in the system menu bar a new menu item
- const Fl_Menu_Item * menu () const
  Return the system menu's array of Fl_Menu_Item's.
- void menu (const Fl_Menu_Item *m)
  create a system menu bar using the given list of menu structs
- int mode (int i) const
  Gets the flags of item i.
- void mode (int i, int fl)
  Sets the flags of item i.
- void remove (int n)
  remove an item from the system menu bar
- void replace (int index, const char *name)
  rename an item from the system menu bar
- void setonly (Fl_Menu_Item *item)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void shortcut (int i, int s)
  Changes the shortcut of item i to n.
- 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

32.128.1 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
that 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 au-
tomatically constructs and handles a Window menu which can be further customized (or even removed) calling
Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum style). Other member functions of this class
allow the app to generate the rest of the system menu bar. It is recommended to localize the system menu bar
using the standard Mac OS X localization procedure (see Internationalization).
Changes to the menu state are immediately visible in the menubar when they are made using member functions
of the Fl_Sys_Menu_Bar class. Other changes (e.g., by a call to Fl_Menu_Item::set()) should be followed by a call
to update() to be visible in the menubar across all platforms. macOS global variable fl_sys_menu_bar points to the
unique, current system menu bar.
A few FLTK menu features are not supported by the Mac System menu:

- no symbolic labels
- no embossed labels
- no font sizes

As described above, the submenu with the application’s own name (usually the second submenu from
the left, immediately following the “Apple” submenu) is a special case, and can be managed with
Fl_Mac_App_Menu::custom_application_menu_items(). For example, to make your own “Appname -> Preferences” dialog, you might use:

```c++
#include <FL/platform.H> // for Fl_Mac_App_Menu class
#include <FL/Fl_Sys_Menu_Bar.H> // for Fl_Menu_Item

void prefs_cb(Fl_Widget *w, void *data) {
    // .Open your preferences dialog here..
}

int main(..) {
    // Items to add to the application menu
    static Fl_Menu_Item appitems[] = {
        { "Preferences", 0, prefs_cb, 0, 0 },
        { 0, 0, 0, 0, 0 }
    };
    Fl_Mac_App_Menu::custom_application_menu_items(appitems); // adds it
}
```
..the result being:

![Example of a custom application menu](image)

**Figure 32.41 Mac Application submenu**

### 32.128.2 Member Enumeration Documentation

#### 32.128.2.1 window_menu_style_enum

```
enum Fl_Sys_Menu_Bar::window_menu_style_enum
```

Possible styles of the Window menu in the system menu bar.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumeral</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_window_menu</td>
<td>No Window menu in the system menu bar.</td>
</tr>
</tbody>
</table>
Enumerator

<table>
<thead>
<tr>
<th>tabbing_mode_none</th>
<th>No tabbed windows, but the system menu bar contains a Window menu.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabbing_mode_automatic</td>
<td>Windows are created by themselves but can be tabbed later.</td>
</tr>
<tr>
<td>tabbing_mode_preferred</td>
<td>Windows are tabbed when created.</td>
</tr>
</tbody>
</table>

32.128.3 Constructor & Destructor Documentation

32.128.3.1 Fl_Sys_Menu_Bar()

The constructor.
On Mac OS X, all arguments are unused. On other platforms they are used as by Fl_Menu_Bar::Fl_Menu_Bar().

32.128.4 Member Function Documentation

32.128.4.1 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

| cb | a callback that will be called by "About myprog" menu item with NULL 1st argument. |
| data | a pointer transmitted as 2nd argument to the callback. |

32.128.4.2 add() [1/3]

int Fl_Sys_Menu_Bar::add ( 
   const char * label, 
   const char * shortcut, 
   Fl_Callback * cb, 
   void * user_data = 0, 
   int flags = 0 ) [inline]

Adds a new menu item.
See also
Fl_Menu_Bar::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

32.128.4.3 add() [2/3]

int Fl_Sys_Menu_Bar::add ( 
   Fl_Callback * cb, 
   void * user_data = 0, 
   int flags = 0 ) [inline]

Generated by Doxygen
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)

32.128.4.4 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)

32.128.4.5 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()

32.128.4.6 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)
32.128.4.7  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

32.128.4.8  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.

32.128.4.9  insert() [1/2]

int Fl_Sys_Menu_Bar::insert (  
   int index,
   const char ∗label,
   const char ∗shortcut,
   Fl_Callback ∗cb,
   void ∗user_data = 0,
   int flags = 0   )  [inline]

Insert a new menu item.
See also
   Fl_Menu_::insert(int index, const char ∗label, const char ∗shortcut, Fl_Callback ∗cb, void ∗user_data=0, int flags=0)

32.128.4.10  insert() [2/2]

int Fl_Sys_Menu_Bar::insert (   
   int index,
   const char ∗label,
   int shortcut,
   Fl_Callback ∗cb,
   void ∗user_data = 0,
   int flags = 0   )

insert in the system menu bar a new menu item
Insert in the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.
Returns

the index into the menu() array, where the entry was inserted

See also

Fl_Menu_::insert(int index, const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags)

### 32.128.4.11 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 |

### 32.128.4.12 mode()

```cpp
void Fl_Sys_Menu_Bar::mode (int i, int fl)
```

Sets the flags of item i.

See also

Fl_Menu_::mode(int i, int fl)

### 32.128.4.13 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 |

### 32.128.4.14 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 |
Parameters

| name | the new item name as a UTF8 string |

### 32.128.4.15 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`.

### 32.128.4.16 window_menu_style()

```cpp
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`

### 32.129 Fl_Table Class Reference

A table of widgets or other content.

```cpp
#include <Fl_Table.H>
```

Inheritance diagram for `Fl_Table`:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Table
  ↓
Fl_Table_Row
```

Generated by Doxygen
Public Types

- enum TableContext {
  CONTEXT_NONE = 0 , CONTEXT_STARTPAGE = 0x01 , CONTEXT_ENDPAGE = 0x02 , CONTEXT_ROW_HEADER = 0x04 , CONTEXT_COL_HEADER = 0x08 , CONTEXT_CELL = 0x10 , CONTEXT_TABLE = 0x20 , CONTEXT_RC_RESIZE = 0x40 
}

The context bit flags for Fl_Table related callbacks.

Public Member Functions

- void add (Fl_Widget &wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table’s group.
- void add (Fl_Widget *wgt)
  The specified widget is removed from its current group (if any) and added to the end of Fl_Table’s group.
- Fl_Widget *const * array ()
  Returns a pointer to the array of children.
- void begin ()
- void callback (Fl_Widget *, void *)
  Callbacks will be called depending on the setting of Fl_Widget::when().
- int callback_col ()
  Returns the current column the event occurred on.
- TableContext callback_context ()
  Returns the current 'table context'.
- int callback_row ()
  Returns the current row the event occurred on.
- Fl_Widget * child (int n) const
  Returns the child widget by an index.
- int children () const
  Returns the number of children in the table.
- virtual void clear ()
  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()|end()| or add()|insert()|etc.
- int col_header ()
  Returns if column headers are enabled or not.
- void col_header (int flag)
  Enable or disable column headers.
- Fl_Color col_header_color ()
  Gets the color for column headers.
- void col_header_color (Fl_Color val)
  Sets the color for column headers and redraws the table.
- int col_header_height ()
  Gets the column header height.
- void col_header_height (int height)
  Sets the height in pixels for column headers and redraws the table.
- int col_position ()
  Returns the current column scroll position as a column number.
- void col_position (int col)
  Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.
- int col_resize ()
  Returns if column resizing by the user is allowed.
- void col_resize (int flag)
Allows/disallows column resizing by the user.

- **int col_resize_min ()**
  
  Returns the current column minimum resize value.

- **void col_resize_min (int val)**
  
  Sets the current column minimum resize value.

- **int col_width (int col)**
  
  Returns the current width of the specified column in pixels.

- **void col_width (int col, int width)**
  
  Sets the width of the specified column in pixels, and the table is redrawn.

- **void col_width_all (int width)**
  
  Convenience method to set the width of all columns to the same value, in pixels.

- **int cols ()**
  
  Get the number of columns in the table.

- **virtual void cols (int val)**
  
  Set the number of columns in the table and redraw.

- **void do_callback (TableContext context, int row, int col)**
  
  Calls the widget callback.

- **void end ()**

- **int find (const Fl_Widget &wgt) const**

- **int find (const Fl_Widget ∗wgt) const**

- **Fl_Table (int X, int Y , int W, int H, const char ∗l=0)**
  
  The constructor for Fl_Table.

- **void get_selection (int &row_top, int &col_left, int &row_bot, int &col_right)**
  
  Gets the region of cells selected (highlighted).

- **void init_sizes ()**
  
  Resets the internal array of widget sizes and positions.

- **void insert (Fl_Widget &wgt, Fl_Widget ∗w2)**
  
  The specified widget is removed from its current group (if any) and inserted into Fl_Table’s group before widget ’w2’.

- **void insert (Fl_Widget &wgt, int n)**
  
  The specified widget is removed from its current group (if any) and inserted into the Fl_Table’s group at position ’n’.

- **int is_interactive_resize ()**
  
  Returns 1 if someone is interactively resizing a row or column.

- **int is_selected (int r, int c)**
  
  See if the cell at row r and column c is selected.

- **int move_cursor (int R, int C)**
  
  Same as move_cursor(R,C,1);.

- **int move_cursor (int R, int C, int shiftselect)**
  
  Moves the selection cursor a relative number of rows/columns specified by R/C.

- **void remove (Fl_Widget &wgt)**
  
  The specified widget is removed from Fl_Table’s group.

- **void resize (int X, int Y , int W, int H)**
  
  Handle resize events if user resizes parent window.

- **int row_header ()**
  
  Returns if row headers are enabled or not.

- **void row_header (int flag)**
  
  Enables/disables showing the row headers.

- **Fl_Color row_header_color ()**
  
  Returns the current row header color.

- **void row_header_color (Fl_Color val)**
  
  Sets the row header color and causes the screen to redraw.

- **int row_header_width ()**
Returns the current row header width (in pixels).

- `void row_header_width (int width)`
  
  Sets the row header width to \( n \) and causes the screen to redraw.

- `int row_height (int row)`
  
  Returns the current height of the specified row as a value in pixels.

- `void row_height (int row, int height)`
  
  Sets the height of the specified row in pixels, and the table is redrawn.

- `void row_height_all (int height)`
  
  Convenience method to set the height of all rows to the same value, in pixels.

- `int row_position ()`
  
  Returns the current row scroll position as a row number.

- `void row_position (int row)`
  
  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.

- `int row_resized ()`
  
  Returns if row resizing by the user is allowed.

- `void row_resized (int flag)`
  
  Allows/disallows row resizing by the user.

- `int row_resized_min ()`
  
  Returns the current row minimum resize value.

- `void row_resized_min (int val)`
  
  Sets the current row minimum resize value.

- `int rows ()`
  
  Returns the number of rows in the table.

- `virtual void rows (int val)`
  
  Sets the number of rows in the table, and the table is redrawn.

- `int scrollbar_size () const`
  
  Gets the current size of the scrollbars’ troughs, in pixels.

- `void scrollbar_size (int newSize)`
  
  Sets the pixel size of the scrollbars’ troughs to \( \text{newSize} \), in pixels.

- `void set_selection (int row_top, int col_left, int row_bot, int col_right)`
  
  Sets the region of cells to be selected (highlighted).

- `int tab_cell_nav () const`
  
  Get state of table's 'Tab' key cell navigation flag.

- `void tab_cell_nav (int val)`
  
  Flag to control if Tab navigates table cells or not.

- `void table_box (Fl_Boxtype val)`
  
  Sets the kind of box drawn around the data table, the default being \text{FL\_NO\_BOX}.

- `Fl_Boxtype table_box (void)`
  
  Returns the current box type used for the data table.

- `int top_row ()`
  
  Returns the current top row shown in the table.

- `void top_row (int row)`
  
  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

- `void visible_cells (int &r1, int &r2, int &c1, int &c2)`
  
  Returns the range of row and column numbers for all visible and partially visible cells in the table.

- `void when (Fl_When flags)`
  
  The \text{Fl\_Widget::when()} function is used to set a group of flags, determining when the widget callback is called:

  - `~Fl_Table ()`
    
    The destructor for \text{Fl\_Table}. 

Generated by Doxygen
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 fltk widgets (if any)

- int table_h
  
  table's virtual height (in pixels)

- int table_w
  
  table's virtual width (in pixels)

- int tih
  
  Data table's inner h dimension, inside bounding box. See Table Dimension Diagram.

- int tiw
  
  Data table's inner w dimension, inside bounding box. See Table Dimension Diagram.

- int tix
  
  Data table's inner x dimension, inside bounding box. See Table Dimension Diagram.

- int 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 toprow
  
  top row# of currently visible table on screen

- int toprow_scrollpos
  
  precomputed scroll position for top row

- int tow
  
  Data table's outer w dimension, outside bounding box. See Table Dimension Diagram.

- int tox
  
  Data table's outer x dimension, outside bounding box. See Table Dimension Diagram.

- int toy
  
  Data table's outer y dimension, outside bounding box. See Table Dimension Diagram.

- Fl_Scrollbar * vscrollbar
  
  child vertical scrollbar widget

- int wih
  
  Table widget's inner h dimension, inside bounding box. See Table Dimension Diagram.

- int wiw
Table widget’s inner y dimension, inside bounding box. See Table Dimension Diagram.

- int wiy

Table widget’s inner y dimension, inside bounding box. See Table Dimension Diagram.

Additional Inherited Members

32.129.1 Detailed Description

A table of widgets or other content.

This is the base class for table widgets. To be useful it must be subclassed and several virtual functions defined. Normally applications use widgets derived from this widget, and do not use this widget directly; this widget is usually too low level to be used directly by applications.

This widget does not handle the data in the table. The draw_cell() method must be overridden by a subclass to manage drawing the contents of the cells.

This widget can be used in several ways:

- As a custom widget; see examples/table-simple.cxx and test/table.cxx. Very optimal for even extremely large tables.
- As a table made up of a single FLTK widget instanced all over the table, simulating a numeric spreadsheet. See examples/table-spreadsheet.cxx and examples/table-spreadsheet-with-keyboard-nav.cxx. Optimal for large tables.
- As a regular container of FLTK widgets, one widget per cell. See examples/table-as-container.cxx. Not recommended for large tables.

When acting as part of a custom widget, events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the setting for Fl_Table::when().

When acting as a container for FLTK widgets, the FLTK widgets maintain themselves. Although the draw_cell() method must be overridden, its contents can be very simple. See the draw_cell() code in examples/table-simple.cxx.
The following variables are available to classes deriving from `Fl_Table`:

<table>
<thead>
<tr>
<th>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/tyi/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]

**32.129.2 Member Enumeration Documentation**
32.129.2.1 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>

32.129.3 Constructor & Destructor Documentation

32.129.3.1 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.

32.129.3.2 ~Fl_Table()

Fl_Table::~Fl_Table ( )
The destructor for Fl_Table. Destroys the table and its associated widgets.

32.129.4 Member Function Documentation

32.129.4.1 array()

Fl_Widget* const* Fl_Table::array ( ) [inline]
Returns a pointer to the array of children. This pointer is only valid until the next time a child is added or removed.

32.129.4.2 callback()

void Fl_Table::callback (  
  Fl_Widget *,  
  void *)

Callbacks will be called depending on the setting of Fl_Widget::when(). Callback functions should use the following functions to determine the context/row/column:
32.129 Fl_Table Class Reference

- Fl_Table::callback_row() returns current row
- Fl_Table::callback_col() returns current column
- Fl_Table::callback_context() returns current table context

callback_row() and callback_col() will be set to the row and column number the event occurred on. If someone clicked on a row header, col will be 0. If someone clicked on a column header, row will be 0.

callback_context() will return one of the following:

<table>
<thead>
<tr>
<th>Fl_Table::CONTEXT_ROW_HEADER</th>
<th>Someone clicked on a row header. Excludes resizing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Table::CONTEXT_COL_HEADER</td>
<td>Someone clicked on a column header. Excludes resizing.</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_CELL</td>
<td>Someone clicked on a cell. To receive callbacks for FL_RELEASE events, you must set when(FL_WHEN_RELEASE).</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_RC_RESIZE</td>
<td>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).</td>
</tr>
</tbody>
</table>

```cpp
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
        }
    }
```

32.129.4.3 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.

32.129.4.4 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.

32.129.4.5 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.
child()

```
Fl_Widget* Fl_Table::child (int n) const [inline]
```

Returns the child widget by an index.
When using the Fl_Table as a container for FLTK widgets, this method returns the widget pointer from the internal
array of widgets in the container.
Typically used in loops, eg:
```
for (int i=0; i<children(); i++) {
    Fl_Widget *w = child(i);
    ...
}
```

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)

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()

```
void Fl_Table::col_header (int flag) [inline]
```
Enable or disable column headers.
If changed, the table is redrawn.

col_resize()

```
void Fl_Table::col_resize (int flag) [inline]
```
Allows/disallows column resizing by the user.
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the column head-
ers, col_header() must also be enabled to allow resizing.

col_resize_min()

```
void Fl_Table::col_resize_min (int val) [inline]
```
Sets the current column minimum resize value.
This is used to prevent the user from interactively resizing any column to be smaller than 'pixels'. Must be a value
>=1.
32.129.4.12 col_width()

```c
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`.

32.129.4.13 col_width_all()

```c
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.

32.129.4.14 cursor2rowcol()

```c
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.

32.129.4.15 damage_zone()

```c
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()`.

32.129.4.16 do_callback()

```c
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()`.

32.129.4.17 draw()

```c
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
32.129.4.18  draw_cell()

virtual void Fl_Table::draw_cell (  
    TableContext  context,  
    int  R = 0,  
    int  C = 0,  
    int  X = 0,  
    int  Y = 0,  
    int  W = 0,  
    int  H = 0 ) [inline], [protected], [virtual]

Subclass should override this method to handle drawing the cells.
This method will be called whenever the table is redrawn, once per cell.
Only cells that are completely (or partially) visible will be told to draw.
context will be one of the following:

<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Table::CONTEXT_STARTPAGE</td>
<td>When table, or parts of the table, are about to be redrawn. Use to initialize static data, such as font selections. R/C will be zero, X/Y/W/H will be the dimensions of the table's entire data area. (Useful for locking a database before accessing; see also visible_cells())</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_ENDPAGE</td>
<td>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)</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_ROW_HEADER</td>
<td>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</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_COL_HEADER</td>
<td>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</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_CELL</td>
<td>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</td>
</tr>
<tr>
<td>Fl_Table::CONTEXT_RC_RESIZE</td>
<td>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.</td>
</tr>
</tbody>
</table>

R and C will be set to the row and column number of the cell being drawn. In the case of row headers, C will be 0. In the case of column headers, R will be 0. X/Y/W/H will be the position and dimensions of where the cell should be drawn. In the case of custom widgets, a minimal draw_cell() override might look like the following. With custom widgets it is up to the caller to handle drawing everything within the dimensions of the cell, including handling the selection color. Note all clipping must be handled as well; this allows drawing outside the dimensions of the cell if so desired for 'custom effects'.

```c++
// This is called whenever Fl_Table wants you to draw a cell
void MyTable::draw_cell(TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0) {
    static char s[40];
    sprintf(s, "%d/%d", R, C); // text for each cell
    switch ( context ) {  
    case CONTEXT_STARTPAGE: // Fl_Table telling us it's starting to draw page
        fl_font(FL_HELVETICA, 16);  
        return;
    case CONTEXT_ROW_HEADER: // Fl_Table telling us to draw row/col headers
        case CONTEXT_COL_HEADER:
        fl_push_clip(X, Y, W, H);
        {   
            fl_draw_box(FL_THIN_UP_BOX, X, Y, W, H, color());
        }
    }
```
32.129 Fl_Table Class Reference

```c
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
}
```

### 32.129.4.19 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.
```

### 32.129.4.20 get_selection()

```
void Fl_Table::get_selection ( int & row_top, int & col_left, int & row_bot, int & col_right )
Gets the region of cells selected (highlighted).

Parameters

| in | row_top | Returns the top row of selection area |
| in | col_left | Returns the left column of selection area |
| in | row_bot | Returns the bottom row of selection area |
| in | col_right | Returns the right column of selection area |
```

### 32.129.4.21 init_sizes()

```
void Fl_Table::init_sizes ( ) [inline]
```

Generated by Doxygen
Resets the internal array of widget sizes and positions.

See also

Fl_Group::init_sizes()

32.129.4.22 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.

32.129.4.23 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().

32.129.4.24 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.

32.129.4.25 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. Internal variables select_row/select_col and current_row/current_col are modified, among others.

Examples:

- `R=1, C=0` -- moves the selection cursor one row downward.
- `R=5, C=0` -- moves the selection cursor 5 rows downward.
- `R=-5, C=0` -- moves the cursor 5 rows upward.
- `R=2, C=2` -- moves the cursor 2 rows down and 2 columns to the right.

32.129.4.26 recalc_dimensions()

```cpp
void Fl_Table::recalc_dimensions ( ) [protected]
```

Recalculate the dimensions of the table, and affect any children. Internally, Fl_Group::resizer() and init_sizes() are called.
32.129.4.27  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.

32.129.4.28  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.

32.129.4.29  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.

32.129.4.30  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.

32.129.4.31  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.

32.129.4.32  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.
### row_resize()

```cpp
void Fl_Table::row_resize ( int flag ) [inline]
```

Allows/disallows row resizing by the user.
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the row headers, `row_header()` must also be enabled to allow resizing.

### row_resize_min()

```cpp
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 $\geq 1$.

### scrollbar_size() [1/2]

```cpp
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>in</th>
<th>newSize</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global <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, |
```

32.129.4.37 set_selection()

```cpp
void Fl_Table::set_selection ( int row_top, int col_left, |
```

32.129.4.37 set_selection()
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>in</th>
<th>row_top</th>
<th>Top row of selection area</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>col_left</td>
<td>Left column of selection area</td>
</tr>
<tr>
<td>in</td>
<td>row_bot</td>
<td>Bottom row of selection area</td>
</tr>
<tr>
<td>in</td>
<td>col_right</td>
<td>Right column of selection area</td>
</tr>
</tbody>
</table>

32.129.4.38 tab_cell_nav() [1/2]

int Fl_Table::tab_cell_nav ( ) const [inline]
Get state of table's 'Tab' key cell navigation flag.

Returns

1 if Tab configured to navigate cells in table
0 to navigate widget focus (default)

See also

tab_cell_nav(int)

32.129.4.39 tab_cell_nav() [2/2]

void Fl_Table::tab_cell_nav ( 
    int val ) [inline]
Flag to control if Tab navigates table cells or not.
If on, Tab key navigates table cells. If off, Tab key navigates fltk widget focus. (default)
As of fltk 1.3, the default behavior of the Tab key is to navigate focus off of the current widget, and on to the next
one. But in some applications, it's useful for Tab to be used to navigate cells in the Fl_Table.

Parameters

| in  | val | If val is 1, Tab key navigates cells in table, not fltk widgets.
|     |     | If val is 0, Tab key will advance focus to the next fltk widget (default), and does not navigate cells
|     |     | in table. |

32.129.4.40 table_box()

void Fl_Table::table_box ( 
    Fl_Boxtype val ) [inline]
Sets the kind of box drawn around the data table, the default being FL_NO_BOX.
Changing this value will cause the table to redraw.

32.129.4.41 table_resized()

void Fl_Table::table_resized ( ) [protected]

Generated by Doxygen
Call this if table was resized, to recalculate internal data.  
Calls recall_dimensions(), and recalculates scrollbar sizes.

### 32.129.4.42 table_scrolled()

```cpp
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[i]xywh has already been recalculated.

### 32.129.4.43 top_row() [1/2]

```cpp
int Fl_Table::top_row ( ) [inline]
```

Returns the current top row shown in the table.  
This row may be partially obscured.

### 32.129.4.44 top_row() [2/2]

```cpp
void Fl_Table::top_row (int row) [inline]
```

Sets which row should be at the top of the table,  
scrolling as necessary, and the table is redrawn.  
If the table cannot be scrolled that far, it is scrolled as far as possible.

### 32.129.4.45 visible_cells()

```cpp
void Fl_Table::visible_cells (int & r1, int & r2, int & c1, int & c2) [inline]
```

Returns the range of row and column numbers for all visible and partially visible cells in the table.  
These values can be used e.g. by your draw_cell() routine during CONTEXT_STARTPAGE to figure out what cells  
are about to be redrawn for the purposes of locking the data from a database before it’s drawn.

```
leftcol rightcol
|: |:|:|:|:|
|toprow .. |-------------------|
| | V I S I B L E |
| | T A B L E |
| |botrow .. '-------------------'
```

e.g. in a table where the visible rows are 5-20, and the visible columns are 100-120, then those variables would be:

- toprow = 5
- botrow = 20
- leftcol = 100
- rightcol = 120

### 32.129.4.46 when()

```cpp
void Fl_Table::when (Fl_When flags)
```

The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

<table>
<thead>
<tr>
<th>FL_WHEN_CHANGED</th>
<th>callback() will be called when rows or columns are resized (interactively or via col_width() or row_height()), passing CONTEXT_RC_RESIZE via callback_context().</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_WHEN_RELEASE</td>
<td>callback() will be called during FL_RELEASE events, such as when someone releases a mouse button somewhere on the table.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
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

### 32.130 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.

- int rows ()
- void rows (int val)
  
  Sets the number of rows in the table, and the table is redrawn.

- void select_all_rows (int flag=1)
  
  This convenience function changes the selection state for all rows based on 'flag'.

- int select_row (int row, int flag=1)
  
  Changes the selection state for 'row', depending on the value of 'flag'.

- TableRowSelectMode type () const
- void type (TableRowSelectMode val)
  
  Sets the table selection mode.

- ~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

### 32.130.1 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().

### 32.130.2 Constructor & Destructor Documentation

#### 32.130.2.1 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.

#### 32.130.2.2 ~Fl_Table_Row()

Fl_Table_Row::~Fl_Table_Row () [inline]
The destructor for the Fl_Table_Row.
Destroys the table and its associated widgets.

### 32.130.3 Member Function Documentation

#### 32.130.3.1 clear()

void Fl_Table_Row::clear () [inline], [virtual]
Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.

See also
- rows(int), cols(int)

Reimplemented from Fl_Table.
32.130.3.2 row_selected()

```cpp
int Fl_Table_Row::row_selected ( int row )
```

Checks to see if `row` is selected.
Returns 1 if selected, 0 if not. You can change the selection of a row by clicking on it, or by using `select_row(row, flag)`

32.130.3.3 select_all_rows()

```cpp
void Fl_Table_Row::select_all_rows ( int flag = 1 )
```

This convenience function changes the selection state for all rows based on `flag`.
0=deselect, 1=select, 2=toggle existing state.

32.130.3.4 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.

32.130.3.5 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`

32.131 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 &lt;Fl_Tabs.H&gt;
```

Inheritance diagram for `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.
- Fl_Align tab_align () const
  Gets the tab label alignment.
- void tab_align (Fl_Align a)
  Sets the tab label alignment.
- Fl_Widget * value ()
  Gets the currently visible widget/tab.
- int value (Fl_Widget *)
  Sets the widget to become the current visible widget/tab.
- virtual Fl_Widget * which (int event_x, int event_y)
  Return 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

32.131.1 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.
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();
}
```

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.

Highlighting The Selected Tab

Figure 32.45 Fl_Tabs

Figure 32.46 Fl_Tabs Default Appearance
The selected "tab" can be highlighted further by setting the `selection_color()` of the Fl_Tab itself, e.g.

```cpp
tabs = new Fl_Tabs(..);
tabs->selection_color(FL_DARK3);
```

The result of the above looks like:

![Figure 32.47 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.

```cpp
void MyTabCallback(Fl_Widget *w, void*) {
    Fl_Tabs *tabs = (Fl_Tabs*)w;
    // When tab changed, make sure it has same color as its group
    tabs->selection_color( (tabs->value())->color() );
}
```

```cpp
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 32.48 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.

```cpp
```
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 when() is FL_WHEN_RELEASE (inherited from Fl_Widget).
3. If FL_WHEN_RELEASE is the only flag specified, the behavior will be as if (FL_WHEN_RELEASE|FL_WHEN_CHANGED) was specified.
4. The value of changed() will be valid during the callback.
5. If both FL_WHEN_CHANGED and FL_WHEN_NOT_CHANGED are specified, the callback is invoked whether the tab has been changed or not. The changed() method can be used to determine the cause.
6. FL_WHEN_NOT_CHANGED can happen if someone clicks on an already selected tab, or if a keyboard navigation attempt results in no change to the tabs, such as using the arrow keys while at the left or right end of the tabs.

### 32.131.2 Constructor & Destructor Documentation

#### 32.131.2.1 Fl_Tabs()

```
Fl_Tabs::Fl_Tabs (int X, int Y, int W, int H, const char ∗ L = 0 )
```

Creates a new Fl_Tabs widget using the given position, size, and label string.
The default boxtype is FL_THIN_UP_BOX.
Use add(Fl_Widget∗) to add each child, which are usually Fl_Group widgets. The children should be sized to stay away from the top or bottom edge of the Fl_Tabs widget, which is where the tabs will be drawn.
All children of Fl_Tabs should have the same size and exactly fit on top of each other. They should only leave space above or below where the tabs will go, but not on the sides. If the first child of Fl_Tabs is set to "resizable()", the riders will not resize when the tabs are resized.
The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Tabs and all of its children can be automatic (local) variables, but you must declare the Fl_Tabs widget first so that it is destroyed last.
32.131.3 Member Function Documentation

32.131.3.1 client_area()

```cpp
void Fl_Tabs::client_area (  
    int & rx,  
    int & ry,  
    int & rw,  
    int & rh,  
    int tabh = 0 )
```

Returns the position and size available to be used by its children. If there isn't any child yet the `tabh` parameter will be used to calculate the return values. This assumes that the children's labelsize is the same as the Fl_Tabs' labelsize and adds a small border. If there are already children, the values of child(0) are returned, and `tabh` is ignored.

Note

Children should always use the same positions and sizes.

`tabh` can be one of

- 0: calculate label size, tabs on top
- -1: calculate label size, tabs on bottom
- $>0$: use given `tabh` value, tabs on top (height = `tabh`)
- $<-1$: use given `tabh` value, tabs on bottom (height = `-tabh`)

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tabh</th>
<th>position and optional height of tabs (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>rx, ry, rw, rh</td>
<td>(x,y,w,h) of client area for children</td>
</tr>
</tbody>
</table>

Since

FLTK 1.3.0

32.131.3.2 draw()

```cpp
void Fl_Tabs::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

32.131.3.3 handle()

```cpp
int Fl_Tabs::handle (  
    int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in | event | the kind of event received |

Return values

| 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.

32.131.3.4 push() [1/2]

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*).

32.131.3.5 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().

32.131.3.6 tab_align() [1/2]

Fl_Align Fl_Tabs::tab_align ( ) const [inline]

Gets the tab label alignment.

See also

tab_align(Fl_Align)
32.131.3.7 tab_align() [2/2]

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.

32.131.3.8 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.

32.131.3.9 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)

32.131.3.10 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

32.132 Fl_Text_Buffer Class Reference

This class manages Unicode text displayed in one or more Fl_Text_Display widgets.
#include <Fl_Text_Buffer.H>

Public Member Functions

• void add_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)
  Adds a callback function that is called whenever the text buffer is modified.
• void add_predelete_callback (Fl_Text_Predelete_Cb bufPredelCB, void *cbArg)
  Adds a callback routine to be called before text is deleted from the buffer.
• char * address (int pos)
  Convert a byte offset in buffer into a memory address.
• const char * address (int pos) const
  Convert a byte offset in buffer into a memory address.
• void append (const char *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.
• int highlight ()
  Returns a non-zero value if text has been highlighted, 0 otherwise.
• void highlight (int start, int end)
  Highlights the specified text within the buffer.
• int highlight_position (int *start, int *end) const
  Highlights the specified text between start and end within the buffer.
• const Fl_Text_Selection * highlight_selection () const
  Returns the current highlight selection.
• char * highlight_text ()
  Returns the highlighted text.
• void insert (int pos, const char *text)
  Inserts null-terminated string text at position pos.
• int insertfile (const char *file, int pos, int buflen=128 * 1024)
  Inserts a file at the specified position.
• 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 \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).

- \textbf{int line_start (int pos) const}
  
  Returns the position of the start of the line containing position \texttt{pos}.

- \textbf{char * line_text (int pos) const}
  
  Returns the text from the entire line containing the specified character position.

- \textbf{int loadfile (const char *file, int buflen=128 *1024)}
  
  Loads a text file into the buffer.

- \textbf{int next_char (int ix) const}
  
  Returns the index of the next character.

- \textbf{int next_char_clipped (int ix) const}

- \textbf{int outputfile (const char *file, int start, int end, int buflen=128 *1024)}
  
  Writes the specified portions of the text buffer to a file.

- \textbf{int prev_char (int ix) const}
  
  Returns the index of the previous character.

- \textbf{int prev_char_clipped (int ix) const}

- \textbf{Fl_Text_Selection * primary_selection ()}
  
  Returns the primary selection.

- \textbf{const Fl_Text_Selection * primary_selection () const}
  
  Returns the primary selection.

- \textbf{void printf (const char *fmt,...)}
  
  Appends printf formatted messages to the end of the buffer.

- \textbf{void remove (int start, int end)}
  
  Deletes a range of characters in the buffer.

- \textbf{void remove_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)}
  
  Removes a modify callback.

- \textbf{void remove_predelete_callback (Fl_Text_Predelete_Cb predelCB, void *cbArg)}
  
  Removes a callback routine \texttt{bufPreDeleteCB} associated with argument \texttt{cbArg} to be called before text is deleted from the buffer.

- \textbf{void remove_secondary_selection ()}
  
  Removes the text from the buffer corresponding to the secondary text selection object.

- \textbf{void remove_selection ()}
  
  Removes the text in the primary selection.

- \textbf{void replace (int start, int end, const char *text)}
  
  Deletes the characters between \texttt{start} and \texttt{end}, and inserts the null-terminated string \texttt{text} in their place in the buffer.

- \textbf{void replace_secondary_selection (const char *text)}
  
  Replaces the text from the buffer corresponding to the secondary text selection object with the new string \texttt{text}.

- \textbf{void replace_selection (const char *text)}
  
  Replaces the text in the primary selection.

- \textbf{int 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.

- \textbf{int savefile (const char *file, int buflen=128 *1024)}
  
  Saves a text file from the current buffer.

- \textbf{int 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}.

- \textbf{int search_forward (int startPos, const char *searchString, int *foundPos, int matchCase=0) const}
  
  Search forwards in buffer for string \texttt{searchString}, starting with the character \texttt{startPos}, and returning the result in \texttt{foundPos}.

- \textbf{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
  Returns the position corresponding to the end of the word.
- int word_start (int pos) const
  Returns the position corresponding to the start of the word.
- ∼Fl_Text_Buffer ()
  Frees a text buffer.
Public Attributes

- int input_file_was_transcoded
  true if the loaded file has been transcoded to UTF-8.
- void(* transcoding_warning_action )(Fl_Text_Buffer *)
  Pointer to a function called after reading a non UTF-8 encoded file.

Static Public Attributes

- static const char * file_encoding_warning_message
  This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

Protected Member Functions

- void call_modify_callbacks (int pos, int nDeleted, int nInserted, int nRestyled, const char *deletedText) const
  Calls the stored modify callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.
- void call_predelete_callbacks (int pos, int nDeleted) const
  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.
- int insert_ (int pos, const char *text)
  Internal (non-redisplaying) version of insert().
- void move_gap (int pos)
  Move the gap to start at a new position.
- void reallocate_with_gap (int newGapStart, int newGapLen)
  Reallocates the text storage in the buffer to have a gap starting at newGapStart and a gap size of newGapLen, preserving the buffer's current contents.
- void redisplay_selection (Fl_Text_Selection *oldSelection, Fl_Text_Selection *newSelection) const
  Calls the stored redisplay procedure(s) for this buffer to update the screen for a change in a selection.
- void remove_ (int start, int end)
  Internal (non-redisplaying) version of remove().
- void remove_selection_ (Fl_Text_Selection *sel)
  Removes the text from the buffer corresponding to sel.
- void replace_selection_ (Fl_Text_Selection *sel, const char *text)
  Replaces the text in selection sel.
- char * selection_text_ (Fl_Text_Selection *sel) const
- void update_selections (int pos, int nDeleted, int nInserted)
  Updates all of the selections in the buffer for changes in the buffer's text.

Protected Attributes

- char * mBuf
  allocated memory where the text is stored
- char mCanUndo
  if this buffer is used for attributes, it must not do any undo calls
- void ** mCbArgs
  caller arguments for modifyProcs above
- int mCursorPosHint
  hint for reasonable cursor position after a buffer modification operation
- int mGapEnd
  points to the first character after the gap
- int mGapStart
  points to the first character of the gap
32.132 Fl_Text_Buffer Class Reference

- 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 nModifyProcs
  number of modify-redisplay procs attached
- int nPredeleteProcs
  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.

32.132.1 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/.

32.132.2 Constructor & Destructor Documentation

32.132.2.1 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>
### 32.132.3 Member Function Documentation

#### 32.132.3.1 `add_modify_callback()`

```c
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);
```

#### 32.132.3.2 `address()` [1/2]

```c
char* Fl_Text_Buffer::address(int pos)
```

Converts a byte offset in buffer into a memory address.

**Parameters**

- `pos` byte offset into buffer

**Returns**

- byte offset converted to a memory address

#### 32.132.3.3 `address()` [2/2]

```c
const char* Fl_Text_Buffer::address(int pos)
```

Converts a byte offset in buffer into a memory address.

**Parameters**

- `pos` byte offset into buffer

**Returns**

- byte offset converted to a memory address

#### 32.132.3.4 `append()`

```c
void Fl_Text_Buffer::append(const char * t)
```

Appends the text string to the end of the buffer.

**Parameters**

- `t` UTF-8 encoded and nul terminated text
32.132.3.5 appendfile()

```cpp
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().

32.132.3.6 byte_at()

```cpp
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

32.132.3.7 char_at()

```cpp
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

32.132.3.8 copy()

```cpp
void Fl_Text_Buffer::copy (Fl_Text_Buffer * fromBuf, int fromStart, int fromEnd, int toPos )
```

Copies text from another Fl_Text_Buffer to this one.

Parameters

| fromBuf | source text buffer, may be the same as this |
| fromStart | byte offset into buffer |
| fromEnd | byte offset into buffer |
| toPos | destination byte offset into buffer |
32.132.3.9 count_displayed_characters()

```cpp
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.

32.132.3.10 count_lines()

```cpp
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.

32.132.3.11 findchar_backward()

```cpp
int Fl_Text_Buffer::findchar_backward ( int startPos, unsigned int searchChar, int *foundPos ) const
```

Search backwards in buffer `buf` for character `searchChar`, starting with the character before `startPos`, returning the result in `foundPos`. Returns 1 if found, 0 if not. The difference between this and `search_backward()` is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchChar</td>
<td>UCS-4 character that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the character was found</td>
</tr>
</tbody>
</table>

**Returns**

1 if found, 0 if not

32.132.3.12 findchar_forward()

```cpp
int Fl_Text_Buffer::findchar_forward ( int startPos, unsigned searchChar, int *foundPos ) const
```

Finds the next occurrence of the specified character. Search forwards in buffer for character `searchChar`, starting with the character `startPos`, and returning the result in `foundPos`. Returns 1 if found, 0 if not. The difference between this and `search_forward()` is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

**Parameters**

<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>
32.132.3.13 highlight_text()

char * Fl_Text_Buffer::highlight_text ( )

Returns the highlighted text.

When you are done with the text, free it using the free() function.

32.132.3.14 insert()

void Fl_Text_Buffer::insert ( int pos,  
const char * text )

Inserts null-terminated string text at position pos.

Parameters

| pos  | insertion position as byte offset (must be UTF-8 character aligned) |
| text | UTF-8 encoded and nul terminated text |

32.132.3.15 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

32.132.3.16 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.

32.132.3.17  is_word_separator()

bool Fl_Text_Buffer::is_word_separator ( int pos ) const

Returns whether character at position pos is a word separator. Pos must be at a character boundary.

32.132.3.18  length()

int Fl_Text_Buffer::length ( ) const [inline]

Returns the number of bytes in the buffer.

Returns

size of text in bytes

32.132.3.19  line_end()

int Fl_Text_Buffer::line_end ( int pos ) const

Finds and returns the position of the end of the line containing position pos (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).

Parameters

pos  byte index into buffer

Returns

byte offset to line end

32.132.3.20  line_start()

int Fl_Text_Buffer::line_start ( int pos ) const

Returns the position of the start of the line containing position pos.

Parameters

pos  byte index into buffer

Returns

byte offset to line start

32.132.3.21  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

### 32.132.3.22 loadfile()

```c
int Fl_Text_Buffer::loadfile (const char * file,
   int buflen = 128*1024 ) [inline]
```

Loads a text file into the buffer.
See also insertfile().

### 32.132.3.23 next_char()

```c
int Fl_Text_Buffer::next_char (int ix) const
```

Returns the index of the next character.

Parameters

- `ix` index to the current character

### 32.132.3.24 outputfile()

```c
int Fl_Text_Buffer::outputfile (const char * file,
   int start,
   int end,
   int buflen = 128*1024 )
```

Writes the specified portions of the text buffer to a file.
Returns

- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

See also
savefile(const char *file, int buflen)

### 32.132.3.25 prev_char()

```c
int Fl_Text_Buffer::prev_char (int ix) const
```

Returns the index of the previous character.
### 32.132.3.26 printf()

```cpp
void Fl_Text_Buffer::printf (const char * fmt, ...)
```

Appends printf formatted messages to the end of the buffer.

**Example:**
```cpp
#include <FL/Fl_Text_Display.H>
int main(..) {
    // Create a text display widget and assign it a text buffer
    Fl_Text_Display *tdsp = new Fl_Text_Display(..);
    Fl_Text_Buffer *tbuf = new Fl_Text_Buffer();
    tdsp->buffer(tbuf);
    // Append three lines of formatted text to the buffer
    tbuf->printf("The current date is: %s.
    The time is: %s
    The current PID is %ld.
", date_str, time_str);
}
```

**Note**

The expanded string is currently limited to 1024 characters.

**Parameters**

- `ix` index to the current character

### 32.132.3.27 remove()

```cpp
void Fl_Text_Buffer::remove (int start, int end)
```

Deletes a range of characters in the buffer.

**Parameters**

- `start` byte offset to first character to be removed
- `end` byte offset to character after last character to be removed
int end,
    const char * text )

Deletes the characters between \texttt{start} and \texttt{end}, and inserts the null-terminated string \texttt{text} in their place in the buffer.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>byte offset to first character to be removed and new insert position</td>
</tr>
<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 null terminated text</td>
</tr>
</tbody>
</table>

32.132.3.30 rewind_lines()

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.

32.132.3.31 savefile()

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)}

32.132.3.32 search_backward()

int Fl_Text_Buffer::search_backward (  
    int startPos,
    const char * searchString,
    int * foundPos,
    int matchCase = 0 ) const

Search backwards in buffer for string \texttt{searchString}, starting with the character at \texttt{startPos}, returning the result in \texttt{foundPos}. Returns 1 if found, 0 if not.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPos</td>
<td>byte offset to position</td>
</tr>
<tr>
<td>searchString</td>
<td>UTF-8 string that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the string was found</td>
</tr>
<tr>
<td>matchCase</td>
<td>if set, match character case</td>
</tr>
</tbody>
</table>
Returns

1 if found, 0 if not

32.132.3.33  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**

<table>
<thead>
<tr>
<th>startPos</th>
<th>byte offset to start position</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>UTF-8 string that we want to find</td>
</tr>
<tr>
<td>foundPos</td>
<td>byte offset where the string was found</td>
</tr>
<tr>
<td>matchCase</td>
<td>if set, match character case</td>
</tr>
</tbody>
</table>

Returns

1 if found, 0 if not

32.132.3.34  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.

32.132.3.35  selection_text()

```cpp
char * Fl_Text_Buffer::selection_text ( )
```

Returns the currently selected text. When you are done with the text, free it using the `free()` function.

32.132.3.36  skip_displayed_characters()

```cpp
int Fl_Text_Buffer::skip_displayed_characters {
    int lineStartPos,
    int nChars )
```

Count forward from buffer position `lineStartPos` in displayed characters. Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.

**Parameters**

<table>
<thead>
<tr>
<th>lineStartPos</th>
<th>byte offset into buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>nChars</td>
<td>number of bytes that are sent to the display</td>
</tr>
</tbody>
</table>
32.132.3.37  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.

32.132.3.38  text() [1/2]

cCHAR * 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

32.132.3.39  text() [2/2]

void Fl_Text_Buffer::text (  
    const char * text )

Replaces the entire contents of the text buffer.

Parameters

| text | Text must be valid UTF-8. If null, an empty string is substituted. |

32.132.3.40  text_range()

cCHAR * Fl_Text_Buffer::text_range (  
    int start,  
    int end ) const

Get a copy of a part of the text buffer.
Return a copy of the text between start and end character positions from text buffer buf. Positions start at 0, and the range does not include the character pointed to by end. When you are done with the text, free it using the free() function.

Parameters

| start | byte offset to first character |
| end   | byte offset after last character in range |

Returns

newly allocated text buffer - must be free'd, text is UTF-8

32.132.3.41  vprintf()

void Fl_Text_Buffer::vprintf (  
    const char * fmt,  

Generated by Doxygen
Can be used by subclasses that need their own printf() style functionality.

\texttt{Fl Simple Terminal::printf()} would wrap around this method.

\textbf{Note}

The expanded string is currently limited to 1024 characters.

\begin{table}
\begin{tabular}{|l|l|}
\hline
\textbf{in} & \textit{fmt} is a printf format string for the message text. \\
\hline
\textbf{in} & \textit{ap} is a \texttt{va_list} created by \texttt{va_start()} and closed with \texttt{va_end()}, which the caller is responsible for handling. \\
\hline
\end{tabular}
\end{table}

\subsection{word_end()}

\begin{verbatim}
int Fl_Text_Buffer::word_end ( 
  int pos ) const
\end{verbatim}

Returns the position corresponding to the end of the word.

\begin{table}
\begin{tabular}{|l|}
\hline
\textbf{pos} byte index into buffer \\
\hline
\end{tabular}
\end{table}

\subsection{word_start()}

\begin{verbatim}
int Fl_Text_Buffer::word_start ( 
  int pos ) const
\end{verbatim}

Returns the position corresponding to the start of the word.

\begin{table}
\begin{tabular}{|l|}
\hline
\textbf{pos} byte index into buffer \\
\hline
\end{tabular}
\end{table}

\subsection{Member Data Documentation}

\subsubsection{file_encoding_warning_message}

\begin{verbatim}
const char * Fl_Text_Buffer::file_encoding_warning_message [static]
\end{verbatim}

Initial value:

\begin{verbatim}
"Displayed text contains the UTF-8 transcoding\n"of the input file which was not UTF-8 encoded.\n"Some changes may have occurred."
\end{verbatim}

This message may be displayed using the \texttt{fl_alert()} function when a file which was not UTF-8 encoded is input.
32.132.4.2  mTabDist

int Fl_Text_Buffer::mTabDist [protected]
equiv. number of characters in a tab

32.132.4.3  transcoding_warning_action

void(* Fl_Text_Buffer::transcoding_warning_action) (Fl_Text_Buffer *)
Pointer to a function called after reading a non UTF-8 encoded file.
This function is called after reading a file if the file content was transcoded to UTF-8. Its default implementation calls
fl_alert() with the text of file_encoding_warning_message. No warning message is displayed if this pointer is set to
NULL. Use input_file_was_transcoded to be informed if file input required transcoding to UTF-8.
The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

32.133  Fl_Text_Display Class Reference

Rich text display widget.
#include <Fl_Text_Display.H>
Inheritance diagram for Fl_Text_Display:

Classes

- struct Style_Table_Entry
  This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr.

Public Types

- enum {
  NORMAL_CURSOR , CARET_CURSOR , DIM_CURSOR , BLOCK_CURSOR ,
  HEAVY_CURSOR , SIMPLE_CURSOR
  }
  text display cursor shapes enumeration

- enum {
  CURSOR_POS , CHARACTER_POS
  }
  the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two
  consecutive characters.

- enum {
  DRAG_NONE = -2 , DRAG_START_DND = -1 , DRAG_CHAR = 0 , DRAG_WORD = 1 ,
  DRAG_LINE = 2
  }
  drag types - they match Fl::event_clicks() so that single clicking to start a collection selects by character, double
  clicking selects by word and triple clicking selects by line.

- enum {
  WRAP_NONE , WRAP_AT_COLUMN , WRAP_AT_PIXEL , WRAP_AT_BOUNDS
  }
  wrap types - used in wrap_mode()
enum {
    ATTR_BGCOLOR = 0x0001 ,
    ATTR_BGCOLOR_EXT_ = 0x0002 ,
    ATTR_BGCOLOR_EXT = 0x0003 ,
    ATTR_UNDERLINE = 0x0004 ,
    ATTR_GRAMMAR = 0x0008 ,
    ATTR_SPELLING = 0x000C ,
    ATTR_STRIKE_THROUGH = 0x0010 ,
    ATTR_LINES_MASK = 0x001C
}

attribute flags in Style_Table_Entry.attr

typedef void(∗ Unfinished_Style_Cb ) (int, void ∗)

Public Member Functions

• Fl_Text_Buffer ∗ buffer () const
    Gets the current text buffer associated with the text widget.

• void buffer (Fl_Text_Buffer &buf)
    Sets the current text buffer associated with the text widget.

• void buffer (Fl_Text_Buffer ∗buf)
    Attach a text buffer to display, replacing the current buffer (if any).

• double col_to_x (double col) const
    Convert a column number into an x pixel position.

• int count_lines (int start, int end, bool start_pos_is_line_start) const
    Count the number of lines between two positions.

• Fl_Color cursor_color () const
    Gets the text cursor color.

• void cursor_color (Fl_Color n)
    Sets the text cursor color.

• int cursor_style () const
    Sets the text cursor style.

• void cursor_style (int style)
    Sets the text cursor style.

• Fl_Text_Display (int X, int Y , int W, int H, const char ∗l=0)
    Creates a new text display widget.

• Fl_Color grammar_underline_color () const
    Gets the underline color for style attribute ATTR_GRAMMAR.

• void grammar_underline_color (Fl_Color color)
    Sets the underline color for style attribute ATTR_GRAMMAR.

• virtual int handle (int e)
    Event handling.

• void hide_cursor ()
    Hides the text cursor.

• void highlight_data (Fl_Text_Buffer ∗styleBuffer, const Style_Table_Entry ∗styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void ∗cbArg)
    Attach (or remove) highlight information in text display and redisplay.

• int in_selection (int x, int y) const
    Check if a pixel position is within the primary selection.

• void insert (const char ∗text)
    Inserts "text" at the current cursor location.

• int insert_position () const
    Gets the position of the text insertion cursor for text display.

• void insert_position (int newPos)
    Sets the position of the text insertion cursor for text display.

• int line_end (int startPos, bool start_pos_is_line_start) const
    Returns the end of a line.

• int line_start (int pos) const
    Return the beginning of a line.
• Fl_Align linenumber_align () const
  Returns the alignment used for line numbers (if enabled).
• void linenumber_align (Fl_Align val)
  Set alignment for line numbers (if enabled).
• Fl_Color linenumberbgcolor () const
  Returns the background color used for line numbers (if enabled).
• void linenumberbgcolor (Fl_Color val)
  Set the background color used for line numbers (if enabled).
• Fl_Color linenumberfgcolor () const
  Return the foreground color used for line numbers (if enabled).
• void linenumberfgcolor (Fl_Color val)
  Set the foreground color used for line numbers (if enabled).
• Fl_Font linenumberfont () const
  Return the font used for line numbers (if enabled).
• void linenumberfont (Fl_Font val)
  Set the font used for line numbers (if enabled).
• const char *linenumber_format () const
  Returns the line number printf() format string.
• void linenumber_format (const char *val)
  Sets the printf() style format string used for line numbers.
• Fl_Fontsize linenumber_size () const
  Return the font size used for line numbers (if enabled).
• void linenumber_size (Fl_Fontsize val)
  Set the font size used for line numbers (if enabled).
• int linenumber_width () const
  Return the screen area width provided for line numbers.
• void linenumber_width (int width)
  Set width of screen area for line numbers.
• int move_down ()
  Moves the current insert position down one line.
• int move_left ()
  Moves the current insert position left one character.
• int move_right ()
  Moves the current insert position right one character.
• int move_up ()
  Moves the current insert position up one line.
• void next_word (void)
  Moves the current insert position right one word.
• void overstrike (const char *text)
  Replaces text at the current insert position.
• int position_style (int lineStartPos, int lineLen, int lineIndex) const
  Find the correct style for a character.
• int position_to_xy (int pos, int +x, int +y) const
  Convert a character index into a pixel position.
• void previous_word (void)
  Moves the current insert position left one word.
• 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.
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 Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

- void scrollbar_width (int width)
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

- Fl_Color secondary_selection_color () const
  Gets the background color for the secondary selection block.

- void secondary_selection_color (Fl_Color color)
  Sets the background color for the secondary selection block.

- int shortcut () const
- void shortcut (int s)

- void show_cursor (int b=1)
  Shows the text cursor.

- void show_insert_position ()
  Scrolls the text buffer to show the current insert position.

- int skip_lines (int startPos, int nLines, bool startPosIsLineStart)
  Skip a number of lines forward.

- Fl_Color spelling_underline_color () const
  Gets the underline color for style attribute ATTR_SPELLING.

- void spelling_underline_color (Fl_Color color)
  Sets the underline color for style attribute ATTR_SPELLING.

- Fl_Text_Buffer *style_buffer () const
  Gets the current style buffer associated with the text widget.

- Fl_Color textcolor () const
  Gets the default color of text in the widget.

- void textcolor (Fl_Color n)
  Sets the default color of text in the widget.

- Fl_Font textfont () const
  Gets the default font used when drawing text in the widget.

- void textfont (Fl_Font s)
  Sets the default font used when drawing text in the widget.

- Fl_Fontsize textsize () const
  Gets the default size of text in the widget.

- void textsize (Fl_Fontsize s)
  Sets the default size of text in the widget.

- int word_end (int pos) const
  Moves the insert position to the end of the current word.

- int word_start (int pos) const
  Moves the insert position to the beginning of the current word.
• void **wrap_mode** (int wrap, int wrap_margin)
   
   Set the new text wrap mode.

• int **wrapped_column** (int row, int column) const

   Nobody knows what this function does.

• int **wrapped_row** (int row) const

   Nobody knows what this function does.

• double **x_to_col** (double x) const

   Convert an x pixel position into a column number.

• ~**Fl_Text_Display** ()

   Free a text display and release its associated memory.

**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 start_pos_is_line_start, int *lineEnd, int *nextLineStart) const

   Finds both the end of the current line and the start of the next line.

• void **find_wrap_range** (const char *deletedText, int pos, int nInserted, int nDeleted, int *modRangeStart, int *modRangeEnd, int *linesInserted, int *linesDeleted)

   Wrapping calculations.
- `int find_x` (const char *s, int len, int style, int x) const
  
  Find the index of the character that lies at the given x position / closest cursor position.

- `int get_absolute_top_line_number` () const
  
  Returns the absolute (non-wrapped) line number of the first line displayed.

- `int handle_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 wrap_used_character` (int lineEndPos) const
  
  Check if the line break is caused by a newline or by line wrapping.

- `void wrapped_line_counter` (FL_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool 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_Color grammar_underline_color_
- Fl_Align linenumber_align_
- Fl_Color linenumberbgcolor_
- Fl_Color linenumberbgcolor_
- Fl_Font linenumber_font_
- const char *linenumber_format_
- Fl_Fontsize linenumber_size_
- int mAbsTopLineNum
- Fl_Text_Buffer *mBuffer
- double mColumnScale
- int mContinuousWrap
- Fl_Color mCursor_color
- int mCursorOldY
- int mCursorOn
- int mCursorPos
- int mCursorPreferredXPos
- int mCursorStyle
- int mCursorToHint
- int mFirstChar
- void *mHighlightCBArg
- int mHorizOffset
- int mHorizOffsetHint
- Fl_Scrollbar *mHScrollBar
- int mLastChar
- int mLNumLeft
- int mLNumWidth
- int *mLineStarts
- int mMaxsize
- int mModifyingTabDistance
- int mNBufferLines
- int mNeedAbsTopLineNum
• int mNLinesDeleted
• int mNStyles
• int mNVisibleLines
• Fl_Text_Buffer * mStyleBuffer
• const Style_Table_Entry * mStyleTable
• int mSuppressResync
• int mTopLineNum
• int mTopLineNumHint
• Unfinished_Style_Cb mUnfinishedHighlightCB
• char mUnfinishedStyle
• Fl_Scrollbar * mVScrollBar
• int mWrapMarginPix
• Fl_Alt.align scrollbar_align_
• int scrollbar_width_
• Fl_Color secondary_selection_color_
• int shortcut_
• Fl_Color spelling_underline_color_

struct {
    int h
    int w
    int x
    int y
} text_area

• Fl_Color textcolor_
• Fl_Font textfont_
• Fl_Fontsize textsize_

Friends
• void fl_text_drag_me (int pos, Fl_Text_Display *d)

Additional Inherited Members

32.133.1 Detailed Description

Rich text display widget. This is the FLTK text display widget. It allows the user to view multiple lines of text and supports highlighting, word wrap, mixes of font faces and colors, line numbers and scrolling. The buffer that is displayed in the widget is managed by the Fl_Text_Buffer class. A single Text Buffer can be displayed by multiple Text Displays.
Example Use

```cpp
#include <FL/FL_Text_Display.H>

int main() {
  FL_Text_Buffer *buff = new FL_Text_Buffer();
  FL_Text_Display *disp = new FL_Text_Display(10, 10, 640, 480);
  disp->buffer(buff); // attach text buffer to display widget
  buff->text("line one
line two"); // add some text to buffer
}
```

Features

- Word wrap: `wrap_mode()`, `wrapped_column()`, `wrapped_row()`
- Font control: `textfont()`, `textsize()`, `textcolor()`
• Font styling: highlight_data()

• Cursor: cursor_style(), show_cursor(), hide_cursor(), cursor_color()

• Line numbers: linenumber_width(), linenumber_font(), linenumber_size(), linenumber_fgcolor(), linenumber_bgcolor(), 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)

32.133.2 Member Enumeration Documentation

32.133.2.1 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>unfille box under the current character</td>
</tr>
<tr>
<td>HEAVY_CURSOR</td>
<td>thick I-beam</td>
</tr>
<tr>
<td>SIMPLE_CURSOR</td>
<td>as cursor as Fl_Input cursor</td>
</tr>
</tbody>
</table>

32.133.2.2 anonymous enum

anonymous enum
wrap types - used in wrap_mode()

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</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>

32.133.2.3 anonymous enum

anonymous enum
attribute flags in Style_Table_Entry.attr

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_BGCOLOR</td>
<td>use the background color in the bgcolor field</td>
</tr>
</tbody>
</table>

Generated by Doxygen
32.133.3 Constructor & Destructor Documentation

32.133.3.1 Fl_Text_Display()

Fl_Text_Display::Fl_Text_Display ( 
    int X,
    int Y,
    int W,
    int H,
    const char ∗ l = 0 )

Creates a new text display widget.

Parameters

| X, Y, W, H | position and size of widget |
| l          | label text, defaults to none |

32.133.3.2 ~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)

32.133.4 Member Function Documentation

32.133.4.1 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.
32.133.4.2 buffer() [1/3]

`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)`

32.133.4.3 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)`

32.133.4.4 buffer() [3/3]

`void Fl_Text_Display::buffer ( Fl_Text_Buffer * buf )`

Attach a text buffer to display, replacing the current buffer (if any).
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 |

32.133.4.5 buffer_modified_cb()

`void Fl_Text_Display::buffer_modified_cb ( int pcs, int nInserted, int nDeleted, int nRestyled, const char * deletedText, void * cbArg ) [static], [protected]`
32.133 Fl_Text_Display Class Reference

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>

32.133.4.6 buffer_predelete_cb()

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>

32.133.4.7 calc_last_char()

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.

32.133.4.8 calc_line_starts()

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.

Parameters

| startLine,endLine | range of lines to scan as line numbers |
### 32.133.4.9 clear_rect()

```c
void Fl_Text_Display::clear_rect (
    int style,
    int X,
    int Y,
    int width,
    int height ) const [protected]
```

Clear a rectangle with the appropriate background color for `style`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>style</code></td>
<td>index into style table</td>
</tr>
<tr>
<td><code>X,Y,width,height</code></td>
<td>size and position of background area</td>
</tr>
</tbody>
</table>

### 32.133.4.10 col_to_x()

```c
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><code>col</code></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

**See also**

`x_to_col()`

### 32.133.4.11 count_lines()

```c
int Fl_Text_Display::count_lines ( int startPos,
                                   int endPos,
                                   bool startPosIsLineStart ) const
```

Count the number of lines between two positions.

Same as `Fl_Text_Buffer::count_lines()`, but takes into account wrapping if wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startPos</code></td>
<td>index to first character</td>
</tr>
<tr>
<td><code>endPos</code></td>
<td>index after last character</td>
</tr>
<tr>
<td><code>startPosIsLineStart</code></td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>
Returns
number of lines

32.133.4.12  cursor_color() [1/2]

Fl_Color Fl_Text_Display::cursor_color ( ) const [inline]
Gets the text cursor color.
Returns
cursor color

32.133.4.13  cursor_color() [2/2]

void Fl_Text_Display::cursor_color ( Fl_Color n ) [inline]
Sets the text cursor color.
Parameters

| n | new cursor color |

32.133.4.14  cursor_style()

void Fl_Text_Display::cursor_style ( int style )
Sets the text cursor style.
Sets the text cursor style to one of the following:

- Fl_Text_Display::NORMAL_CURSOR - Shows an I beam.
- Fl_Text_Display::CARET_CURSOR - Shows a caret under the text.
- Fl_Text_Display::DIM_CURSOR - Shows a dimmed I beam.
- Fl_Text_Display::BLOCK_CURSOR - Shows an unfilled box around the current character.
- Fl_Text_Display::HEAVY_CURSOR - Shows a thick I beam.

This call also switches the cursor on and may trigger a redraw.
Parameters

| style | new cursor style |

32.133.4.15  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?
32.133.4.16  draw()

```cpp
void Fl_Text_Display::draw ( void ) [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.

32.133.4.17  draw_cursor()

```cpp
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>X, Y</th>
<th>cursor position in pixels</th>
</tr>
</thead>
</table>

32.133.4.18  draw_line_numbers()

```cpp
void Fl_Text_Display::draw_line_numbers ( bool clearAll ) [protected]
```

Refresh the line number area.

**Parameters**

<table>
<thead>
<tr>
<th>clearAll</th>
<th>– (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.</th>
</tr>
</thead>
</table>

32.133.4.19  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>startpos</th>
<th>index of first character to draw</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpos</td>
<td>index after last character to draw</td>
</tr>
</tbody>
</table>
32.133.4.20  draw_string()

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, # 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>

32.133.4.21  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

| left, top | are in coordinates of the text drawing window. |
| width, height | size in pixels |

32.133.4.22  draw_vline()

void Fl_Text_Display::draw_vline (  
  int visLineNum,  
  int leftClip,  
  int rightClip,  
  int leftCharIndex,  
  int rightCharIndex ) [protected]

Draw a single line of text. Draw the text on a single line represented by visLineNum (the number of lines down from the top of the display), limited by leftClip and rightClip window coordinates and leftCharIndex and rightCharIndex character positions (not including the character at position rightCharIndex).

Parameters

| visLineNum | index of line in the visible line number lookup |
| leftClip, rightClip | pixel position of clipped area |
Parameters

leftCharIndex, rightCharIndex  index into line of segment that we want to draw

32.133.4.23  empty_vlines()

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

32.133.4.24  extend_range_for_styles()

void Fl_Text_Display::extend_range_for_styles (  
    int * startPos,
    int * endpos  )  [protected]
I don't know what this does!
Extend the range of a redraw request (from *start to *end) with additional redraw requests resulting from changes to the attached style buffer (which contains auxiliary information for coloring or styling text).

Parameters

startpos ??
endpos ??

Todo  Unicode?

32.133.4.25  find_line_end()

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 lineEnd</td>
</tr>
<tr>
<td>out nextLineStart</td>
</tr>
</tbody>
</table>
### 32.133.4.26 find_wrap_range()

```c
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 re-display.

**Parameters**

| deletedText | const char * deletedText |
| pos         | int pos |
|Inserted     | int nInserted |
|Deleted      | int nDeleted |
|ModRangeStart| int * modRangeStart |
|ModRangeEnd  | int * modRangeEnd |
|LinesInserted| int * linesInserted |
|LinesDeleted | int * linesDeleted |

### 32.133.4.27 find_x()

```c
int Fl_Text_Display::find_x (  
    const char * s,  
    int len,  
    int style,  
    int x ) const [protected]
```

Find the index of the character that lies at the given x position / closest cursor position.

**Parameters**

| s       | UTF-8 text string |
| len     | length of string |
| style   | index into style lookup table |
| x       | position in pixels - negative returns closest cursor position |

**Returns**

index into buffer

### 32.133.4.28 get_absolute_top_line_number()

```c
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.
32.133.4.29  grammar_underline_color() [1/2]

`Fl_Color F1_Text_Display::grammar_underline_color () const [inline]`

Gets the underline color for style attribute ATTR_GRAMMAR.

Returns

underline color

32.133.4.30  grammar_underline_color() [2/2]

`void F1_Text_Display::grammar_underline_color ( Fl_Color color ) [inline]`

Sets the underline color for style attribute ATTR_GRAMMAR.

Parameters

- **color** underline color

32.133.4.31  handle_vline()

`int F1_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></th>
<th>mode</th>
<th>DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>lineStartPos</td>
<td>index of first character</td>
</tr>
<tr>
<td>in</td>
<td>lineLen</td>
<td>size of string in bytes</td>
</tr>
<tr>
<td>in</td>
<td>leftChar,rightChar</td>
<td>drawing position</td>
</tr>
<tr>
<td>in</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>bottomClip,leftClip,rightClip</td>
<td>stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>DRAW_LINE</th>
<th>index of last drawn character</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_WIDTH</td>
<td>width in pixels of text segment if we would draw it</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>FIND_INDEX</th>
<th>index of character at given x position in window coordinates</th>
</tr>
</thead>
<tbody>
<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

### 32.133.4.32 highlight_data()

```cpp
void Fl_Text_Display::highlight_data (  
    Fl_Text_Buffer *styleBuffer,  
    const Style_Table_Entry *styleTable,  
    int nStyles,  
    char unfinishedStyle,  
    Unfinished_Style_Cb unfinishedHighlightCB,  
    void *cbArg )
```

Attach (or remove) highlight information in text display and redisplay.

Highlighting information consists of a style buffer which parallels the normal text buffer, but codes font and color information for the display; a style table which translates style buffer codes (indexed by buffer character - 'A') into fonts and colors; and a callback mechanism for as-needed highlighting, triggered by a style buffer entry of "unfinished←Style". Style buffer can trigger additional redisplay during a normal buffer modification if the buffer contains a primary Fl_Text_Selection (see extendRangeForStyleMods for more information on this protocol). Style buffers, tables and their associated memory are managed by the caller.

**Styles** are ranged from 65 ('A') to 126.

**Note**

Style information in the style buffer must have the same byte offset as the corresponding character in the text buffer. UTF-8 characters can have a maximum length of four bytes. Style information must take this into account and fill the unused bytes with 0. See `fl_utf8len()`.

**Text:** "*g r ü n*", where normal style is 'A', and bold is 'B'

- **Text Buffer(hex):** 67 72 c3 bc 6e : gr..n
- **Style Buffer(hex):** 42 41 41 00 42 : BAA.B

#### Parameters

| styleBuffer | 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. |
| styleTable | a list of styles indexed by the style buffer |
| nStyles | number of styles in the style table |
| unfinishedStyle | if this style is found, the callback below is called |
| unfinishedHighlightCB | if a character with an unfinished style is found, this callback will be called |
| cbArg | an optional argument for the callback above, usually a pointer to the Text Display. |

**Todo** "extendRangeForStyleMods" does not exist (might be a hangover from the port from nedit). Find the correct function.

**See also**

- `Fl_Text_Display::style_buffer()`
32.133.4.33 in_selection()

```cpp
text Fl_Text_Display::in_selection (  
    int X,  
    int Y ) const  
```

Check if a pixel position is within the primary selection.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

**Returns**

1 if position (X, Y) is inside of the primary Fl_Text_Selection

32.133.4.34 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**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
</tr>
</tbody>
</table>

new text in UTF-8 encoding.

32.133.4.35 insert_position() [1/2]

```cpp
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)

32.133.4.36 insert_position() [2/2]

```cpp
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**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>newPos</td>
</tr>
</tbody>
</table>

new caret position
32.133.4.37 line_end()

```cpp
int Fl_Text_Display::line_end (  
    int startPos,  
    bool startPosIsLineStart ) const
```

Returns the end of a line.
Same as buffer()-line_end(startPos), but takes into account line breaks when wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Note that the definition of the end of a line is less clear when continuous wrap is on. With continuous wrap off, it's just a pointer to the newline that ends the line. When it's on, it's the character beyond the last displayable character on the line, where a whitespace character which has been "converted" to a newline for wrapping is not considered displayable. Also note that a line can be wrapped at a non-whitespace character if the line had no whitespace. In this case, this routine returns a pointer to the start of the next line. This is also consistent with the model used by visLineLength.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
</table>

| startPosIsLineStart | avoid scanning back to the line start |

Returns

new position as index

32.133.4.38 line_start()

```cpp
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

32.133.4.39 linenumber_align()

```cpp
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

32.133.4.40 linenumberbgcolor()

```cpp
void Fl_Text_Display::linenumberbgcolor (  
    Fl_Color val )
```

Set the background color used for line numbers (if enabled).
32.133.4.41  linenumber_fgcolor()

void Fl_Text_Display::linenumber_fgcolor (  
    Fl_Color val  
)  
Set the foreground color used for line numbers (if enabled).

Version  
1.3.3

32.133.4.42  linenumber_font()

void Fl_Text_Display::linenumber_font (  
    Fl_Font val  
)  
Set the font used for line numbers (if enabled).

Version  
1.3.3

32.133.4.43  linenumber_format()

void Fl_Text_Display::linenumber_format (  
    const char ∗ val  
)  
Sets the printf() style format string used for line numbers.  
Default is "%d" for normal unpadded decimal integers.  
An internal copy of val is allocated and managed; it is automatically freed whenever a new value is assigned, or when the widget is destroyed.  
The value of val must not be NULL.  
Example values:  
- "%d" -- For normal line numbers without padding (Default)  
- "%03d" -- For 000 padding  
- "%x" -- For hexadecimal line numbers  
- "%o" -- For octal line numbers

Version  
1.3.3

32.133.4.44  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
32.133.45 linenumber_width()

```cpp
void Fl_Text_Display::linenumber_width ( 
    int width )
```

Set width of screen area for line numbers.
Use to also enable/disable line numbers. A value of 0 disables line numbering, values >0 enable the line number display.

**Parameters**

| width | The new width of the area for line numbers to appear, in pixels. 0 disables line numbers (default) |

32.133.46 longest_vline()

```cpp
int Fl_Text_Display::longest_vline ( ) const [protected]
```

Find the longest line of all visible lines.

**Returns**

the width of the longest visible line in pixels

32.133.47 maintain_absolute_top_line_number()

```cpp
void Fl_Text_Display::maintain_absolute_top_line_number ( 
    int state ) [protected]
```

Line numbering stuff, currently unused.
In continuous wrap mode, internal line numbers are calculated after wrapping. A separate non-wrapped line count is maintained when line numbering is turned on. There is some performance cost to maintaining this line count, so normally absolute line numbers are not tracked if line numbering is off. This routine allows callers to specify that they still want this line count maintained (for use via TextDPosToLineAndCol). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

**Todo** TextDPosToLineAndCol does not exist (nedit port?)

32.133.48 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).

32.133.49 measure_deleted_lines()

```cpp
void Fl_Text_Display::measure_deleted_lines ( 
    int pos, 
    int nDeleted ) [protected]
```

Wrapping calculations.
This is a stripped-down version of the findWrapRange() function above, intended to be used to calculate the number of "deleted" lines during a buffer modification. It is called before the modification takes place.
This function should only be called in continuous wrap mode with a non-fixed font width. In that case, it is impossible to calculate the number of deleted lines, because the necessary style information is no longer available after the modification. In other cases, we can still perform the calculation afterwards (possibly even more efficiently).
**Parameters**

| pos | nDeleted |

### 32.133.4.50 measure_proportional_character()

```cpp
double Fl_Text_Display::measure_proportional_character (const char * s, int xPix, int pos) const [protected]
```

Wrapping calculations.

Measure the width in pixels of the first character of string "s" at a particular column "colNum" and buffer position "pos". This is for measuring characters in proportional or mixed-width highlighting fonts.

A note about proportional and mixed-width fonts: the mixed width and proportional font code in nedit does not get much use in general editing, because nedit doesn't allow per-language-mode fonts, and editing programs in a proportional font is usually a bad idea, so very few users would choose a proportional font as a default. There are still probably mixed-width syntax highlighting cases where things don't redraw properly for insertion/deletion, though static display and wrapping and resizing should now be solid because they are now used for online help display.

**Parameters**

| s | text string |
| xPix | x pixel position needed for calculating tab widths |
| pos | offset within string |

**Returns**

width of character in pixels

### 32.133.4.51 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

### 32.133.4.52 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
32.133.4.53  move_left()

int Fl_Text_Display::move_left ( )
Moves the current insert position left one character.

Returns

1 if the cursor moved, 0 if the beginning of the text was reached

32.133.4.54  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

32.133.4.55  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

32.133.4.56  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 |

32.133.4.57  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.
32.133.4.58  position_style()

```cpp
int Fl_Text_Display::position_style (
    int lineStartPos,
    int lineLen,
    int lineIndex ) const
```

Find the correct style for a character.
Determine the drawing method to use to draw a specific character from “buf”.
`lineStartPos` gives the character index where the line begins, `lineIndex`, the number of characters past
the beginning of the line, and `lineLen` the number of displayed characters past the beginning of the line. Passing
`lineStartPos` of -1 returns the drawing style for “no text”.
Why not just: `position_style(pos)`? Because style applies to blank areas of the window beyond the text boundaries,
and because this routine must also decide whether a position is inside of a rectangular Fl_Text_Selection, and do
so efficiently, without re-counting character positions from the start of the line.
Note that style is a somewhat incorrect name, drawing method would be more appropriate.
If `lineIndex` is pointing to the last character in a line, and the second to last character has the ATTR_BGCOLOR_EXT
set, the background color will extend into the remaining line.

**Parameters**

- `lineStartPos`  beginning of this line
- `lineLen`  number of bytes in line
- `lineIndex`  position of character within line

**Returns**

- style for the given character

32.133.4.59  position_to_line()

```cpp
int Fl_Text_Display::position_to_line (
    int pos,
    int * lineNum ) const [protected]
```

Convert a position index into a line number offset.
Find the line number of position `pos` relative to the first line of displayed text, counting from 0 to `visible lines - 1`.
The line number is returned in `lineNum`.
Returns 0 if the line is not displayed. In this case `lineNum` is 0 as well.
Returns 1 if the line is displayed. In this case `lineNum` is the relative line number.

**Parameters**

- `in pos`  byte position in buffer
- `out lineNum`  relative line number of byte `pos` in buffer

**Returns**

- whether the character at byte position `pos` is currently displayed

**Return values**

- 0  `pos` is not displayed; `lineNum` is invalid (zero)
- 1  `pos` is displayed; `lineNum` is valid
32.133.4.60  position_to_linecol()

int Fl_Text_Display::position_to_linecol (  
  int pos,  
  int * lineNum,  
  int * column) const [protected]

Find the line and column number of position pos.
This only works for displayed lines. If the line is not displayed, the function returns 0 (without the mLineStarts array it could turn in to very long calculation involving scanning large amounts of text in the buffer). If continuous wrap mode is on, returns the absolute line number (as opposed to the wrapped line number which is used for scrolling).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>character index</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>lineNum</td>
</tr>
<tr>
<td></td>
<td>absolute (unwrapped) line number</td>
</tr>
<tr>
<td>out</td>
<td>column</td>
</tr>
<tr>
<td></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.

32.133.4.61  position_to_xy()

int Fl_Text_Display::position_to_xy (  
  int pos,  
  int * X,  
  int * Y) const

Convert a character index into a pixel position.
Translate a buffer text position to the XY location where the top left of the cursor would be positioned to point to that character. Returns 0 if the position is not displayed because it is vertically out of view. If the position is horizontally out of view, returns the X coordinate where the position would be if it were visible.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>character index</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X,Y</td>
</tr>
<tr>
<td></td>
<td>pixel position of character on screen</td>
</tr>
</tbody>
</table>

Returns

0 if character vertically out of view, X & Y positions otherwise

32.133.4.62  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>
32.133.4.63 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.

32.133.4.64 resize()

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.

32.133.4.65 rewind_lines()

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

| startPos | index to starting character |
| nLines   | number of lines to skip back |

Returns

new position as index

32.133.4.66 scroll()

void Fl_Text_Display::scroll ( 
  int topLineNum, 
  int horizOffset )
Scrolls the current buffer to start at the specified line and column.

Parameters

| topLineNum | top line number |
| horizOffset | column number |
Todo  Column numbers make little sense here.

32.133.4.67 scroll()

`int Fl_Text_Display::scroll_ (int topLineNum, int horizOffset) [protected]`

Scrolls the current buffer to start at the specified line and column.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>topLineNum</code></td>
<td>top line number</td>
</tr>
<tr>
<td><code>horizOffset</code></td>
<td>in pixels</td>
</tr>
</tbody>
</table>

Returns

0 if nothing changed, 1 if we scrolled

32.133.4.68 scroll_timer_cb()

`void Fl_Text_Display::scroll_timer_cb (void *user_data) [static], [protected]`

Timer callback for scroll events.
This timer event scrolls the text view proportionally to how far the mouse pointer has left the text area. This allows for smooth scrolling without "wiggeling" the mouse.

32.133.4.69 scrollbar_align() [1/2]

`Fl_Align Fl_Text_Display::scrollbar_align () const [inline]`

Gets the scrollbar alignment type.

Returns

scrollbar alignment

32.133.4.70 scrollbar_align() [2/2]

`void Fl_Text_Display::scrollbar_align (Fl_Align a) [inline]`

Sets the scrollbar alignment type.

Parameters

| `a` | new scrollbar alignment |

32.133.4.71 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)`

### 32.133.4.72 scrollbar_size() [2/2]

```cpp
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()`

### 32.133.4.73 scrollbar_width() [1/2]

```cpp
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.

### 32.133.4.74 scrollbar_width() [2/2]

```cpp
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

### 32.133.4.75 secondary_selection_color() [1/2]

```cpp
Fl_Color Fl_Text_Display::secondary_selection_color ( ) const [inline]
```

Gets the background color for the secondary selection block.

Returns

background color color
32.133.4.76  secondary_selection_color() [2/2]

void Fl_Text_Display::secondary_selection_color (Fl_Color color) [inline]

Sets the background color for the secondary selection block.

Parameters

| color | background color |

32.133.4.77  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

32.133.4.78  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 |

32.133.4.79  show_cursor()

void Fl_Text_Display::show_cursor (int b = 1)

Shows the text cursor.
This function may trigger a redraw.

Parameters

| b | show(1) or hide(0) the text cursor (caret) |

32.133.4.80  show_insert_position()

void Fl_Text_Display::show_insert_position ()

Scrolls the text buffer to show the current insert position.
This function triggers a complete recalculation, ending in a call to Fl_Text_Display::display_insert()
skip_lines()

```c
int Fl_Text_Display::skip_lines (  
    int startPos,  
    int nLines,  
    bool startPosIsLineStart )
```

Skip a number of lines forward.
Same as Fl_Text_Buffer::skip_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLines</td>
<td>number of lines to skip ahead</td>
</tr>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

**Returns**

new position as index

spelling_underline_color() [1/2]

```c
Fl_Color Fl_Text_Display::spelling_underline_color ( ) const [inline]
```

Gets the underline color for style attribute ATTR_SPELLING.

**Returns**

underline color

spelling_underline_color() [2/2]

```c
void Fl_Text_Display::spelling_underline_color (  
    Fl_Color color ) [inline]
```

Sets the underline color for style attribute ATTR_SPELLING.

**Parameters**

| color | underline color |

string_width()

```c
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>
32.133.4.85 style_buffer()

Fl_Text_Buffer* Fl_Text_Display::style_buffer () const [inline]

Gets the current style buffer associated with the text widget.
Multiple text widgets can be associated with the same style buffer.

Returns

current style buffer

See also

Fl_Text_Display::highlight_data()

32.133.4.86 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

32.133.4.87 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

32.133.4.88 textfont() [1/2]

Fl_Font Fl_Text_Display::textfont () const [inline]

Gets the default font used when drawing text in the widget.

Returns

current text font face unless overridden by a style

32.133.4.89 textfont() [2/2]

void Fl_Text_Display::textfont ( Fl_Font s ) [inline]

Sets the default font used when drawing text in the widget.
Parameters

$s$ | default text font face

### 32.133.4.90 `textsize()` [1/2]

```cpp
Fl_Fontsize Fl_Text_Display::textsize () const [inline]
```

Gets the default size of text in the widget.

**Returns**

Current text height unless overridden by a style.

### 32.133.4.91 `textsize()` [2/2]

```cpp
void Fl_Text_Display::textsize ( Fl_Fontsize s ) [inline]
```

Sets the default size of text in the widget.

**Parameters**

$s$ | new text size

### 32.133.4.92 `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.

### 32.133.4.93 `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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pos</code></td>
<td>index into buffer of recent changes</td>
</tr>
<tr>
<td><code>charsInserted</code></td>
<td>number of bytes(!) inserted</td>
</tr>
<tr>
<td><code>charsDeleted</code></td>
<td>number of bytes(!) deleted</td>
</tr>
<tr>
<td><code>linesInserted</code></td>
<td>number of lines</td>
</tr>
<tr>
<td><code>linesDeleted</code></td>
<td>number of lines</td>
</tr>
<tr>
<td><code>scrolled</code></td>
<td>set to 1 if the text display needs to be scrolled</td>
</tr>
</tbody>
</table>
32.133.4.94 update_v_scrollbar()

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.

32.133.4.95 vline_length()

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

the number of bytes in this line

32.133.4.96 word_end()

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

the index of the first character after the end of the word

32.133.4.97 word_start()

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 |
**wrap_mode()**

```cpp
void Fl_Text_Display::wrap_mode(
    int wrap,
    int wrapMargin)
```

Set the new text wrap mode.

If `wrap` mode is not zero, this call enables automatic word wrapping at column `wrapMargin`. Word-wrapping does not change the text buffer itself, only the way the text is displayed. Different Text Displays can have different wrap modes, even if they share the same Text Buffer.

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>

**wrap_uses_character()**

```cpp
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?)
32.133.4.100 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**

<table>
<thead>
<tr>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
</tr>
</tbody>
</table>

**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?

32.133.4.101 wrapped_line_counter()

```cpp
void Fl_Text_Display::wrapped_line_counter (  
    Fl_Text_Buffer ∗ buf,  
    int startPos,  
    int maxPos,  
    int maxLines,  
    bool startPosIsLineStart,  
    int styleBufOffset,  
    int ∗ retPos,  
    int ∗ retLines,  
    int ∗ retLineStart,  
    int ∗ retLineEnd,  
    bool countLastLineMissingNewLine = true ) const [protected]
```

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**

<table>
<thead>
<tr>
<th>in buf</th>
<th>The text buffer to operate on</th>
</tr>
</thead>
<tbody>
<tr>
<td>in startPos</td>
<td>Starting index position into the buffer</td>
</tr>
<tr>
<td>in maxPos</td>
<td>Maximum index position into the buffer we'll reach</td>
</tr>
<tr>
<td>in maxLines</td>
<td>Maximum number of lines we'll reach</td>
</tr>
<tr>
<td>in startPosIsLineStart</td>
<td>Flag indicating if startPos is start of line. (If set, prevents our having to find the line start)</td>
</tr>
<tr>
<td>in styleBufOffset</td>
<td>Offset index position into style buffer.</td>
</tr>
<tr>
<td>out retPos</td>
<td>Position where counting ended. When counting lines, the position returned is the start of the line &quot;maxLines&quot; lines beyond &quot;startPos&quot;.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
32.133.4.102  wrapped_row()

```cpp
to_int F1_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` number of pixels from the left margin

**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?)

32.133.4.103  x_to_col()

```cpp
double F1_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

32.133.4.104  xy_to_position()

```cpp
int F1_Text_Display::xy_to_position (  
    int X,  
    int Y,  
    int posType = CHARACTER_POS ) const [protected]
```

Translate a pixel position into a character index.

Translate window coordinates to the nearest (insert cursor or character cell) text position. The parameter `posType` specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest cursor position, and CHARACTER_POS means return the position of the character closest to (X, Y).
Parameters

<table>
<thead>
<tr>
<th>X,Y</th>
<th>pixel position</th>
</tr>
</thead>
<tbody>
<tr>
<td>posType</td>
<td>CURSOR_POS or CHARACTER_POS</td>
</tr>
</tbody>
</table>

Returns

index into text buffer

32.133.4.105  xy_to_rowcol()

```cpp
void Fl_Text_Display::xy_to_rowcol {
  int X,
  int Y,
  int * row,
  int * column,
  int posType = CHARACTER_POS ) const [protected]
```

Translate pixel coordinates into row and column.
Translate window coordinates to the nearest row and column number for positioning the cursor. This, of course, makes no sense when the font is proportional, since there are no absolute columns. The parameter posType specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest position between characters, and CHARACTER_POS means translate the position to the nearest character cell.

Parameters

<table>
<thead>
<tr>
<th>X,Y</th>
<th>pixel coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>row,column nearest row and column</td>
</tr>
<tr>
<td>posType</td>
<td>CURSOR_POS or CHARACTER_POS</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Text_Display.H
- Fl_Text_Display.cxx

32.134  Fl_Text_Editor Class Reference

This is the FLTK text editor widget.

#include <Fl_Text_Editor.H>

Inheritance diagram for Fl_Text_Editor:

- Fl_Widget
- Fl_Group
- Fl_Text_Display
- Fl_Text_Editor

Classes

- struct Key_Binding

Simple linked list item associating a key/state to a function.
Public Types

• typedef int (Key_Func) (int key, Fl_Text_Editor ∗editor)

  Key function binding callback type.

Public Member Functions

• void add_default_key_bindings (Key_Binding ∗∗list)

  Adds all of the default editor key bindings to the specified key binding list.

• void add_key_binding (int key, int state, Key_Func f)

  Adds a key of state state with the function f.

• void add_key_binding (int key, int state, Key_Func f, Key_Binding ∗∗list)

  Adds a key of state state with the function function to an arbitrary key binding list list.

• Key_Func bound_key_function (int key, int state) const

  Returns the function associated with a key binding.

• Key_Func bound_key_function (int key, int state, Key_Binding ∗list) const

  Returns the function associated with a key binding.

• void default_key_function (Key_Func f)

  Sets the default key function for unassigned keys.

• Fl_Text_Editor (int X, int Y, int W, int H, const char ∗l=0)

  The constructor creates a new text editor widget.

  • virtual int handle (int e)

    Event handling.

  • int insert_mode ()

    Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.

  • void insert_mode (int b)

    Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.

  • void remove_all_key_bindings ()

    Removes all of the key bindings associated with the text editor or list.

  • void remove_all_key_bindings (Key_Binding ∗∗list)

    Removes all of the key bindings associated with the text editor or list.

  • void remove_key_binding (int key, int state)

    Removes the key binding associated with the key "key" of state "state".

  • void remove_key_binding (int key, int state, Key_Binding ∗∗list)

    Removes the key binding associated with the key key of state state from the Key_Binding list list.

  • int tab_nav () const

    Check if Tab focus navigation is enabled.

  • void tab_nav (int val)

    Enables or disables Tab key focus navigation.

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'.

Generated by Doxygen
• static int kf_default (int c, Fl_Text_Editor *e)
  Inserts the text associated with key ‘c’ in editor ‘e’.
• static int kf_delete (int c, Fl_Text_Editor *e)
  Does a delete of selected text or the current character in the current buffer of editor ‘e’.
• static int kf_down (int c, Fl_Text_Editor *e)
  Moves the text cursor one line down for editor ‘e’.
• static int kf_end (int c, Fl_Text_Editor *e)
  Moves the text cursor to the end of the current line in editor ‘e’.
• static int kf_enter (int c, Fl_Text_Editor *e)
  Inserts a newline for key ‘c’ at the current cursor position in editor ‘e’.
• static int kf_home (int, Fl_Text_Editor *e)
  Moves the text cursor to the beginning of the current line in editor ‘e’.
• static int kf_ignore (int c, Fl_Text_Editor *e)
  Ignores the key ‘c’ in editor ‘e’.
• static int kf_insert (int c, Fl_Text_Editor *e)
  Toggles the insert mode for editor ‘e’.
• static int kf_left (int c, Fl_Text_Editor *e)
  Moves the text cursor one character to the left for editor ‘e’.
• static int kf_m_s_move (int c, Fl_Text_Editor *e)
  Extends the current selection in the direction indicated by meta key ‘c’ in editor ‘e’.
• static int kf_meta_move (int c, Fl_Text_Editor *e)
  Moves the current text cursor in the direction indicated by meta key ‘c’ in editor ‘e’.
• static int kf_move (int c, Fl_Text_Editor *e)
  Moves the text cursor in the direction indicated by key ‘c’ in editor ‘e’.
• static int kf_page_down (int c, Fl_Text_Editor *e)
  Moves the text cursor down one page for editor ‘e’.
• static int kf_page_up (int c, Fl_Text_Editor *e)
  Moves the text cursor up one page for editor ‘e’.
• static int kf_paste (int c, Fl_Text_Editor *e)
  Does a paste of selected text in the current buffer of editor ‘e’.
• static int kf_right (int c, Fl_Text_Editor *e)
  Moves the text cursor one character to the right for editor ‘e’.
• static int kf_select_all (int c, Fl_Text_Editor *e)
  Selects all text in the current buffer in editor ‘e’.
• static int kf_shift_move (int c, Fl_Text_Editor *e)
  Extends the current selection in the direction of key ‘c’ in editor ‘e’.
• static int kf_undo (int c, Fl_Text_Editor *e)
  Undo last edit in the current buffer of editor ‘e’.
• static int kf_up (int c, Fl_Text_Editor *e)
  Moves the text cursor one line up for editor ‘e’.

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**

**32.134.1 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.

**32.134.2 Member Function Documentation**

**32.134.2.1 add_key_binding()**

```cpp
void Fl_Text_Editor::add_key_binding (     
    int key,     
    int state,    
    Key_Func function,    
    Key_Binding **list )
```

Adds a key of state state with the function function to an arbitrary key binding list list. This can be used in derived classes to add global key bindings by using the global (static) `Key_Binding` list `Fl_Text_Editor::global_key_bindings`.

**32.134.2.2 insert_mode() [1/2]**

```cpp
int Fl_Text_Editor::insert_mode ( ) [inline]
```

Gets the current insert mode; if non-zero, new text is inserted before the current cursor position. Otherwise, new text replaces text at the current cursor position.

**32.134.2.3 insert_mode() [2/2]**

```cpp
void Fl_Text_Editor::insert_mode (     
    int b ) [inline]
```

Sets the current insert mode; if non-zero, new text is inserted before the current cursor position. Otherwise, new text replaces text at the current cursor position.

**32.134.2.4 kf_backspace()**

```cpp
int Fl_Text_Editor::kf_backspace (     
    int c,     
    Fl_Text_Editor * e ) [static]
```

Does a backspace for key 'c' in the current buffer of editor 'e'. Any current selection is deleted. Otherwise, the character left is deleted and the cursor moved. The key value 'c' is currently unused.

**32.134.2.5 kf_c_s_move()**

```cpp
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()`.
32.134.2.6 kf_copy()

int Fl_Text_Editor::kf_copy (  
    int c,  
    Fl_Text_Editor * e ) [static]

Does a copy of selected text or the current character in the current buffer of editor 'e'.  
The key value 'c' is currently unused.

32.134.2.7 kf_ctrl_move()

int Fl_Text_Editor::kf_ctrl_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.  
Supported values for 'c' are currently:

- FL_Home -- moves the cursor to the beginning of the document
- FL_End -- moves the cursor to the end of the document
- FL_Left -- moves the cursor left one word
- FL_Right -- moves the cursor right one word
- FL_Up -- scrolls up one line, without moving cursor
- FL_Down -- scrolls down one line, without moving cursor
- FL_Page_Up -- moves the cursor to the beginning of the top line on the current page
- FL_Page_Down -- moves the cursor to the beginning of the last line on the current page

32.134.2.8 kf_cut()

int Fl_Text_Editor::kf_cut (  
    int c,  
    Fl_Text_Editor * e ) [static]

Does a cut of selected text in the current buffer of editor 'e'.  
The key value 'c' is currently unused.

32.134.2.9 kf_default()

int Fl_Text_Editor::kf_default (  
    int c,  
    Fl_Text_Editor * e ) [static]

Inserts the text associated with key 'c' in editor 'e'.  
Honors the current selection and insert/overstrike mode.

32.134.2.10 kf_delete()

int Fl_Text_Editor::kf_delete (  
    int c,  
    Fl_Text_Editor * e ) [static]

Does a delete of selected text or the current character in the current buffer of editor 'e'.  
The key value 'c' is currently unused.

32.134.2.11 kf_down()

int Fl_Text_Editor::kf_down (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one line down for editor 'e'.  
Same as kf_move(FL_Down, e). The key value 'c' is currently unused.

32.134.2.12 kf_end()

int Fl_Text_Editor::kf_end (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the text cursor to the end of the current line in editor 'e'.
Same as kf_move(FL_End, e). The key value 'c' is currently unused.

32.134.2.13  kf_enter()

int Fl_Text_Editor::kf_enter (  
    int c,  
    Fl_Text_Editor * e ) [static]

Inserts a newline for key 'c' at the current cursor position in editor 'e'.
The key value 'c' is currently unused.

32.134.2.14  kf_home()

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.

32.134.2.15  kf_ignore()

int Fl_Text_Editor::kf_ignore (  
    int c,  
    Fl_Text_Editor * e ) [static]

Ignores the key 'c' in editor 'e'.
This method can be used as a keyboard binding to disable a key that might otherwise be handled or entered as text.
An example would be disabling FL_Escape, so that it isn't added to the buffer when invoked by the user.

32.134.2.16  kf_insert()

int Fl_Text_Editor::kf_insert (  
    int c,  
    Fl_Text_Editor * e ) [static]

Toggles the insert mode for editor 'e'.
The key value 'c' is currently unused.

32.134.2.17  kf_left()

int Fl_Text_Editor::kf_left (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one character to the left in editor 'e'.
Same as kf_move(FL_Left, e). The key value 'c' is currently unused.

32.134.2.18  kf_m_s_move()

int Fl_Text_Editor::kf_m_s_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction indicated by meta key 'c' in editor 'e'.
See also
    kf_meta_move().
32.134.2.19  kf_meta_move()

int Fl_Text_Editor::kf_meta_move (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the current text cursor in the direction indicated by meta key 'c' in editor 'e'.  
Supported values for 'c' are currently:  
FL_Up   -- moves cursor to the beginning of the current document  
FL_Down -- moves cursor to the end of the current document  
FL_Left -- moves the cursor to the beginning of the current line  
FL_Right -- moves the cursor to the end of the current line

32.134.2.20  kf_move()

int Fl_Text_Editor::kf_move (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the text cursor in the direction indicated by key 'c' in editor 'e'.  
Supported values for 'c' are currently:  
FL_Home   -- moves the cursor to the beginning of the current line  
FL_End    -- moves the cursor to the end of the current line  
FL_Left   -- moves the cursor left one character  
FL_Right  -- moves the cursor right one character  
FL_Up     -- moves the cursor up one line  
FL_Down   -- moves the cursor down one line  
FL_Page_Up -- moves the cursor up one page  
FL_Page_Down -- moves the cursor down one page

32.134.2.21  kf_page_down()

int Fl_Text_Editor::kf_page_down (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the text cursor down one page for editor 'e'.  
Same as kf_move(FL_Page_Down, e). The key value 'c' is currently unused.

32.134.2.22  kf_page_up()

int Fl_Text_Editor::kf_page_up (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the text cursor up one page for editor 'e'.  
Same as kf_move(FL_Page_Up, e). The key value 'c' is currently unused.

32.134.2.23  kf_paste()

int Fl_Text_Editor::kf_paste (  
    int c,  
    Fl_Text_Editor * e ) [static]
Does a paste of selected text in the current buffer of editor 'e'.  
Any current selection is replaced with the pasted content. The key value 'c' is currently unused.

32.134.2.24  kf_right()

int Fl_Text_Editor::kf_right (  
    int c,  
    Fl_Text_Editor * e ) [static]
Moves the text cursor one character to the right for editor 'e'.  
Same as kf_move(FL_Right, e). The key value 'c' is currently unused.
32.134.25  kf_select_all()

int Fl_Text_Editor::kf_select_all (  
    int c,  
    Fl_Text_Editor * e ) [static]

Selects all text in the current buffer in editor 'e'.
The key value 'c' is currently unused.

32.134.26  kf_shift_move()

int Fl_Text_Editor::kf_shift_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction of key 'c' in editor 'e'.
See also
    kf_move()

32.134.27  kf_undo()

int Fl_Text_Editor::kf_undo (  
    int c,  
    Fl_Text_Editor * e ) [static]

Undo last edit in the current buffer of editor 'e'.
Also deselects previous selection. The key value 'c' is currently unused.

32.134.28  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.

32.134.29  remove_key_binding()

void Fl_Text_Editor::remove_key_binding (  
    int key,  
    int state,  
    Key_Binding ** list )

Removes the key binding associated with the key key of state state from the Key_Binding list list.
This can be used in derived classes to remove global key bindings by using the global (static) Key_Binding list
Fl_Text_Editor::global_key_bindings.

32.134.30  tab_nav() [1/2]

int Fl_Text_Editor::tab_nav ( ) const

Check if Tab focus navigation is enabled.
If disabled (default), hitting Tab inserts a tab character into the editor buffer.
If enabled, hitting Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget.

Returns

<table>
<thead>
<tr>
<th>Return values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tab inserts tab characters (default)</td>
</tr>
<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

32.134.2.31 tab_nav() [2/2]

void Fl_Text_Editor::tab_nav (  
    int val  
)  

Enables or disables Tab key focus navigation.  
When disabled (default), tab characters are inserted into Fl_Text_Editor. Only the mouse can change focus. This  
behavior is desirable when Fl_Text_Editor is used, e.g. in a source code editor.  
When enabled, Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget. This  
behavior is desirable when Fl_Text_Editor is used e.g. in a database input form.  
Currently, this method is implemented as a convenience method that adjusts the key bindings for the Tab key. This  
implementation detail may change in the future. Know that changing the editor's key bindings for Tab and Shift-Tab  
may affect tab navigation.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>val</td>
</tr>
</tbody>
</table>

If val is 0, Tab inserts a tab character (default).  
If val is 1, Tab navigates widget focus.

See also

    tab_nav(), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

32.134.3 Member Data Documentation

32.134.3.1 global_key_bindings

Key_Binding∗ Fl_Text_Editor::global_keyBindings  [static],  [protected]

Global key binding list.
Derived classes can add key bindings for all Fl_Text_Editor widgets by adding a Key_Binding to this list.

See also

    add_key_binding(int key, int state, Key_Func f, Key_Binding∗∗ list);

The documentation for this class was generated from the following files:

- Fl_Text_Editor.H
- Fl_Text_Editor.cxx

32.135 Fl_Text_Selection Class Reference

This is an internal class for Fl_Text_Buffer to manage text selections.

#include <Fl_Text_Buffer.H>

Generated by Doxygen
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

32.135.1 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):

```
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

```
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.
32.135.2 Member Function Documentation

32.135.2.1 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.

32.135.2.2 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.

32.135.2.3 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

32.135.2.4 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startpos</code></td>
<td>return byte offset to first selected character</td>
</tr>
<tr>
<td><code>endpos</code></td>
<td>return byte offset pointing after last selected character</td>
</tr>
</tbody>
</table>
Returns
whether the selection is active (selected()) or not

Return values

<table>
<thead>
<tr>
<th></th>
<th>if not selected</th>
<th>if selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

See also
selected(), start(), end()

32.135.2.5 selected() [1/2]

```cpp
bool Fl_Text_Selection::selected ( ) const [inline]
```
Returns true if any text is selected.

Returns
true if any text has been selected, or false if no text is selected.

32.135.2.6 selected() [2/2]

```cpp
void Fl_Text_Selection::selected ( bool b ) [inline]
```
Modifies the 'selected' flag.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>new flag</td>
</tr>
</tbody>
</table>

32.135.2.7 set()

```cpp
void Fl_Text_Selection::set ( int startpos,
   int endpos )
```
Sets the selection range.
startpos and endpos must be at a character boundary.
If startpos != endpos selected() is set to true, else to false.
If startpos is greater than endpos they are swapped so that startpos <= endpos.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startpos</td>
</tr>
<tr>
<td>in</td>
<td>endpos</td>
</tr>
</tbody>
</table>

32.135.2.8 start()

```cpp
int Fl_Text_Selection::start ( ) const [inline]
```
Returns the byte offset to the first selected character.
The returned offset is only valid if `selected()` returns true. If the selection is not valid the returned offset is 0 since FLTK 1.4.0.

**Note**
In FLTK 1.3.x the returned offset could be non-zero even if `selected()` would have returned 0.

**Returns**
byte offset or 0 if not selected.

### 32.135.2.9 update()

```cpp
void Fl_Text_Selection::update (int pos, int nDeleted, int nInserted)
```

Updates a selection after text was modified.
Updates an individual selection for changes in the corresponding text.

**Parameters**

<table>
<thead>
<tr>
<th>pos</th>
<th>byte offset into text buffer at which the change occurred</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeleted</td>
<td>number of bytes deleted from the buffer</td>
</tr>
<tr>
<td>nInserted</td>
<td>number of bytes inserted into the buffer</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- `Fl_Text_Buffer.H`
- `Fl_Text_Buffer.cxx`

### 32.136 Fl_Tile Class Reference

The `Fl_Tile` class lets you resize its children by dragging the border between them.

Inheritance diagram for Fl_Tile:

```
Fl_Widget
    ↓
Fl_Group
    ↓
Fl_Tile
```

**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

32.136.1 Detailed Description

The Fl_Tile class lets you resize its children by dragging the border between them.

![Figure 32.51 Fl_Tile](image)

For the tiling to work correctly, the children of an Fl_Tile must cover the entire area of the widget, but not overlap. This means that all children must touch each other at their edges, and no gaps can be left inside the Fl_Tile. Fl_Tile does not normally draw any graphics of its own. The "borders" which can be seen in the snapshot above are actually part of the children. Their boxtypes have been set to FL_DOWN_BOX creating the impression of "ridges" where the boxes touch. What you see are actually two adjacent FL_DOWN_BOX's drawn next to each other. All neighboring widgets share the same edge - the widget's thick borders make it appear as though the widgets aren't actually touching, but they are. If the edges of adjacent widgets do not touch, then it will be impossible to drag the corresponding edges.

Fl_Tile allows objects to be resized to zero dimensions. To prevent this you can use the resizable() to limit where corners can be dragged to. For more information see note below.

Even though objects can be resized to zero sizes, they must initially have non-zero sizes so the Fl_Tile can figure out their layout. If desired, call position() after creating the children but before displaying the window to set the borders where you want.

**Note on resizable(Fl_Widget &w):** The "resizable" child widget (which should be invisible) limits where the borders can be dragged to. All dragging will be limited inside the resizable widget's borders. If you don't set it, it will be possible to drag the borders right to the edges of the Fl_Tile widget, and thus resize objects on the edges to zero width or height. When the entire Fl_Tile widget is resized, the resizable() widget will keep its border distance to all borders the same (this is normal resize behavior), so that you can effectively set a border width that will never change. To ensure correct event delivery to all child widgets the resizable() widget must be the first child of the Fl_Tile widget group. Otherwise some events (e.g. FL_MOVE and FL_ENTER) might be consumed by the resizable() widget so that they are lost for widgets covered (overlapped) by the resizable() widget.

**Note**

You can still resize widgets inside the resizable() to zero width and/or height, i.e. box 2b above to zero width and box 3a to zero height.

See also

```c
void Fl_Group::resizable(Fl_Widget &w)
```

Example for resizable with 20 pixel border distance:

```c
int dx = 20, dy = dx;
Fl_Tile tile(50,50,300,300);
// create resizable() box first
Fl_Box r(tile.x()+dx,tile.y()+dy,tile.w()-2*dy,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.
32.136.2.1  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  

32.136.3  Member Function Documentation  

32.136.3.1  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  
1  if the event was used and can be deleted  

See also  
   Fl_Event  

Reimplemented from Fl_Group.  

32.136.3.2  position()

void Fl_Tile::position (  
   int oldx,  
   int oldy,  
   int newx,  
   int newy )

Drags the intersection at (oldx,oldy) to (newx,newy).  
This redraws all the necessary children.  
Pass zero as oldx or oldy to disable drag in that direction.
32.136.3.3 resize()

void Fl_Tile::resize (  
  int X,  
  int Y,  
  int W,  
  int H ) [virtual]


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

32.137 Fl_Tiled_Image Class Reference

This class supports tiling of images over a specified area.

```cpp
#include <Fl_Tiled_Image.H>
```

Inheritance diagram for Fl_Tiled_Image:

```
Fl_Tiled_Image

└── Fl_Image
```

Public Member Functions

- virtual void color_average (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- virtual Fl_Image * copy (int W, int H) const
  Creates a resized copy of the image.
- virtual void desaturate ()
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  Draws a tiled image.
- Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)
  The constructors create a new tiled image containing the specified image.
- Fl_Image * image ()
  Gets The image that is tiled.
- virtual ~Fl_Tiled_Image ()
  The destructor frees all memory and server resources that are used by the tiled image.

Static Public Member Functions

- static bool drawing_tiled_image ()
  Returns true when the FLTK library is currently drawing an Fl_Tiled_Image object.
### Protected Attributes

- int alloc_image_
- Fl_Image * image_

### Additional Inherited Members

#### 32.137.1 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.

#### 32.137.2 Constructor & Destructor Documentation

##### 32.137.2.1 Fl_Tiled_Image()

```cpp
Fl_Tiled_Image::Fl_Tiled_Image (Fl_Image * i, int W = 0, int H = 0)
```

The constructors create a new tiled image containing the specified image. Use a width and height of 0 to tile the whole window/widget.

**Note**

Due to implementation constraints in FLTK 1.3.3 and later width and height of 0 may not work as expected when used as background image in widgets other than windows. You may need to center and clip the image (label) and set the label type to FL_NORMAL_LABEL. Doing so will let the tiled image fill the whole widget as its background image. Other combinations of label flags may or may not work.

```cpp
#include "bg.xpm"
Fl_Pixmap *bg_xpm = new Fl_Pixmap(bg_xpm);
Fl_Tiled_Image *bg_tiled = new Fl_Tiled_Image(bg_xpm,0,0);
Fl_Box *box = new Fl_Box(40,40,300,100,"");
box->box(FL_UP_BOX);
box->labeltype(FL_NORMAL_LABEL);
box->align(FL_ALIGN_INSIDE | FL_ALIGN_CENTER | FL_ALIGN_CLIP);
box->image(bg_tiled);
```

**Note**

Setting an image (label) for a window may not work as expected due to implementation constraints in FLTK 1.3.x and maybe later. The reason is the way Fl::scheme() initializes the window’s label type and image. A possible workaround is to use another Fl_Group as the only child widget and to set the background image for this group as described above.

**Todo** Fix `Fl_Tiled_Image` as background image for widgets and windows and fix the implementation of `Fl::scheme(const char *)`.

#### 32.137.3 Member Function Documentation

##### 32.137.3.1 color_average()

```cpp
void Fl_Tiled_Image::color_average { Fl_Color c, float i } [virtual]
```

The `color_average()` method averages the colors in the image with the provided FLTK color value. The first argument specifies the FLTK color to be used. The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

32.137.3.2 copy()

Fl_Image * Fl_Tiled_Image::copy (  
   int W, 
   int H ) const [virtual]

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- \( w() == \text{data}_w() == W \)
- \( h() == \text{data}_h() == H \)

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Parameters

| in | \( W,H \) | Requested width and height of the new image |

Note

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' must also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.
Reimplemented from Fl_Image.

32.137.3.3 desaturate()

void Fl_Tiled_Image::desaturate ( ) [virtual]

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

32.137.3.4 draw()

void Fl_Tiled_Image::draw (  
   int \( X \),  
   int \( Y \),  
   int \( W \),  
   int \( H \),  
   int \( cx = 0 \),  
   int \( cy = 0 \) ) [virtual]
Draws a tiled image. Tiled images can be used as background images for widgets and windows. However, due to implementation 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

### 32.138 Fl_Timeout Class Reference

Class Fl_Timeout handles all timeout related functions.

```c
#include <Fl_Timeout.h>
```

#### Static Public Member Functions

- static void add_timeout (double time, Fl_Timeout_Handler cb, void *data)
- static void do_timeouts ()
  
  *Elapse timers and call their callbacks if any timers are expired.*
- static void elapse_timeouts ()
  
  *Elapse all timers w/o calling their callbacks.*
- static int has_timeout (Fl_Timeout_Handler cb, void *data)
  
  *Returns whether the given timeout is active.*
- static void remove_timeout (Fl_Timeout_Handler cb, void *data)
  
  *Remove a timeout callback.*
- static void repeat_timeout (double time, Fl_Timeout_Handler cb, void *data)
- static double time_to_wait (double ttw)
  
  *Returns the delay in seconds until the next timer expires, limited by ttw.*

#### Protected Member Functions

- double delay ()
  
  *Get the timer's delay in seconds.*
- void delay (double t)
  
  *Set the timer's delay in seconds.*
- void insert ()
  
  *Insert this timer entry into the active timer queue.*
- void make_current ()
  
  *Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.*
- void release ()
  
  *Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free timers.*
Static Protected Member Functions

- static Fl_Timeout * current ()
  
  Returns the first (top-most) timeout from the current timeout stack.

- static Fl_Timeout * get (double time, Fl_Timeout_Handler cb, void *data)
  
  Get an Fl_Timeout instance for further handling.

Protected Attributes

- Fl_Timeout_Handler callback
- void * data
- Fl_Timeout * next
- int skip
- double time

Static Protected Attributes

- static Fl_Timeout * current_timeout = 0
  
  The list of current timeouts is used to store the timeout whose callback is called while the callback is executed.

- static Fl_Timeout * first_timeout = 0
  
  List of active timeouts.

- static Fl_Timeout * free_timeout = 0
  
  List of free timeouts after use.

32.138.1 Detailed Description

Class Fl_Timeout handles all timeout related functions. All code is platform independent except retrieving a timestamp which requires calling a system driver function and potentially results in different timer resolutions (from milliseconds to microseconds).

32.138.2 Member Function Documentation

32.138.2.1 current()

Fl_Timeout * Fl_Timeout::current ( ) [static], [protected]
Returns the first (top-most) timeout from the current timeout stack. This returns a pointer to the timeout but does not remove it from the list of current timeouts. This should be the timeout that is currently executing its callback.

Returns

Fl_Timeout* The current timeout whose callback is running.

Return values

NULL if no callback is currently running.

32.138.2.2 elapse_timeouts()

void Fl_Timeout::elapse_timeouts ( ) [static]
Elapse all timers w/o calling their callbacks.
All timer values are adjusted by the delta time since the last call. This method does NOT call timer callbacks if timers are expired.
This must be called before new timers are added to the timer queue to make sure that the next timer decrement does not count down too much time.

See also

Fl_Timeout::do_timeouts()

32.138.2.3 get()

Fl_Timeout * Fl_Timeout::get (  
  double time,  
  Fl_Timeout_Handler cb,  
  void * data ) [static], [protected]

Get an Fl_Timeout instance for further handling.
The timer object will be initialized with the input parameters as given by Fl::add_timeout() or Fl::repeat_timeout().
Fl_Timeout objects are maintained in three queues:

- active timer queue
- list (stack, i.e. LIFO) of currently executing timer callbacks
- free timer entries.

When the FLTK program is launched all queues are empty. Whenever a new timer object is required the get() method is called and a timer object is either found in the queue of free timer entries or a new timer object is created (operator new).
Active timer entries are inserted into the "active timer queue" until they expire and their callback is called.
Before the callback is called the timer entry is inserted into the list of current timers, i.e. it becomes the Fl_Timeout::current() timeout. This can be used in Fl::repeat_timeout() to find out if and how long the current timeout has been delayed.
When a timer is no longer used it is popped from the current list and inserted into the "free timer" list so it can be reused later.
Timer queue entries are never returned to the system, there’s no garbage collection. The total number of timer objects is determined by the largest number of concurrently active timers.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>requested delta time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>timer callback</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>userdata for timer callback</td>
</tr>
</tbody>
</table>

Returns

Fl_Timeout* Timer entry

See also

Fl::add_timeout(), Fl::repeat_timeout()

32.138.2.4 has_timeout()

int Fl_Timeout::has_timeout (  
  Fl_Timeout_Handler cb,  
  void * data ) [static]

Returns whether the given timeout is active.
This returns whether a timeout handler already exists in the queue of active timers.
If data == NULL only the Fl_Timeout_Handler cb must match to return true, otherwise data must also match.
Note

It is a restriction that there is no way to look for a timeout whose data is NULL (zero). Therefore using 0 (zero, NULL) as the timeout data value is discouraged, unless you're sure that you will never need to use Fl::has_timeout(callback, (void *)0);

Implements Fl::has_timeout(Fl_Timeout_Handler cb, void *data)

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
</tr>
</tbody>
</table>

Returns

whether the timer was found in the queue

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not found</td>
</tr>
<tr>
<td>1</td>
<td>found</td>
</tr>
</tbody>
</table>

### 32.138.2.5 insert()

void Fl_Timeout::insert ( ) [protected]
Insert this timer entry into the active timer queue.
The timer is inserted at the required position so the timer queue is always ordered by due time.

### 32.138.2.6 make_current()

void Fl_Timeout::make_current ( ) [protected]
Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.
This becomes the current() timeout which can be used in Fl::repeat_timeout().

See also

Fl_Timeout::current()

### 32.138.2.7 release()

void Fl_Timeout::release ( ) [protected]
Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free timers.
Typical code in the library would look like:

// The timeout ptr Fl_Timeout *t has expired, run its callback
(t->make_current());
(t->callback)(t->data);
(t->release());

### 32.138.2.8 remove_timeout()

void Fl_Timeout::remove_timeout (  
    Fl_Timeout_Handler cb,  
    void * data ) [static]
Remove a timeout callback.
It is harmless to remove a timeout callback that no longer exists.
32.138 Fl_Timeout Class Reference

Note
This version removes all matching timeouts, not just the first one. This may change in the future.

Implements Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)

32.138.2.9 time_to_wait()

double Fl_Timeout::time_to_wait()
  double ttw ) [static]
Returns the delay in seconds until the next timer expires, limited by ttw.
This function calculates the time to wait for the FLTK event queue processing, depending on the given value ttw.
If at least one timer is active and its timeout value is smaller than ttw then this value is returned. Fl::wait() will wait
no longer than until the next timer expires.
If no timer is active this returns the input value ttw unchanged.
If at least one timer is expired this returns 0.0 so the event processing does not wait.

Parameters
	in ttw time to wait from Fl::wait() etc. (upper limit)

Returns
delay until next timeout or 0.0 (see description)

32.138.3 Member Data Documentation

32.138.3.1 current_timeout

Fl_Timeout * Fl_Timeout::current_timeout = 0 [static], [protected]
The list of current timeouts is used to store the timeout whose callback is called while the callback is executed.
This is used like a stack, the current timeout is pushed to the front of the list and once the callback is finished, that
timeout is removed and entered into the free list.
Background: Fl::repeat_timeout() needs to know which timeout triggered it and the exact schedule time and/or the
delay of that timeout, i.e. how long the scheduled time was missed before the callback was called. A static, global
variable is not sufficient since the user code can call other functions, e.g. dialogs, that run a nested event loop which
can run another timeout callback. Hence this list of "current" timeouts is used like a stack (last in, first out).
See also
  Fl_Timeout::push() Member function (method)

32.138.3.2 first_timeout

Fl_Timeout * Fl_Timeout::first_timeout = 0 [static], [protected]
List of active timeouts.
These timeouts can be triggered when due, which calls their callbacks. The lifetime of a timeout:

• active, in this queue
• callback running, in queue current_timeout
• done, in list of free timeouts, ready to be reused.
32.138.3.3 free_timeout

*Fl_Timeout* :: *Fl_Timeout::free_timeout = 0*  [static], [protected]

List of free timeouts after use.
Timeouts can be reused many times.
The documentation for this class was generated from the following files:

- *Fl_Timeout.h*
- *Fl_Timeout.cxx*

32.139 Fl_Timer Class Reference

This is provided only to emulate the Forms Timer widget.

```c
#include <Fl_Timer.H>
```

Inheritance diagram for Fl_Timer:

```
 Fl_Widget
  ↓
 Fl_Timer
```

**Public Member Functions**

- char `direction ()` const
  
  *Gets or sets the direction of the timer.*
- void `direction (char d)`
  
  *Gets or sets the direction of the timer.*
- `Fl_Timer (uchar t, int x, int y, int w, int h, const char *l)`
  
  *Creates a new Fl_Timer widget using the given type, position, size, and label string.*
- int `handle (int)`
  
  *Handles the specified event.*
- char `suspended ()` const
  
  *Gets or sets whether the timer is suspended.*
- void `suspended (char d)`
  
  *Sets the current timer value.*
- double `value ()` const
  
  *See void Fl_Timer::value(double)*
  
- void `value (double)`
  
  *Sets the current timer value.*
- `~Fl_Timer ()`
  
  *Destroys the timer and removes the timeout.*

**Protected Member Functions**

- void `draw ()`
  
  *Draws the widget.*

**Additional Inherited Members**

32.139.1 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.
32.139.2 Constructor & Destructor Documentation

32.139.2.1 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.

32.139.3 Member Function Documentation

32.139.3.1 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().

32.139.3.2 direction() [2/2]

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().

32.139.3.3 draw()

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:

```c++
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

32.139.3.4 handle()

int Fl_Timer::handle (  
  int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.

32.139.3.5  suspended()

char Fl_Timer::suspended ( ) const [inline]

Gets or sets whether the timer is suspended.

The documentation for this class was generated from the following files:

- Fl_Timer.H
- forms_timer.cxx

32.140  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

32.140.1  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().
32.140.2 Constructor & Destructor Documentation

32.140.2.1 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

| in | X, Y, W, H | position and size of the widget |
| in | L          | widget label, default is no label |

The documentation for this class was generated from the following files:
- Fl_Toggle_Button.H
- Fl_Button.cxx

32.141 Fl_Tooltip Class Reference

The Fl_Tooltip class provides tooltip support for all FLTK widgets.

#include <Fl_Tooltip.H>

Static Public Member Functions

- static Fl_Color color ()  
  
  Gets the background color for tooltips.
- static void color (Fl_Color c)  
  
  Sets the background color for tooltips.
- static Fl_Widget ∗ current ()  
  
  Gets the current widget target.
- static void current (Fl_Widget ∗)  
  
  Sets the current widget target.
- static 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 f)
  
  *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 *)
32.141.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets. It contains only static methods.

![Figure 32.52 Fl_Tooltip Options]

32.141.2 Member Function Documentation

32.141.2.1 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.

32.141.2.2 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.

32.141.2.3 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.

32.141.2.4 delay() [1/2]

static float Fl_Tooltip::delay ( ) [inline], [static]

Gets the tooltip delay. The default delay is 1.0 seconds.

32.141.2.5 delay() [2/2]

static void Fl_Tooltip::delay ( float f ) [inline], [static]

Sets the tooltip delay. The default delay is 1.0 seconds.
32.141.2.6 disable()

static void Fl_Tooltip::disable ( ) [inline], [static]
Same as enable(0), disables tooltips on all widgets.

32.141.2.7 enable()

static void Fl_Tooltip::enable ( int b = 1 ) [inline], [static]
Enables tooltips on all widgets (or disables if b is false).

32.141.2.8 enabled()

static int Fl_Tooltip::enabled ( ) [inline], [static]
Returns non-zero if tooltips are enabled.

32.141.2.9 enter_area()

void Fl_Tooltip::enter_area ( Fl_Widget * wid, int x, int y, int w, int h, const char * t ) [static]
You may be able to use this to provide tooltips for internal pieces of your widget. Call this after setting Fl::belowmouse() to your widget (because that calls the above enter() method). Then figure out what thing the mouse is pointing at, and call this with the widget (this pointer is used to remove the tooltip if the widget is deleted or hidden, and to locate the tooltip), the rectangle surrounding the area, relative to the top-left corner of the widget (used to calculate where to put the tooltip), and the text of the tooltip (which must be a pointer to static data as it is not copied).

32.141.2.10 font() [1/2]

static Fl_Font Fl_Tooltip::font ( ) [inline], [static]
Gets the typeface for the tooltip text.

32.141.2.11 font() [2/2]

static void Fl_Tooltip::font ( Fl_Font i ) [inline], [static]
Sets the typeface for the tooltip text.

32.141.2.12 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.
32.141.2.13  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.

32.141.2.14  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.

32.141.2.15  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.

32.141.2.16  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.

32.141.2.17  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.

32.141.2.18  margin_width() [1/2]

static int Fl_Tooltip::margin_width ( ) [inline], [static]
Gets the amount of extra space left/right of the tooltip's text.
Default is 3.

32.141.2.19  margin_width() [2/2]

static void Fl_Tooltip::margin_width (  
    int v ) [inline], [static]
Sets the amount of extra space left/right of the tooltip's text.
Default is 3.

32.141.2.20  size() [1/2]

static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]
Gets the size of the tooltip text.

32.141.2.21  size() [2/2]

static void Fl_Tooltip::size (  
    Fl_Fontsize s ) [inline], [static]
Sets the size of the tooltip text.
32.141.2.22  textcolor() [1/2]

```cpp
static Fl_Color Fl_Tooltip::textcolor () [inline], [static]
```

Gets the color of the text in the tooltip.
The default is black.

32.141.2.23  textcolor() [2/2]

```cpp
static void Fl_Tooltip::textcolor ( Fl_Color c ) [inline], [static]
```

Sets the color of the text in the tooltip.
The default is black.

32.141.2.24  wrap_width() [1/2]

```cpp
static int Fl_Tooltip::wrap_width ( ) [inline], [static]
```

Gets the maximum width for tooltip's text before it word wraps.
Default is 400.

32.141.2.25  wrap_width() [2/2]

```cpp
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

32.142  Fl_Tree Class Reference

Tree widget.

```cpp
#include <Fl_Tree.H>
```

Inheritance diagram for Fl_Tree:

```
Fl_Widget
```

```
Fl_Group
```

```
Fl_Tree
```

Public Member Functions

- **Fl_Tree_Item ∗ add (const char ∗path, Fl_Tree_Item ∗newitem=0)**
  
  Adds a new item, given a menu style 'path'.

- **Fl_Tree_Item ∗ add (Fl_Tree_Item ∗parent_item, const char ∗name)**
  
  Add a new child item labeled 'name' to the specified 'parent_item'.

- **void calc_dimensions ()**
  
  Recalculate widget dimensions and scrollbar visibility, normally managed automatically.
• void calc_tree ()
  Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.

• Fl_Tree_Item * callback_item ()
  Gets the item that caused the callback.

• void callback_item (Fl_Tree_Item *item)
  Sets the item that was changed for this callback.

• Fl_Tree_Reason callback_reason () const
  Gets the reason for this callback.

• void callback_reason (Fl_Tree_Reason reason)
  Sets the reason for this callback.

• void clear ()
  Clear the entire tree's children, including the root.

• void clear_children (Fl_Tree_Item *item)
  Clear all the children for 'item'.

• int close (const char *path, int docallback=1)
  Closes the item specified by 'path'.

• int close (Fl_Tree_Item *item, int docallback=1)
  Closes the specified 'item'.

• Fl_Image * closeicon () const
  Returns the icon to be used as the 'close' icon.

• void closeicon (Fl_Image *val)
  Sets the icon to be used as the 'close' icon.

• Fl_Color connectorcolor () const
  Get the connector color used for tree connection lines.

• void connectorcolor (Fl_Color val)
  Set the connector color used for tree connection lines.

• Fl_Tree_Connector connectorstyle () const
  Returns the line drawing style for inter-connecting items.

• void connectorstyle (Fl_Tree_Connector val)
  Sets the line drawing style for inter-connecting items.

• int connectorwidth () const
  Gets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

• void connectorwidth (int val)
  Sets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

• int deselect (const char *path, int docallback=1)
  Deselect an item specified by 'path'.

• int deselect (Fl_Tree_Item *item, int docallback=1)
  Deselect the specified item.

• int deselect_all (Fl_Tree_Item *item=0, int docallback=1)
  Deselect 'item' and all its children.

• void display (Fl_Tree_Item *item)
  Displays 'item', scrolling the tree as necessary.

• int displayed (Fl_Tree_Item *item)
  See if 'item' is currently displayed on-screen (visible within the widget).

• int extend_selection (Fl_Tree_Item *from, Fl_Tree_Item *to, int val=1, bool visible=false)
  Extend a selection between 'from' and 'to' depending on 'visible'.

• int extend_selection_dir (Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible)
  Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.

• Fl_Tree_Item * find_clicked (int yonly=0)
  Non-const version of Fl_Tree::find_clicked(int yonly) const.
const Fl_Tree_Item * find_clicked(int yonly=0) const
Find the item that was last clicked on.

Fl_Tree_Item * find_item(const char *path)
Non-const version of Fl_Tree::find_item(const char *path) const
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(const char *path) const
See if item specified by 'path' is closed.

int is_close(Fl_Tree_Item *item) const
See if the specified 'item' is closed.

int is_hscroll_visible() const
See if the horizontal scrollbar is currently visible.

int is_open(const char *path) const
See if item specified by 'path' is open.

int is_open(Fl_Tree_Item *item) const
See if 'item' is open.

int is_scrollbar(Fl_Widget *w)
See if widget 'w' is one of the Fl_Tree widget's scrollbars.

int is_selected(const char *path)
See if item specified by 'path' is selected.

int is_selected(Fl_Tree_Item *item) const
See if the specified 'item' is selected.

int is_vscroll_visible() const
See if the vertical scrollbar is currently visible.

Fl_Tree_Item * item_clicked()
Return the item that was last clicked.

Fl_Tree_Item_Draw_Mode item_draw_mode() const
Get the 'item draw mode' used for the tree.

- void item_draw_mode (Fl_Tree_Item_Draw_Mode mode)
  Set the 'item draw mode' used for the tree to `mode`.

- void item_draw_mode (int mode)
  Set the 'item draw mode' used for the tree to integer `mode`.

- void item_labelbgcolor (Fl_Color val)
  Set the default label background color used for creating new items.

- Fl_Color item_labelbgcolor (void) const
  Get the default label background color used for creating new items.

- void item_labelfgcolor (Fl_Color val)
  Set the default label foreground color used for creating new items.

- Fl_Color item_labelfgcolor (void) const
  Get the default label foreground color used for creating new items.

- Fl_Font item_labelfont () const
  Get the default font face used for creating new items.

- void item_labelfont (Fl_Font val)
  Set the default font face used for creating new items.

- Fl_Fontsize item_labelsize () const
  Get the default label font size used for creating new items.

- void item_labelsize (Fl_Fontsize val)
  Set the default label font size used for creating new items.

- int item_pathname (char ∗pathname, int pathnamelen, const Fl_Tree_Item ∗item) const
  Return `pathname' of size `pathnamelen' for the specified `item'.

- Fl_Tree_Item_Reselect_Mode item_reselect_mode () const
  Returns the current item re/selection mode.

- void item_reselect_mode (Fl_Tree_Item_Reselect_Mode mode)
  Sets the item re/selection mode.

- int labelmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the label text.

- void labelmarginleft (int val)
  Set the amount of white space (in pixels) that should appear to the left of the label text.

- Fl_Tree_Item ∗last ()
  Returns the last item in the tree.

- Fl_Tree_Item ∗last_selected_item ()
  Returns the last selected item in the tree.

- Fl_Tree_Item ∗last_visible ()
  Returns the last open(), visible item in the tree.

- Fl_Tree_Item ∗last_visible_item ()
  Returns the last open(), visible item in the tree.

- int linespacing () const
  Get the amount of white space (in pixels) that should appear between items in the tree.

- void linespacing (int val)
  Sets the amount of white space (in pixels) that should appear between items in the tree.

- void load (class Fl_Preferences &)
  Load FLTK preferences.

- int marginbottom () const
  Get the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

- void marginbottom (int val)
  Sets the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.
• int marginleft () const
  Get the amount of white space (in pixels) that should appear between the widget's left border and the tree's contents.
• void marginleft (int val)
  Set the amount of white space (in pixels) that should appear between the widget's left border and the left side of the tree's contents.
• int margintop () const
  Get the amount of white space (in pixels) that should appear between the widget's top border and the top of the tree's contents.
• void margintop (int val)
  Sets the amount of white space (in pixels) that should appear between the widget's top border and the top of the tree's contents.
• Fl_Tree_Item ∗ next (Fl_Tree_Item ∗item=0)
  Return the next item after 'item', or 0 if no more items.
• Fl_Tree_Item ∗ next_item (Fl_Tree_Item ∗item, int dir=FL_Down, bool visible=false)
  Returns next item after 'item' in direction 'dir' depending on 'visible'.
• Fl_Tree_Item ∗ next_selected_item (Fl_Tree_Item ∗item=0, int dir=FL_Down)
  Returns the next selected item above or below 'item', depending on 'dir'.
• Fl_Tree_Item ∗ next_visible_item (Fl_Tree_Item ∗start, int dir)
  Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified 'item', or 0 if no more items.
• int open (const char ∗path, int docallback=1)
  Opens the item specified by 'path'.
• int open (Fl_Tree_Item ∗item, int docallback=1)
  Open the specified 'item'.
• void open_toggle (Fl_Tree_Item ∗item, int docallback=1)
  Toggle the open state of 'item'.
• int openchild_marginbottom () const
  Get the amount of white space (in pixels) that should appear below an open child tree's contents.
• void openchild_marginbottom (int val)
  Set the amount of white space (in pixels) that should appear below an open child tree's contents.
• Fl_Image ∗ openicon () const
  Returns the icon to be used as the 'open' icon.
• void openicon (Fl_Image ∗val)
  Sets the icon to be used as the 'open' icon.
• const Fl_Tree_Prefs & prefs () const
• Fl_Tree_Item ∗ prev (Fl_Tree_Item ∗item=0)
  Return the previous item before 'item', or 0 if no more items.
• void recalc_tree ()
  Schedule tree to recalc the entire tree size.
• int remove (Fl_Tree_Item ∗item)
  Remove the specified 'item' from the tree.
• void resize (int, int, int, int)
  Resizes the Fl_Group widget and all of its children.
• Fl_Tree_Item ∗ root ()
  Returns the root item.
• void root (Fl_Tree_Item ∗newitem)
  Sets the root item to 'newitem'.
• void root_label (const char ∗new_label)
  Set the label for the root item to 'new_label'.
• int scrollbar_size () const
  Gets the default size of scrollbars' troughs for this widget in pixels.
• void scrollbar_size (int size)
  Sets the pixel size of the scrollbars' troughs to 'size' for this widget, in pixels.

• int select (const char *path, int docallback=1)
  Select the item specified by 'path'.

• int select (FI_Tree_Item *item, int docallback=1)
  Select the specified 'item'.

• int select_all (FI_Tree_Item *item=0, int docallback=1)
  Select 'item' and all its children.

• int select_only (FI_Tree_Item *selitem, int docallback=1)
  Select only the specified item, deselecting all others that might be selected.

• void select_toggle (FI_Tree_Item *item, int docallback=1)
  Toggle the select state of the specified 'item'.

• FI_Boxtype selectbox () const
  Sets the style of box used to draw selected items.

• void selectbox (FI_Boxtype val)
  Gets the style of box used to draw selected items.

• FI_Tree_Select selectmode () const
  Gets the tree's current selection mode.

• void selectmode (FI_Tree_Select val)
  Sets the tree's selection mode.

• void set_item_focus (FI_Tree_Item *item)
  Set the item that currently should have keyboard focus.

• void show_item (FI_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 (FI_Tree_Item *item, int yoff)
  Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the FI_Tree widget's display.

• void show_item_bottom (FI_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

• void show_item_middle (FI_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is in the middle of the display.

• void show_item_top (FI_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is at the top of the display.

• void show_self ()
  Print the tree as 'ascii art' to stdout.

• int showcollapse () const
  Returns 1 if the collapse icon is enabled, 0 if not.

• void showcollapse (int val)
  Set if we should show the collapse icon or not.

• int showroot () const
  Returns 1 if the root item is to be shown, or 0 if not.

• void showroot (int val)
  Set if the root item should be shown or not.

• FI_Tree_Sort sortorder () const
  Set the default sort order used when items are added to the tree.

• void sortorder (FI_Tree_Sort val)
  Gets the sort order used to add items to the tree.

• FI_Image *usericon () const
  Returns the FI_Image being used as the default user icon for all newly created items.

• void usericon (FI_Image *val)
  Sets the FI_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 _tih
  Tree widget inner xywh dimension: inside borders + scrollbars.
• int _tiw
• int _tix
• int _tiy
• int _toh
  Tree widget outer xywh dimension: outside scrollbars, inside widget border.
• int _tow
• int _tox
• int _toy
• int _tree_h
  the calculated height of the entire tree hierarchy. See calc_tree()
• int _tree_w
  the calculated width of the entire tree hierarchy. See calc_tree()
• Fl_Scrollbar * _vscroll
  Vertical scrollbar.

Friends

• class Fl_Tree_Item
Additional Inherited Members

32.142.1 Detailed Description

Tree widget.

```cpp
#include <FL/Fl_Tree.H>

Fl_Tree tree(X,Y,W,H);
```

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:

```cpp
Fl_Tree tree(X,Y,W,H);
```

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_TreePrefs`, 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:
**Class Documentation**

- **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:

```c
// Walk all the items in the tree, and print their labels
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) 
    printf("Item: %s\n", item->label());
```

To recursively walk all the children of a particular item, define a function that uses recursion:

```c
// Find all of the item's children and print an indented report of their labels
void my_print_all_children(Fl_Tree_Item *item, int indent=0) {
    for ( int t=0; t<item->children(); t++ ) {
        printf("%s Item: %s\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:

```cpp
tree = new Fl_Tree(...);
tree->item_labelfont(FL_COURIER); // Use Courier font for all new items
tree->item_labelfgcolor(FL_RED); // Use red color for labels of all new items
// Now create the items in the tree using the above defaults.
tree->add("Aaa");
tree->add("Bbb");
```

To change the font and color of all existing items in the tree:

```cpp
// Change the font and color of all items currently in the tree
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    item->labelfont(FL_COURIER);
    item->labelcolor(FL_RED);
}
```

**DISPLAY DESCRIPTION**

The following image shows the tree's various visual elements and the methods that control them:

![Figure 32.54 Fl_Tree elements](image_url)

The following shows the protected dimension variables 'tree inner' (tix...) and 'tree outer' (tox..):

![Figure 32.55 Fl_Tree inner/outer dimensions](image_url)
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>

32.142.2 Member Function Documentation

32.142.2.1 add() [1/2]

\[\text{Fl\_Tree\_Item} \ast \text{Fl\_Tree::add (}\]
\[\text{const char} \ast \text{path,}\]
\[\text{Fl\_Tree\_Item} \ast \text{item = 0 })\]

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.

```c
: tree->add("/Holidays/Photos/12\25\2010"); // Adds item "12/25/2010"
: tree->add("/Pathnames/c:\\Program Files\\MyApp"); // Adds item "c:\Program Files\MyApp"
:
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>The path to the item, e.g. “Flintstone/Fred”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The new item to be added. If NULL, a new item is created with a name that is the last element in 'path'.</td>
</tr>
</tbody>
</table>

Returns

The new item added, or 0 on error.
32.142.2.2  add() [2/2]

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. |
| in | name         | The label for the new item                                           |

Returns

The new item added.

Version

1.3.0 release

32.142.2.3  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

32.142.2.4  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)

32.142.2.5 callback_item() [1/2]

```c
Fl_Tree_Item * Fl_Tree::callback_item ( )
```

Gets the item that caused the callback.
The `callback()` can use this value to see which item changed.

32.142.2.6 callback_item() [2/2]

```c
void Fl_Tree::callback_item ( Fl_Tree_Item * item )
```

Sets the item that was changed for this callback.
Used internally to pass the item that invoked the callback.

32.142.2.7 callback_reason() [1/2]

```c
Fl_Tree_Reason Fl_Tree::callback_reason ( ) const
```

Gets the reason for this callback.
The `callback()` can use this value to see why it was called. Example:

```c
:
void MyTreeCallback(Fl_Widget *w, void *userdata) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = tree->callback_item(); // the item changed (can be NULL if more than one
    // item was changed!)
    switch (tree->callback_reason( ) ) { // reason callback was invoked
        case FL_TREE_REASON_OPENED: ..item was opened..
        case FL_TREE_REASON_CLOSED: ..item was closed..
        case FL_TREE_REASON_SELECTED: ..item was selected..
        case FL_TREE_REASON_RESELECTED: ..item was reselected (double-clicked, etc)..
        case FL_TREE_REASON_DESELECTED: ..item was deselected..
    }
}
:
:
See also

`item_reselect_mode()` – enables `FL_TREE_REASON_RESELECTED` events

32.142.2.8 callback_reason() [2/2]

```c
void Fl_Tree::callback_reason ( Fl_Tree_Reason reason )
```

Sets the reason for this callback.
Used internally to pass the reason the callback was invoked.

32.142.2.9 clear()

```c
void Fl_Tree::clear ( )
```

Clear the entire tree's children, including the root.
The tree will be left completely empty.
### 32.142.2.10 clear_children()

```cpp
void Fl_Tree::clear_children (  
    Fl_Tree_Item * item  
)
```

Clear all the children for 'item'.
Item may not be NULL.

### 32.142.2.11 close() [1/2]

```cpp
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>
<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 – 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()

### 32.142.2.12 close() [2/2]

```cpp
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>

Generated by Doxygen
Returns

- 1 – item was closed
- 0 – item was already closed, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

32.142.2.13 closeicon() [1/2]

Fl_Image * Fl_Tree::closeicon() const

Returns the icon to be used as the 'close' icon.
If none was set, the internal default is returned, a simple '[' icon.

32.142.2.14 closeicon() [2/2]

void Fl_Tree::closeicon(Fl_Image * val)

Sets the icon to be used as the 'close' icon.
This overrides the built in default '[' icon.

Parameters

| in | val | -- The new image, or zero to use the default [''] icon. |

32.142.2.15 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.

32.142.2.16 deselect() [1/2]

int Fl_Tree::deselect(const char * path, int docallback = 1)

Deselect an item specified by 'path'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. deselect("← Holidays/12\25\2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

| in | path | -- the tree item's pathname (e.g. "Flintstones/Fred") |
| in | docallback | -- A flag that determines if the callback() is invoked or not: |
|     |          | • 0 - the callback() is not invoked |
|     |          | • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED |
Returns

- 1 - OK: item's state was changed
- 0 - OK: item was already deselected, no change was made
- -1 - ERROR: item was not found

32.142.2.17 deselect() [2/2]

```cpp
int Fl_Tree::deselect (
    Fl_Tree_Item * item,
    int docallback = 1 )
```

Deselect the specified item.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

Parameters

| 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

32.142.2.18 deselect_all()

```cpp
int Fl_Tree::deselect_all ( 
    Fl_Tree_Item * item = 0,
    int docallback = 1 )
```

Deselect 'item' and all its children.
If item is NULL, `first()` is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

Parameters

| in | item | The item that will be deselected (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_DESELECTED |
Returns

Count of how many items were actually changed to the deselected state.

32.142.2.19 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.

32.142.2.20 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.

32.142.2.21 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 `extend_selection_dir(Fl_Tree_Item*,Fl_Tree_Item*,int dir,int val,bool vis)` method, but direction (up or down) doesn't need to be known. We're less efficient because we search the tree for to/from, then operate on items in between. The more efficient method avoids the "search", but necessitates a direction to be specified to find 'to'. Used by SHIFT-click to extend a selection between two items inclusive. Handles calling `redraw()` if anything changed.

Parameters

- **in from** Starting item
- **in to** Ending item
- **in val** Select or deselect items (0=deselect, 1=select, 2=toggle)
- **in visible** true=affect only open(), visible items, false=affect open or closed items (default)
Returns

The number of items whose selection states were changed, if any.

Version

1.3.3 ABI feature

32.142.2.22 extend_selection_dir()

```cpp
int Fl_Tree::extend_selection_dir (Fl_Tree_Item * from, Fl_Tree_Item * to, int dir, int val, bool visible)
```

Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.
Efficient: does not walk entire tree; starts with 'from' and stops at 'to' while moving in direction 'dir'. Dir must be specified though.
If dir cannot be known in advance, such as during SHIFT-click operations, the method `extend_selection(Fl_Tree_Item*, Fl_Tree_Item*, int, bool)` should be used.
Handles calling `redraw()` if anything changed.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>from</th>
<th>Starting item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>to</td>
<td>Ending item</td>
</tr>
<tr>
<td>in</td>
<td>dir</td>
<td>Direction to extend selection (FL_Up or FL_Down)</td>
</tr>
<tr>
<td>in</td>
<td>val</td>
<td>0=deselect, 1=select, 2=toggle</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=affect only open(), visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3

32.142.2.23 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

32.142.2.24 find_item()

const Fl_Tree_Item * Fl_Tree::find_item (const char * path) const

Find the item, given a menu style path, e.g. "/Parent/Child/item". There is both a const and non-const version of this method. Const version allows pure const methods to use this method to do lookups without causing compiler errors. To specify items or submenus that contain slashes ("/" or ") use an escape character to protect them, e.g.

```
: tree->add("/Holidays/Photos/12/25/2010"); // Adds item "12/25/2010"
```

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()

32.142.2.25 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.

```
: 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()
### first_selected_item()

```cpp
Fl_Tree_Item * Fl_Tree::first_selected_item ( )
```

Returns the first selected item in the tree.

Use this to walk the tree from top to bottom looking for all the selected items, e.g.

```cpp
// Walk tree forward, from top to bottom
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i) )
    printf("Selected item: %s\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()

### 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()  
* get_selected_items()

### 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;
for ( int t=0; t<items.total(); t++ ) {
    Fl_Tree_Item item = items[t];
    // do stuff with each selected item...
}
```
Parameters

| out | ret_items | The returned array of selected items. |

Returns

The number of items in the returned array.

See also

first_selected_item(), next_selected_item()

Version

1.3.3 ABI feature

32.142.2.30 handle()

int Fl_Tree::handle (int e) [virtual]

Standard FLTK event handler for this widget.

Todo add Fl_Widget_Tracker (see Fl_Browser_cxx::handle())

Reimplemented from Fl_Group.

32.142.2.31 hposition() [1/2]

int Fl_Tree::hposition ( ) const

Returns the horizontal scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the left edge of the screen.

See also

hposition(int), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

32.142.2.32 hposition() [2/2]

void Fl_Tree::hposition (int pos)

Sets the horizontal scroll offset to position 'pos'.
The position is how many pixels of the tree are scrolled off the left edge of the screen.

Parameters

| in | pos | The vertical position (in pixels) to scroll the tree to. |

See also

hposition(), vposition(), vposition(int)
32.142.2.33  insert()

Fl_Tree_Item * Fl_Tree::insert (Fl_Tree_Item * item, const char * name, int pos )

Insert a new item 'name' into 'item's children at position 'pos'.
If pos is out of range the new item is

- prepended if pos < 0 or
- appended if pos > item->children().

Note: pos == children() is not considered out of range: the item is appended to the child list.

Example:

```cpp
Fl_Tree_Item *item = tree->find_item("Aaa"); // get parent item Aaa
if (item) tree->insert(item, "New item", 2); // insert as a child of Aaa at index #2
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The existing item to insert new child into. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>The label for the new item</td>
</tr>
<tr>
<td>in</td>
<td>pos</td>
<td>The position of the new item in the child list</td>
</tr>
</tbody>
</table>

Returns

The new item added.

See also

insert_above()
Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>above</code></td>
<td>the item above which to insert the new item. Must not be NULL.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>name</code></td>
<td>the name of the new item</td>
</tr>
</tbody>
</table>

Returns

The new item added, or 0 if 'above' could not be found.

See also

`insert()`

### 32.142.2.35 is_close() [1/2]

```cpp
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

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>path</code></td>
<td>the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</td>
</tr>
</tbody>
</table>

Returns

- 1 - OK: item is closed
- 0 - OK: item is open
- -1 - ERROR: item was not found

### 32.142.2.36 is_close() [2/2]

```cpp
int Fl_Tree::is_close (Fl_Tree_Item * item ) const
```

See if the specified 'item' is closed.

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>item</code></td>
<td>the item to be tested. Must not be NULL.</td>
</tr>
</tbody>
</table>

Returns

- 1 : item is closed
- 0 : item is open

### 32.142.2.37 is_hscroll_visible()

```cpp
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.

32.142.2.38  is_open() [1/2]

int Fl_Tree::is_open ( const char * path ) const

See if item specified by 'path' is open.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_open("Holidays/12\25\2010").
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

| in | path | – the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

- 1 - OK: item is open
- 0 - OK: item is closed
- -1 - ERROR: item was not found

See also

Fl_Tree_Item::visible_r()
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

32.142.2.41  is_selected() [1/2]

int Fl_Tree::is_selected (const char * path) const
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

32.142.2.42  is_selected() [2/2]

int Fl_Tree::is_selected (Fl_Tree_Item * item) const
See if the specified 'item' is selected.

Parameters

in  item  – the item to be tested. Must not be NULL.

Returns

• 1 : item selected
• 0 : item deselected

32.142.2.43  is_vscroll_visible()

int Fl_Tree::is_vscroll_visible () const
See if the vertical scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

### 32.142.2.44 item_clicked() [1/2]

**Fl_Tree_Item** ∗ [Fl_Tree::item_clicked ( )](#)

Return the item that was last clicked.

Valid only from within the [callback()](#).

Returns

The item clicked, or 0 if none. 0 may also be used to indicate several items were clicked/changed.

**Deprecated** in 1.3.3 ABI – use [callback_item()](#) instead.

### 32.142.2.45 item_clicked() [2/2]

```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.

### 32.142.2.46 item_draw_mode() [1/3]

**Fl_Tree_Item_Draw_Mode** [Fl_Tree::item_draw_mode ( ) const](#)

Get the 'item draw mode' used for the tree.

**Version**

1.3.1 ABI feature

### 32.142.2.47 item_draw_mode() [2/3]

```cpp
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

### 32.142.2.48 item_draw_mode() [3/3]

```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
32.142.2.49  **item_labelbgcolor()** [1/2]

```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)`.

32.142.2.50  **item_labelbgcolor()** [2/2]

```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'.

32.142.2.51  **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)`.

32.142.2.52  **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)`.

32.142.2.53  **item_labelsize()**

```cpp
void Fl_Tree::item_labelsize (  
    Fl_Fontsize val  
)
```

Set the default label font size used for creating new items. To change the font size on a per-item basis, use `Fl_Tree_Item::labelsize(Fl_Fontsize)`.

32.142.2.54  **item_pathname()**

```cpp
int Fl_Tree::item_pathname (  
    char * pathname,  
    int pathnamelen,  
    const Fl_Tree_Item * item ) const
```

Return 'pathname' of size 'pathnamelen' for the specified 'item'. If 'item' is NULL, `root()` is used.

The tree's root will be included in the pathname if `showroot()` is on. Menu items or submenus that contain slashes ('/' or '\') in their names will be escaped with a backslash. This is symmetrical with the `add()` function which uses the same escape pattern to set names.

**Parameters**

<table>
<thead>
<tr>
<th>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="")
32.142.2.55  item_reselect_mode() [1/2]

Fl_Tree_Item_Reselect_Mode Fl_Tree::item_reselect_mode () const

Returns the current item re/selection mode.

Version

1.3.1 ABI feature

32.142.2.56  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

32.142.2.57  last()

Fl_Tree_Item * Fl_Tree::last ( )

Returns the last item in the tree.

This can be used to walk the tree in reverse, e.g.

    for ( Fl_Tree_Item *item = tree->last(); item; item = tree->prev() )
        printf("Item: %s\n", item->label());

Returns

Last item in the tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()

32.142.2.58  last_selected_item()

Fl_Tree_Item * Fl_Tree::last_selected_item ( )

Returns the last selected item in the tree.

Use this to walk the tree in reverse from bottom to top looking for all the selected items, e.g.

    // Walk tree in reverse, from bottom to top
    for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
        printf("Selected item: %s\n", i->label());

Returns

The last selected item, or 0 if none.
32.142.2.59  last_visible()

Fl_Tree_Item * Fl_Tree::last_visible ( )

Returns the last open(), visible item in the tree.

Deprecated in 1.3.3 – use last_visible_item() instead.

32.142.2.60  last_visible_item()

Fl_Tree_Item * Fl_Tree::last_visible_item ( )

Returns the last open(), visible item in the tree.

Returns

  Last visible item in the tree, or 0 if none.

See also

  first_visible_item(), last_visible_item(), next_visible_item()

32.142.2.61  load()

void Fl_Tree::load ( class Fl_Preferences & prefs )

Load FLTK preferences.
Read a preferences database into the tree widget.
A preferences database is a hierarchical collection of data which can be directly loaded into the tree view for inspection.

Parameters

  in  prefs  the Fl_Preferences database

32.142.2.62  next()

Fl_Tree_Item * Fl_Tree::next ( Fl_Tree_Item * item = 0 )

Return the next item after 'item', or 0 if no more items.
Use this code to walk the entire tree:
for ( Fl_Tree_Item *i = tree->first(); i; i = tree->next(i) )
  printf("Item: %s
", 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()

32.142.2.63  next_item()

Fl_Tree_Item * Fl_Tree::next_item ( 
  Fl_Tree_Item * item,
  int dir = FL_Down,
  bool visible = false )

Returns next item after 'item' in direction 'dir' depending on 'visible'.
Next item will be above (if dir==FL_Up) or below (if dir==FL_Down). If 'visible' is true, only items whose
parents are open() will be returned. If 'visible' is false, even items whose parents are close()ed will be
returned.
If item is 0, the return value will be the result of this truth table:

<table>
<thead>
<tr>
<th>visible=true</th>
<th>visible=false</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir=FL_Up:</td>
<td>last_visible_item()</td>
</tr>
<tr>
<td>dir=FL_Down:</td>
<td>first_visible_item()</td>
</tr>
</tbody>
</table>

Example use:

;  // Walk down the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_item(i, FL_Down, true) )
  printf("Item: %s\n", i->label());
;  // Walk up the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_item(i, FL_Up, true) )
  printf("Item: %s\n", i->label());
;  // Walk down the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->first(); i; i=tree->next_item(i, FL_Down, false) )
  printf("Item: %s\n", i->label());
;  // Walk up the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->last(); i; i=tree->next_item(i, FL_Up, false) )
  printf("Item: %s\n", i->label());
;

Parameters

| in | item | The item to use to find the next item. If NULL, returns 0. |
| in | dir | Can be FL_Up or FL_Down (default=FL_Down or 'next') |
| in | visible | true=return only open(), visible items, 
false=return open or closed items (default) |
Returns

Next item in tree in the direction and visibility specified, or 0 if no more items of specified visibility in that
direction.

See also

first(), last(), next(),
first_visible_item(), last_visible_item(), next_visible_item(),
first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

32.142.2.64  next_selected_item()

\begin{code}
Fl_Tree_Item * Fl_Tree::next_selected_item ( 
    Fl_Tree_Item * item = 0, 
    int dir = FL_Down )
\end{code}

Returns the next selected item above or below 'item', depending on 'dir'.
If 'item' is 0, search starts at either first() or last(), depending on 'dir':
first() if 'dir' is FL_Down (default),
last() if 'dir' is FL_Up.
Use this to walk the tree looking for all the selected items, e.g.

```
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i, FL_Down) )
    printf("Item: %s\n", i->label());
// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
    printf("Item: %s\n", i->label());
```

Parameters

| in | item | The item above or below which we'll find the next selected item. If NULL, first() is used if FL_Down, last() if FL_Up. (default=NULL) |
| in | dir  | The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default) |

Returns

The next selected item, or 0 if there are no more selected items.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

32.142.2.65  next_visible_item()

\begin{code}
Fl_Tree_Item * Fl_Tree::next_visible_item ( 
    Fl_Tree_Item * item, 
    int dir )
\end{code}

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
", 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
", 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

---

32.142.2.66 open() [1/2]

```c
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()
32.142.67  open() [2/2]

```
int Fl_Tree::open (  
    Fl_Tree_Item ∗ item,  
    int docallback = 1 )
```

Open the specified 'item'.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

| in  | item | − the item to be opened. Must not be NULL. |
| in  | docallback | − A flag that determines if the callback() is invoked or not: |
|     |     | • 0 - callback() is not invoked |
|     |     | • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED |

Returns

• 1 – item was opened  
• 0 – item was already open, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

32.142.68  open_toggle()

```
void Fl_Tree::open_toggle (  
    Fl_Tree_Item ∗ item,  
    int docallback = 1 )
```

Toggle the open state of 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

| in  | item | − the item whose open state is to be toggled. Must not be NULL. |
| in  | docallback | − A flag that determines if the callback() is invoked or not: |
|     |     | • 0 - callback() is not invoked |
|     |     | • 1 - callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED |

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()
32.142.2.69 openicon() [1/2]

Returns the icon to be used as the 'open' icon.
If none was set, the internal default is returned, a simple '[+]' icon.

32.142.2.70 openicon() [2/2]

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. |

32.142.2.71 prev()

Return the previous item before 'item', or 0 if no more items.
This can be used to walk the tree in reverse, e.g.

```c
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->prev(item) )
    printf("Item: %s
", 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()

32.142.2.72 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.

32.142.2.73 remove()

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.

32.142.2.74 resize()

void Fl_Tree::resize (  
  int X,  
  int Y,  
  int W,  
  int H ) [virtual]

Resizes the Fl_Group widget and all of its children.  
The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules docu-
mented for Fl_Group::resizable(Fl_Widget*)

See also

Fl_Group::resizable(Fl_Widget*)
Fl_Group::resizable()
Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Group.

32.142.2.75 root()

void Fl_Tree::root (  
  Fl_Tree_Item * newitem )

Sets the root item to 'newitem'.  
If a root item already exists, clear() is called first to clear it before replacing it with newitem.  
Use this to install a custom item (derived from Fl_Tree_Item) as the root of the tree. This allows the derived class to 
implement custom drawing by overriding Fl_Tree_Item::draw_item_content().

Version

1.3.3

32.142.2.76 root_label()

void Fl_Tree::root_label (  
  const char * new_label )

Set the label for the root item to 'new_label'.  
Makes an internally managed copy of 'new_label'.

32.142.2.77 scrollbar_size() [1/2]

int Fl_Tree::scrollbar_size ( ) const

Gets the default size of scrollbars' troughs for this widget in pixels.  
If this value is zero (default), this widget will use the global Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)
### 32.142.2.78 scrollbar_size() [2/2]

```cpp
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>
</tr>
</thead>
<tbody>
<tr>
<td></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()`

### 32.142.2.79 select() [1/2]

```cpp
int Fl_Tree::select ( const char * path, int docallback = 1 )
```

Select the item specified by `path`.

Invokes the callback depending on the value of optional parameter `docallback`.

Handles calling `redraw()` if anything changed.

Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. `select("Holidays/12/25/2010").` The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- A flag that determines if the <code>callback()</code> is invoked or not:</td>
</tr>
<tr>
<td></td>
<td>• 0 - the <code>callback()</code> is not invoked</td>
</tr>
<tr>
<td></td>
<td>• 1 - the <code>callback()</code> is invoked if item changed state (default), <code>callback_reason()</code> will be <code>FL_TREE_REASON_SELECTED</code></td>
</tr>
</tbody>
</table>

#### Returns

- 1 : OK: item's state was changed
- 0 : OK: item was already selected, no change was made
- -1 : ERROR: item was not found

### 32.142.2.80 select() [2/2]

```cpp
int Fl_Tree::select ( Fl_Tree_Item * item, int docallback = 1 )
```

Generated by Doxygen
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

32.142.2.81 select_all()

int Fl_Tree::select_all ( Fl_Tree_Item ∗ item = 0, int docallback = 1 )
Select 'item' and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item that will be selected (along with all its children). If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>-- A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>

Returns

Count of how many items were actually changed to the selected state.

32.142.2.82 select_only()

int Fl_Tree::select_only ( Fl_Tree_Item ∗ selitem, int docallback = 1 )
Select only the specified item, deselecting all others that might be selected.
If 'selitem' is 0, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>selitem</th>
<th>The item to be selected. If NULL, first() is used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked for each item that changed state (default), 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.

32.142.2.83 select_toggle()

void Fl_Tree::select_toggle ( Fl_Tree_Item * item, int docallback = 1 )
Toggle the select state of the specified 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<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>

32.142.2.84 selectbox() [1/2]

Fl_Boxtype Fl_Tree::selectbox ( ) const
Sets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme()

32.142.2.85 selectbox() [2/2]

void Fl_Tree::selectbox ( Fl_Boxtype val )
Gets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme()
selectmode() [1/2]

`Fl_Tree::selectmode()` const

Gets the tree's current selection mode.
See `Fl_Tree_Select` for possible values.

selectmode() [2/2]

`void Fl_Tree::selectmode (Fl_Tree_Select val)`

Sets the tree's selection mode.
See `Fl_Tree_Select` for possible values.

set_item_focus()

`void Fl_Tree::set_item_focus (Fl_Tree_Item *item)`

Set the item that currently should have keyboard focus.
Handles calling `redraw()` to update the focus box (if it is visible).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item that should take focus. If NULL, none will have focus.</th>
</tr>
</thead>
</table>

show_item()

`void Fl_Tree::show_item (Fl_Tree_Item *item)`

Adjust the vertical scrollbar to show `item` at the top of the display IF it is currently off-screen (for instance `show_item_top()`).
If it is already on-screen, no change is made.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to be shown. If NULL, <code>first()</code> is used.</th>
</tr>
</thead>
</table>

See also

`show_item_top(), show_item_middle(), show_item_bottom()`

show_item()

`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, <code>first()</code> 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()

32.142.2.91  show_item_bottom()

void Fl_Tree::show_item_bottom ( 
    Fl_Tree_Item * item )

Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

Parameters

| in | item | The item to be shown. If NULL, first() is used. |

32.142.2.92  show_item_middle()

void Fl_Tree::show_item_middle ( 
    Fl_Tree_Item * item )

Adjust the vertical scrollbar so that 'item' is in the middle of the display.

Parameters

| in | item | The item to be shown. If NULL, first() is used. |

32.142.2.93  show_item_top()

void Fl_Tree::show_item_top ( 
    Fl_Tree_Item * item )

Adjust the vertical scrollbar so that 'item' is at the top of the display.

Parameters

| in | item | The item to be shown. If NULL, first() is used. |

32.142.2.94  show_self()

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

32.142.2.95  showcollapse() [1/2]

int Fl_Tree::showcollapse ( ) const

Returns 1 if the collapse icon is enabled, 0 if not.
See also

\texttt{showcollapse(int)}

### 32.142.2.96 \texttt{showcollapse() [2/2]}

```cpp
void Fl_Tree::showcollapse ( 
  int val )
```

Set if we should show the collapse icon or not.
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via \texttt{open()} and \texttt{close()}.

#### Parameters

| in  | val | 1: shows collapse icons (default), 0: hides collapse icons.
|-----|-----|------------------------------------------------------------------|

### 32.142.2.97 \texttt{showroot()}

```cpp
void Fl_Tree::showroot ( 
  int val )
```

Set if the root item should be shown or not.

#### Parameters

| in  | val | 1 – show the root item (default) 0 – hide the root item.
|-----|-----|--------------------------------------------------------------------------------|

### 32.142.2.98 \texttt{sortorder()}

```cpp
Fl_Tree_Sort Fl_Tree::sortorder ( ) const
```

Set the default sort order used when items are added to the tree.
See \texttt{Fl_Tree_Sort} for possible values.

### 32.142.2.99 \texttt{usericon() [1/2]}

```cpp
Fl_Image * Fl_Tree::usericon ( ) const
```

Returns the \texttt{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.

### 32.142.2.100 \texttt{usericon() [2/2]}

```cpp
void Fl_Tree::usericon ( 
  Fl_Image * val )
```

Sets the \texttt{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 \texttt{Fl_Tree_Item::usericon()} instead.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>– The new image to be used, or zero to disable user icons.</th>
</tr>
</thead>
</table>
32.142.2.101  vposition()  [1/2]
int Fl_Tree::vposition ( ) const
Returns the vertical scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the top edge of the screen.
See also
vposition(int), hposition(), hposition(int)

32.142.2.102  vposition()  [2/2]
void Fl_Tree::vposition ( int pos )
Sets the vertical scroll offset to position 'pos'.
The position is how many pixels of the tree are scrolled off the top edge of the screen.
Parameters
| in pos | The vertical position (in pixels) to scroll the tree to. |
See also
vposition(), hposition(), hposition(int)
The documentation for this class was generated from the following files:
- Fl_Tree.H
- Fl_Tree.cxx

32.143  Fl_Tree_Item Class Reference
Tree widget item.
#include <Fl_Tree_Item.H>

Public Member Functions
- void activate (int val=1)
  Change the item's activation state to the optionally specified 'val'.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, char **arr)
  Descend into the path specified by 'arr', and add a new child there.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, char **arr, Fl_Tree_Item *newitem)
  Descend into path specified by 'arr' and add 'newitem' there.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, const char *new_label)
  Add a new child to this item with the name 'new_label' and defaults from 'prefs'.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, const char *new_label, Fl_Tree_Item *newitem)
  Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.
- Fl_Tree_Item * child (int index)
  Return the child item for the given 'index'.
- const Fl_Tree_Item * child (int t) const
  Return the const child item for the given 'index'.
- int children () const
  Return the number of children this item has.
- void clear_children ()
  Clear all the children for this item.
• void close ()
  Close this item and all its children.

• void deactivate ()
  Deactivate the item; the callback() won't be invoked when clicked.

• Fl_Tree_Item * deparent (int index)
  Deparent child at index position 'pos'.

• int depth () const
  Returns how many levels deep this item is in the hierarchy.

• void deselect ()
  Disable the item's selection state.

• int deselect_all ()
  Deselect item and all its children.

• void draw (int X, int Y, int W, Fl_Tree_Item *itemfocus, int &tree_item_xmax, int lastchild=1, int render=1)
  Draw this item and its children.

• virtual int draw_item_content (int render)
  Draw the item content.

• int event_on_collapse_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'collapse' button of this item?

• int event_on_item (const Fl_Tree_Prefs &prefs) const
  Was event anywhere on the item?

• int event_on_label (const Fl_Tree_Prefs &prefs) const
  Was event on the label() of this item?

• int event_on_user_icon (const Fl_Tree_Prefs &prefs) const
  Was the event on the 'user icon' of this item, if any?

• int find_child (const char *name)
  Return the index of the immediate child of this item that has the label 'name'.

• int find_child (Fl_Tree_Item *item)
  Find the index number for the specified 'item' in the current item's list of children.

• Fl_Tree_Item * find_child_item (char **arr)
  Non-const version of Fl_Tree_Item::find_child_item(char **arr) const.

• const Fl_Tree_Item * find_child_item (char **arr) const
  Find child item by descending array 'arr' of names.

• Fl_Tree_Item * find_child_item (const char *name)
  Non-const version of Fl_Tree_Item::find_child_item(const char *name) const.

• const Fl_Tree_Item * find_child_item (const char *name) const
  Return the immediate child of current item that has the label 'name'.

• Fl_Tree_Item * find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0)
  Non-const version of Fl_Tree_Item::find_clicked(const Fl_Tree_Prefs &, int) const.

• const Fl_Tree_Item * find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0) const
  Find the item that the last event was over.

• Fl_Tree_Item * find_item (char **arr)
  Non-const version of Fl_Tree_Item::find_item(char **names) const.

• const Fl_Tree_Item * find_item (char **arr) const
  Find item by descending array of 'names'.

• Fl_Tree_Item (const Fl_Tree_Item *)
  Copy constructor.

• Fl_Tree_Item (const Fl_Tree_Prefs &prefs)
  Constructor.

• Fl_Tree_Item (Fl_Tree *)
  Constructor.

• int h () const
The item's height.

- int has_children() const
  See if this item has children.

- Fl_Tree_Item* insert(const Fl_Tree_Prefs &prefs, const char* new_label, int pos=0)
  Insert a new item named `new_label` into current item's children at a specified position `pos`.

- Fl_Tree_Item* insert_above(const Fl_Tree_Prefs &prefs, const char* new_label)
  Insert a new item named `new_label` above this item.

- char is_activated() const
  See if the item is activated.

- char is_active() const
  See if the item is activated. Alias for is_activated().

- int is_close() const
  See if the item is 'closed'.

- int is_open() const
  See if the item is 'open'.

- int is_root() const
  Is this item the root of the tree?

- char is_selected() const
  See if the item is selected.

- int is_visible() const
  See if the item is visible.

- const char* label() const
  Return the label.

- void label(const char* val)
  Set the label to `name`.

- int label_h() const
  The item's label height.

- int label_w() const
  The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.

- int label_x() const
  The item's label x position relative to the window.

- int label_y() const
  The item's label y position relative to the window.

- Fl_Color labelbgcolor() const
  Return item's label background text color.

- void labelbgcolor(Fl_Color val)
  Set item's label background color.

- Fl_Color labelcolor() const
  Return item's label text color. Alias for labelfgcolor() const.

- void labelcolor(Fl_Color val)
  Set item's label text color. Alias for labelfgcolor(Fl_Color).

- Fl_Color labelfgcolor() const
  Return item's label foreground text color.

- void labelfgcolor(Fl_Color val)
  Set item's label foreground text color.

- Fl_Font labelfont() const
  Get item's label font face.

- void labelfont(Fl_Font val)
  Set item's label font face.

- Fl_Fontsize labelsize() const
  Get item's label font size.
• void labelsize (Fl_Fontsize val)
  
  Set item's label font size.
• int move (Fl_Tree_Item *item, int op=0, int pos=0)
  
  Move the current item above/below/into the specified 'item', where 'op' determines the type of move:
• int move (int to, int from)
  
  Move an item within its parent using index numbers.
• int move_above (Fl_Tree_Item *item)
  
  Move the current item above the specified 'item'.
• int move_below (Fl_Tree_Item *item)
  
  Move the current item below the specified 'item'.
• int move_into (Fl_Tree_Item *item, int pos=0)
  
  Parent the current item as a child of the specified 'item'.

• Fl_Tree_Item * next ()
  
  Return the next item in the tree.
• Fl_Tree_Item * next_displayed (Fl_Tree_Prefs &prefs)
  
  Same as next_visible().
• Fl_Tree_Item * next_sibling ()
  
  Return this item's next sibling.
• Fl_Tree_Item * next_visible (Fl_Tree_Prefs &prefs)
  
  Return the next open(), visible() item.
• void open ()
  
  Open this item and all its children.
• void open_toggle ()
  
  Toggle the item's open/closed state.
• Fl_Tree_Item * parent ()
  
  Return the parent for this item. Returns NULL if we are the root.
• const Fl_Tree_Item * parent () const
  
  Return the const parent for this item. Returns NULL if we are the root.
• void parent (Fl_Tree_Item *val)
  
  Set the parent for this item.
• const Fl_Tree_Prefs & prefs () const
  
  Return the parent tree's prefs.
• Fl_Tree_Item * prev ()
  
  Return the previous item in the tree.
• Fl_Tree_Item * prev_displayed (Fl_Tree_Prefs &prefs)
  
  Same as prev_visible().
• Fl_Tree_Item * prev_sibling ()
  
  Return this item's previous sibling.
• Fl_Tree_Item * prev_visible (Fl_Tree_Prefs &prefs)
  
  Return the previous open(), visible() item.
• int remove_child (const char *new_label)
  
  Remove immediate child (and its children) by its label 'name'.
• int remove_child (Fl_Tree_Item *item)
  
  Remove 'item' from the current item's children.
• int reparent (Fl_Tree_Item *newchild, int index)
  
  Reparent specified item as a child of ourself at position 'pos'.
• Fl_Tree_Item * replace (Fl_Tree_Item *new_item)
  
  Replace the current item with a new item.
• Fl_Tree_Item * replace_child (Fl_Tree_Item *olditem, Fl_Tree_Item *newitem)
  
  Replace existing child 'olditem' with 'newitem'.
• void select (int val=1)
Change the item's selection state to the optionally specified 'val'.

- `int select_all ()`
  Select item and all its children.
- `void select_toggle ()`
  Toggle the item's selection state.
- `void show_self (const char *indent="") const`
  Print the tree as 'ascii art' to stdout.
- `int swap_children (Fl_Tree_Item *a, Fl_Tree_Item *b)`
  Swap two of our immediate children, given item pointers.
- `void swap_children (int ax, int bx)`
  Swap two of our children, given two child index values 'ax' and 'bx'.
- `Fl_Tree * tree ()`
  Return the tree for this item.
- `const Fl_Tree * tree () const`
  Return the tree for this item.
- `void update_prev_next (int index)`
  Update our _prev_sibling and _next_sibling pointers to point to neighbors given index as being our current position in the parent's item array.
- `void * user_data () const`
  Retrieve the user-data value that has been assigned to the item.
- `void user_data (void *data)`
  Set a user-data value for the item.
- `Fl_Image * userdeicon () const`
  Return the deactivated version of the user icon, if any.
- `void userdeicon (Fl_Image *val)`
  Set the usericon to draw when the item is deactivated.
- `Fl_Image * usericon () const`
  Get the item's user icon as an Fl_Image. Returns '0' if disabled.
- `void usericon (Fl_Image *val)`
  Set the item's user icon to an Fl_Image.
- `int visible () const`
  See if the item is visible. Alias for is_visible().
- `int visible_r () const`
  See if item and all its parents are open() and visible().
- `int w () const`
  The entire item's width to right edge of Fl_Tree's inner width within scrollbars.
- `Fl_Widget * widget () const`
  Return FLTK widget assigned to this item.
- `void widget (Fl_Widget *val)`
  Assign an FLTK widget to this item.
- `int x () const`
  The item's x position relative to the window.
- `int y () const`
  The item's y position relative to the window.
Protected Member Functions

- void _Init (const Fl_Tree_Prefs &prefs, Fl_Tree *tree)
- int calc_item_height (const Fl_Tree_Prefs &prefs) const
  Return the item’s 'visible' height.
- void draw_horizontal_connector (int x1, int x2, int y, const Fl_Tree_Prefs &prefs)
  Internal: Horizontal connector line based on preference settings.
- void draw_vertical_connector (int x, int y1, int y2, const Fl_Tree_Prefs &prefs)
  Internal: Vertical connector line based on preference settings.
- Fl_Color drawbgcolor () const
  Returns the recommended background color used for drawing this item.
- Fl_Color drawfgcolor () const
  Returns the recommended foreground color used for drawing this item.
- void hide_widgets ()
  Internal: Hide the FLTK widget() for this item and all children.
- int is_flag (unsigned short val) const
  See if flag set. Returns 0 or 1.
- void recalc_tree ()
  Call this when our geometry is changed.
- void set_flag (unsigned short flag, int val)
  Set a flag to an on or off value. val is 0 or 1.
- void show_widgets ()
  Internal: Show the FLTK widget() for this item and all children.

32.143.1 Detailed Description

Tree widget item.
This class is a single tree item, and manages all of the item’s attributes. Fl_Tree_Item is used by Fl_Tree, which is comprised of many instances of Fl_Tree_Item.
Fl_Tree_Item is hierarchical; it dynamically manages an Fl_Tree_Item_Array of children that are themselves instances of Fl_Tree_Item. Each item can have zero or more children. When an item has children, close() and open() can be used to hide or show them.
Items have their own attributes; font size, face, color. Items maintain their own hierarchy of children.
When you make changes to items, you'll need to tell the tree to redraw() for the changes to show up.
New 1.3.3 ABI feature: You can define custom items by either adding a custom widget to the item with Fl_Tree_Item::widget(), or override the draw_item_content() method if you want to just redefine how the label is drawn.
The following shows the Fl_Tree_Item’s dimensions, useful when overriding the draw_item_content() method:

![Figure 32.56 Fl_Tree_Item’s internal dimensions.](image)

32.143.2 Constructor & Destructor Documentation

Generated by Doxygen
32.143.2.1 Fl_Tree_Item() [1/2]

Fl_Tree_Item::Fl_Tree_Item (const Fl_Tree_Prefs & prefs)

Constructor.
Makes a new instance of Fl_Tree_Item using defaults from 'prefs'.

**Deprecated** in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree*) for proper horizontal scrollbar behavior.

32.143.2.2 Fl_Tree_Item() [2/2]

Fl_Tree_Item::Fl_Tree_Item (Fl_Tree * tree)

Constructor.
Makes a new instance of Fl_Tree_Item for 'tree'.
This must be used instead of the older, deprecated Fl_Tree_Item(Fl_Tree_Prefs) constructor for proper horizontal scrollbar calculation.

Version

1.3.3 ABI feature

32.143.3 Member Function Documentation

32.143.3.1 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.

32.143.3.2 add() [1/4]

```cpp
Fl_Tree_Item * Fl_Tree_Item::add (const Fl_Tree_Prefs & prefs, char ** arr)
```

Descend into 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

32.143.3.3 add() [2/4]

```cpp
Fl_Tree_Item * Fl_Tree_Item::add (const Fl_Tree_Prefs & prefs, char ** arr, Fl_Tree_Item * newitem)
```

Descend into path specified by 'arr' and add 'newitem' there.
Should be used only by Fl_Tree's internals. If item is NULL, a new item is created. Adds the item based on the value of prefs.sortorder().
Returns

the item added.

Version

1.3.3 ABI feature

32.143.3.4 add() [3/4]

Fl_Tree_Item * Fl_Tree_Item::add (const Fl_Tree_Prefs & prefs,
   const char * new_label )

Add a new child to this item with the name 'new_label' and defaults from 'prefs'. An internally managed copy is made of the label string. Adds the item based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.0 release

32.143.3.5 add() [4/4]

Fl_Tree_Item * Fl_Tree_Item::add (const Fl_Tree_Prefs & prefs,
   const char * new_label,
   Fl_Tree_Item * item )

Add 'item' as immediate child with 'new_label' and defaults from 'prefs'. If 'item' is NULL, a new item is created. An internally managed copy is made of the label string. Adds the item based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.3

32.143.3.6 calc_item_height()

int Fl_Tree_Item::calc_item_height (const Fl_Tree_Prefs & prefs ) const [protected]

Return the item's 'visible' height.

Takes into account the item's:

• visibility (if !is_visible(), returns 0)
• labelfont() height: if label() != NULL
• widget() height: if widget() != NULL
• openicon() height (if not NULL)
• usericon() height (if not NULL) Does NOT include Fl_Tree::linespacing();

Returns

maximum pixel height
32.143.3.7 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'.

32.143.3.8 deactivate()

void Fl_Tree_Item::deactivate ( ) [inline]
Deactivate the item; the callback() won't be invoked when clicked.
Same as activate(0)

32.143.3.9 deparent()

Fl_Tree_Item * Fl_Tree_Item::deparent ( int pos )
Deparent child at index position 'pos'.
This creates an "orphaned" item that is still allocated, but has no parent or siblings. Normally the caller would want
to immediately reparent the orphan elsewhere.
A successfully orphaned item will have its parent() and prev_sibling()/next_sibling() set to NULL.
Returns
• pointer to orphaned item on success
• NULL on error (could not deparent the item)

See also
reparent()

32.143.3.10 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.

32.143.3.11 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".

32.143.3.12 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

| in | X | Horizontal position for item being drawn |

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out Y</td>
<td>Vertical position for item being drawn, returns new position for next item</td>
</tr>
<tr>
<td>in W</td>
<td>Recommended width for item</td>
</tr>
<tr>
<td>in itemfocus</td>
<td>The tree's current focus item (if any)</td>
</tr>
<tr>
<td>in,out 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 lastchild</td>
<td>Is this item the last child in a subtree?</td>
</tr>
<tr>
<td>in 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

32.143.3.13 draw_horizontal_connector()

```cpp
void Fl_Tree_Item::draw_horizontal_connector ( int x1, int x2, int y,
                                           const Fl_Tree_Prefs & prefs ) [protected]
```

Internal: Horizontal connector line based on preference settings.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>The left hand X position of the horizontal connector</td>
</tr>
<tr>
<td>x2</td>
<td>The right hand X position of the horizontal connector</td>
</tr>
<tr>
<td>y</td>
<td>The vertical position of the horizontal connector</td>
</tr>
<tr>
<td>prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>

32.143.3.14 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()); fl_drawf(label()); // use item's label font/size
      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
  }
};
```

Generated by Doxygen
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:

```c
// 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:

```c
// 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

#### 32.143.3.15 `draw_vertical_connector()`

```c
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

| in  | x       | The x position of the vertical connector |
| in  | y1      | The top of the vertical connector       |
| in  | y2      | The bottom of the vertical connector    |
| in  | prefs   | The `Fl_Tree` prefs                     |

#### 32.143.3.16 `drawbgcolor()`

```c
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
32.143.3.17 drawfgcolor()

`Fl_Color Fl_Tree_Item::drawfgcolor ( ) const [protected]`

Returns the recommended foreground color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI ABI

32.143.3.18 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

32.143.3.19 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.

32.143.3.20 find_child_item () [1/2]

```cpp
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
32.143.3.21 find_child_item() [2/2]

const Fl_Tree_Item * Fl_Tree_Item::find_child_item (const char * name) const

Return the *immediate* child of current item that has the label 'name'.

Returns

const found item, or 0 if not found.

Version

1.3.3

32.143.3.22 find_clicked()

const Fl_Tree_Item * Fl_Tree_Item::find_clicked (const Fl_Tree_Prefs & prefs, int yonly = 0) const

Find the item that the last event was over. If 'yonly' is 1, only check event's y value, don't care about x.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>prefs</th>
<th>The parent tree's Fl_Tree_Prefs</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>yonly</td>
<td>– 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X.</td>
</tr>
</tbody>
</table>

Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

32.143.3.23 find_item()

const Fl_Tree_Item * Fl_Tree_Item::find_item (char ** names) const

Find item by descending array of 'names'. Includes self in search. Only Fl_Tree should need this method. Use Fl_Tree::find_item() instead.

Returns

const item, or 0 if not found

32.143.3.24 hide_widgets()

void Fl_Tree_Item::hide_widgets ( ) [protected]

Internal: Hide the FLTK widget() for this item and all children. Used by close() to hide widgets.

32.143.3.25 insert()

Fl_Tree_Item * Fl_Tree_Item::insert (const Fl_Tree_Prefs & prefs,
const char * new_label,
int pos = 0)

Insert a new item named 'new_label' into current item's children at a specified position 'pos'.
If pos is out of range the new item is

- prepended if pos < 0 or
- appended if pos > item->children().

Returns
the new item inserted

See also
Fl_Tree::insert()

32.143.3.26 insert_above()

Fl_Tree_Item * Fl_Tree_Item::insert_above (const Fl_Tree_Prefs & prefs,
const char * new_label)

Insert a new item named 'new_label' above this item.

Returns
the new item inserted, or 0 if an error occurred.

32.143.3.27 label()

void Fl_Tree_Item::label (const char * name)

Set the label to 'name'.
Makes and manages an internal copy of 'name'.

32.143.3.28 label_h()

int Fl_Tree_Item::label_h ( ) const [inline]
The item's label height.

Version
1.3.3

32.143.3.29 label_w()

int Fl_Tree_Item::label_w ( ) const [inline]
The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.

Version
1.3.3
32.143.3.30 label_x()

int Fl_Tree_Item::label_x ( ) const [inline]
The item's label x position relative to the window.

Version
1.3.3

32.143.3.31 label_y()

int Fl_Tree_Item::label_y ( ) const [inline]
The item's label y position relative to the window.

Version
1.3.3

32.143.3.32 labelbgcolor() [1/2]

Fl_Color Fl_Tree_Item::labelbgcolor ( ) const [inline]
Return item's label background text color.
If the color is 0xffffffff, the default behavior is the parent tree's bg color will be used. (An overloaded draw_item_content() can override this behavior.)

32.143.3.33 labelbgcolor() [2/2]

doctorb

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.

32.143.3.34 move() [1/2]

int Fl_Tree_Item::move ( Fl_Tree_Item * item,
                      int op = 0,
                      int pos = 0 )
Move the current item above/below/into the specified 'item', where 'op' determines the type of move:

• 0: move above 'item' ('pos' ignored)
• 1: move below 'item' ('pos' ignored)
• 2: move into 'item' as a child (at optional position 'pos')

..and 'pos' determines an optional index position after the move.

Returns
0 on success. a negative number on error:

• -1: one of the items has no parent
• -2: item's index could not be determined
• -3: bad 'op'
• -4: index range error
• -5: could not deparent
• -6: could not reparent at 'pos'
• (Other return values reserved for future use.)
See also

`move_above(), move_below(), move_into(), move(int,int)`

### 32.143.3.35 move()

```cpp
class Fl_Tree_Item {
public:
    int move (int to, int from) {
        // Move an item within its parent using index numbers.
        // Item is moved 'to' its new position 'from' its old position.
        // Returns
        // • 0: Success
        // • -1: range error (e.g. if 'to' or 'from' out of range).
        // • (Other return values reserved for future use)
    }
};
```

### 32.143.3.36 move_above()

```cpp
class Fl_Tree_Item {
public:
    int 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.
    }
};
```

### 32.143.3.37 move_below()

```cpp
class Fl_Tree_Item {
public:
    int 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.
    }
};
```
32.143.3.38 move_into()

```cpp
int Fl_Tree_Item::move_into (  
    Fl_Tree_Item * item,  
    int pos = 0  
)
```

Parent the current item as a child of the specified 'item'.
This is the equivalent of calling move(item,2,pos).

Returns

0 on success.
On error returns a negative value; see move(Fl_Tree_Item*,int,int) for possible error codes.

See also

move_above(), move_below(), move(int,int), move(Fl_Tree_Item*,int,int)

32.143.3.39 next()

```cpp
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.

32.143.3.40 next_displayed()

```cpp
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

32.143.3.41 next_sibling()

```cpp
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.

32.143.3.42 next_visible()

```cpp
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

32.143.3.43 parent()

void Fl_Tree_Item::parent ( 
        Fl_Tree_Item ∗ val ) [inline]

Set the parent for this item.
Should only be used by Fl_Tree's internals.

32.143.3.44 prefs()

const Fl_Tree_Prefs & Fl_Tree_Item::prefs ( ) const

Return the parent tree's prefs.

Returns

a reference to the parent tree's Fl_Tree_Prefs

Version

1.3.3 ABI feature

32.143.3.45 prev()

Fl_Tree_Item ∗ Fl_Tree_Item::prev ( )

Return the previous item in the tree.
This method can be used to walk the tree backwards. For an example of how to use this method, see Fl_Tree::last().

Returns

the previous item in the tree, or 0 if there's no item above this one (hit the root).

32.143.3.46 prev_displayed()

Fl_Tree_Item ∗ Fl_Tree_Item::prev_displayed ( 
        Fl_Tree_Prefs & prefs )

Same as prev_visible().

Deprecated in 1.3.3 for confusing name, use prev_visible()

32.143.3.47 prev_sibling()

Fl_Tree_Item ∗ Fl_Tree_Item::prev_sibling ( )

Return this item's previous sibling.
Moves to the previous item above us at the same level (sibling). Use this to move up the tree without changing depth().

Returns

This item's previous sibling, or 0 if none.
32.143.3.48 prev_visible()

`Fl_Tree_Item * Fl_Tree_Item::prev_visible ( Fl_Tree_Prefs & prefs )`

Return the previous open(), visible() item.
(If this item above us has children and is closed, its children are skipped)
This method can be used to walk the tree backward, skipping items that are not currently open/visible to the user.

Returns
the previous open() visible() item above us, or 0 if there's no more items.

32.143.3.49 recalc_tree()

`void Fl_Tree_Item::recalc_tree ( ) [protected]`

Call this when our geometry is changed.
(Font size, label contents, etc) Schedules tree to recalculate itself, as changes to us may affect tree widget's scrollbar visibility and tab sizes.

Version

1.3.3 ABI

32.143.3.50 remove_child() [1/2]

`int Fl_Tree_Item::remove_child ( const char * name )`

Remove immediate child (and its children) by its label 'name'.
If more than one item matches 'name', only the first matching item is removed.

Parameters

| in  | name     | The label name of the immediate child to remove |

Returns

0 if removed, -1 if not found.

Version

1.3.3

32.143.3.51 remove_child() [2/2]

`int Fl_Tree_Item::remove_child ( Fl_Tree_Item * item )`

Remove 'item' from the current item's children.

Returns

0 if removed, -1 if item not an immediate child.

32.143.3.52 reparent()

`int Fl_Tree_Item::reparent ( Fl_Tree_Item * newchild, int pos )`

Generated by Doxygen
Reparent specified item as a child of ourself at position 'pos'. Typically 'newchild' was recently orphaned with deparent().

Returns

- 0: on success
- -1: on error (e.g. if 'pos' out of range) with no changes made.

See also

deparent()

### 32.143.3.53 replace()

```cpp
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

### 32.143.3.54 replace_child()

```cpp
Fl_Tree_Item * Fl_Tree_Item::replace_child (Fl_Tree_Item * olditem, Fl_Tree_Item * newitem)
```

Replace existing child 'olditem' with 'newitem'. The 'olditem' is destroyed if successful. Can be used to put custom items (derived from Fl_Tree_Item) into the tree. No checks are made to see if an item with the same name exists.

Parameters

- `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

32.143.3.55  select()

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.

32.143.3.56  select_all()

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".

32.143.3.57  show_self()

void Fl_Tree_Item::show_self (  
    const char * indent = "" ) const

Print the tree as 'ascii art' to stdout.
Used mainly for debugging.

32.143.3.58  show_widgets()

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()

32.143.3.59  swap_children() [1/2]

int Fl_Tree_Item::swap_children (  
    Fl_Tree_Item * a,  
    Fl_Tree_Item * b )

Swap two of our immediate children, given item pointers.
Use e.g. for sorting.
This method is SLOW because it involves linear lookups.
For speed, use swap_children(int,int) instead.

Parameters

| 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.

32.143.3.60  swap_children() [2/2]

void Fl_Tree_Item::swap_children (  

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ax, bx</code></td>
<td>the index of the items to swap</td>
</tr>
</tbody>
</table>

32.143.3.61  tree() [1/2]

`Fl_Tree* Fl_Tree_Item::tree ( ) [inline]`

Return the tree for this item.

Version

1.3.4

32.143.3.62  tree() [2/2]

`const Fl_Tree* Fl_Tree_Item::tree ( ) const [inline]`

Return the tree for this item.

Version

1.3.3

32.143.3.63  update_prev_next()

`void Fl_Tree_Item::update_prev_next ( int index )`

Update our _prev_sibling and _next_sibling pointers to point to neighbors given `index` as being our current position in the parent's item array.
Call this whenever items in the array are added/removed/moved/swapped/etc.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>Our index# in the parent. Special case if index=-1: become an orphan; null out all parent/sibling associations.</td>
</tr>
</tbody>
</table>

32.143.3.64  userdeicon() [1/2]

`Fl_Image* Fl_Tree_Item::userdeicon ( ) const [inline]`

Return the deactivated version of the user icon, if any.
Returns 0 if none.

32.143.3.65  userdeicon() [2/2]

`void Fl_Tree_Item::userdeicon ( Fl_Image* val ) [inline]`

Set the usericon to draw when the item is deactivated.
Use '0' to disable. No internal copy is made; caller must manage icon's memory.

To create a typical 'grayed out' version of your usericon image, you can do the following:

```c++
// Create tree + usericon for items
Fl_Tree *tree = new Fl_Tree(...);
Fl_Image *usr_icon = new Fl_Pixmap(...); // your usericon
Fl_Image *de_icon = usr_icon->copy(); // make a copy, and..
de_icon->inactive(); // make it 'grayed out'
...
for ( ... ) { // item loop..
    item = tree->add("..."); // create new item
    item->usericon(usr_icon); // assign usericon to items
    item->userdeicon(de_icon); // assign userdeicon to items
    ..
}
```

In the above example, the app should 'delete' the two icons when they're no longer needed (e.g. after the tree is destroyed)

Version
1.3.4

32.143.3.66 usericon()

```c++
void Fl_Tree_Item::usericon (Fl_Image *val) [inline]
```

Set the item's user icon to an Fl_Image.

Use '0' to disable. No internal copy is made, caller must manage icon's memory.

Note, if you expect your items to be deactivated(), use userdeicon(Fl_Image+) to set up a 'grayed out' version of your icon to be used for display.

See also
userdeicon(Fl_Image+)

32.143.3.67 visible_r()

```c++
int Fl_Tree_Item::visible_r ( ) const
```

See if item and all its parents are open() and visible().

Returns
1 – item and its parents are open() and visible() 0 – item (or one of its parents) are invisible or close()ed.

The documentation for this class was generated from the following files:
- Fl_Tree_Item.H
- Fl_Tree_Item.cxx

32.144 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.
954 Class Documentation

- **Fl_Tree_Item_Array (const Fl_Tree_Item_Array *o)**
  
  Copy constructor. Makes new copy of array, with new instances of each item.

- **Fl_Tree_Item_Array (int new_chunksize=10)**
  
  Constructor; creates an empty array.

- **void insert (int pos, Fl_Tree_Item *new_item)**
  
  Insert an item at index position pos.

- **int manage_item_destroy () const**

  Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.

- **void manage_item_destroy (int val)**

  Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.

- **int move (int to, int from)**

  Move item at 'from' to new position 'to' in the array.

- **Fl_Tree_Item * operator[] (int i)**

  Return the item and index i.

- **const Fl_Tree_Item * operator[] (int i) const**

  Const version of operator[](int i)

- **int remove (Fl_Tree_Item *item)**

  Remove the item from the array.

- **void remove (int index)**

  Remove the item at.

- **int reparent (Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos)**

  Reparent specified item as a child of ourself.

- **void replace (int pos, Fl_Tree_Item *new_item)**

  Replace the item at index with newitem.

- **void swap (int ax, int bx)**

  Swap the two items at index positions ax and bx.

- **int total () const**

  Return the total items in the array, or 0 if empty.

- **~Fl_Tree_Item_Array ()**

  Destructor. Calls each item's destructor, destroys internal_items array.

### 32.144.1 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).

### 32.144.2 Constructor & Destructor Documentation

#### 32.144.2.1 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.

### 32.144.3 Member Function Documentation
32.144.3.1 add()

```cpp
void Fl_Tree_Item_Array::add (
    Fl_Tree_Item * val )
```

Add an item to the end of the array.

Assumes the item was created with 'new', and will remain allocated. Fl_Tree_Item_Array will handle the item's destructor when the array is cleared or the item remove()’ed.

32.144.3.2 clear()

```cpp
void Fl_Tree_Item_Array::clear ( )
```

Clear the entire array.

Each item will be deleted (destructors will be called), and the array will be cleared. total() will return 0.

32.144.3.3 deparent()

```cpp
int Fl_Tree_Item_Array::deparent (
    int pos )
```

Deparent item at 'pos' from our list of children. Similar to a remove() without the destruction of the item. This creates an orphaned item (still allocated, has no parent) which soon after is typically reparented elsewhere.

\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)

32.144.3.4 insert()

```cpp
void Fl_Tree_Item_Array::insert (
    int pos,
    Fl_Tree_Item * new_item )
```

Insert an item at index position pos.

Handles enlarging array if needed, total increased by 1. If \p pos \textgreater= total(), the item is appended to the array. If \p pos \textless 0, the item is prepended (works like pos == 0).

32.144.3.5 manage_item_destroy()

```cpp
void Fl_Tree_Item_Array::manage_item_destroy ( [inline]
    int val )
```

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.

32.144.3.6 move()

```cpp
int Fl_Tree_Item_Array::move ( [inline]
    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'.

Generated by Doxygen
Returns
0 on success, -1 on range error (e.g. if 'to' or 'from' out of range)

32.144.3.7 remove() [1/2]

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.

32.144.3.8 remove() [2/2]

void Fl_Tree_Item_Array::remove (int index)
Remove the item at.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
<th>from the array.</th>
</tr>
</thead>
</table>
|     |       | The item will be delete’d (if non-NULL), so its destructor will be called.

32.144.3.9 reparent()

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)

32.144.3.10 replace()

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

32.145 Fl_Tree_Prefs Class Reference

Tree widget's preferences.
#include <Fl_Tree_Prefs.H>
Public Member Functions

- **FL_Image * closedeicon () const**
  
  Return the deactivated version of the close icon, if any.

- **FL_Image * closeicon () const**
  
  Gets the default 'close' icon Returns the FL_Image* of the icon, or 0 if none.

- **void closeicon (FL_Image *val)**
  
  Sets the icon to be used as the 'close' icon.

- **FL_Color connectorcolor () const**
  
  Get the connector color used for tree connection lines.

- **void connectorcolor (FL_Color val)**
  
  Set the connector color used for tree connection lines.

- **FL_Tree_Connector connectorstyle () const**
  
  Get the connector style.

- **void connectorstyle (FL_Tree_Connector val)**
  
  Set the connector style.

- **void connectorstyle (int val)**
  
  Set the connector style [integer].

- **int connectorwidth () const**
  
  Get the tree connection line's width.

- **void connectorwidth (int val)**
  
  Set the tree connection line's width.

- **void do_item_draw_callback (FL_Tree_Item *o) const**

- **FL_Tree_Prefs ()**
  
  FL_Tree_Prefs constructor.

- **FL_Tree_Item_Draw_Callback * item_draw_callback () const**

- **void item_draw_callback (FL_Tree_Item_Draw_Callback *cb, void *data=0)**

- **FL_Tree_Item_Draw_Mode item_draw_mode () const**
  
  Get the 'item draw mode' used for the tree.

- **void item_draw_mode (FL_Tree_Item_Draw_Mode val)**
  
  Set the 'item draw mode' used for the tree to val.

- **void * item_draw_user_data () const**

- **FL_Color item_labelbgcolor () const**
  
  Get the default label background color.

- **void item_labelbgcolor (FL_Color val)**
  
  Set the default label background color.

- **FL_Color item_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)**
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_Im * opendeicon () const**
  
  Return the deactivated version of the open icon, if any.

- **Fl_Im * openicon () const**
  
  Get the current default 'open' icon.

- **void openicon (Fl_Im * val)**
  
  Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.

- **Fl_Boxtype selectbox () const**
  
  Get the default selection box's box drawing style as an Fl_Boxtype.

- **void selectbox (Fl_Boxtype val)**
  
  Set the default selection box's box drawing style to `val`.  

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.

### 32.145.1 Detailed Description

Tree widget's preferences.

*Fl_Tree’s Preferences class.*

This class manages the Fl_Tree’s defaults. You should probably be using the methods in Fl_Tree instead of trying to accessing tree's preferences settings directly.

### 32.145.2 Member Function Documentation

#### 32.145.2.1 closedeicon()

```cpp
Fl_Image * Fl_Tree_Prefs::closedeicon ( ) const [inline]
```

Return the deactivated version of the close icon, if any.

Returns 0 if none.
closeicon()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>– The new image, or zero to use the default [+] icon.</td>
<td></td>
</tr>
</tbody>
</table>

### 32.145.2.3 item_draw_mode()

```cpp
void Fl_TreePrefs::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.

### 32.145.2.4 item_labelbgcolor() [1/2]

```cpp
Fl_Color Fl_TreePrefs::item_labelbgcolor () const [inline]
```

Get the default label background color. This returns the `Fl_Tree::color()` unless `item_labelbgcolor()` has been set explicitly.

### 32.145.2.5 item_labelbgcolor() [2/2]

```cpp
void Fl_TreePrefs::item_labelbgcolor (Fl_Color val) [inline]
```

Set the default label background color. Once set, overrides the default behavior of using `Fl_Tree::color()`.

### 32.145.2.6 marginbottom()

```cpp
int Fl_TreePrefs::marginbottom () const [inline]
```

Get the bottom margin's value in pixels. This is the extra distance the vertical scroller lets you travel.

### 32.145.2.7 opendeicon()

```cpp
Fl_Image* Fl_TreePrefs::opendeicon () const [inline]
```

Return the deactivated version of the open icon, if any. Returns 0 if none.

### 32.145.2.8 openicon() [1/2]

```cpp
Fl_Image* Fl_TreePrefs::openicon () const [inline]
```

Get the current default 'open' icon. Returns the `Fl_Image*` of the icon, or 0 if none.

### 32.145.2.9 openicon() [2/2]

```cpp
void Fl_TreePrefs::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

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>– The new image, or zero to use the default [+] icon.</td>
<td></td>
</tr>
</tbody>
</table>
32.145.2.10  selectmode()

void Fl_Tree_Prefs::selectmode (  
    Fl_Tree_Select  val  )  [inline]

Set the selection mode used for the tree to val.
This affects how items in the tree are selected when clicked on and dragged over by the mouse. See Fl_Tree_Select for possible values.

32.145.2.11  showcollapse()

void Fl_Tree_Prefs::showcollapse (  
    int  val  )  [inline]

Set if we should show the collapse icon or not.
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>1: shows collapse icons (default), 0: hides collapse icons.</th>
</tr>
</thead>
</table>

32.145.2.12  showroot()

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>

32.145.2.13  sortorder()

void Fl_Tree_Prefs::sortorder (  
    Fl_Tree_Sort  val  )  [inline]

Set the default sort order value.
Defines the order new items appear when add()ed to the tree. See Fl_Tree_Sort for possible values.

32.145.2.14  userdeicon()

Fl_Image* Fl_Tree_Prefs::userdeicon ( ) const  [inline]

Return the deactivated version of the user icon, if any.
Returns 0 if none.

The documentation for this class was generated from the following files:

- Fl_Tree_Prefs.H
- Fl_Tree_Prefs.cxx

32.146  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:
Public Member Functions

- void **bounds** (double a, double b)
  
  Sets the minimum (a) and maximum (b) values for the valuator widget.

- double **clamp** (double)
  
  Clamps the passed value to the valuator range.

- virtual int **format** (char ∗)
  
  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- double **increment** (double, int)
  
  Adds n times the step value to the passed value.

- double **maximum** () const
  
  Gets the maximum value for the valuator.

- void **maximum** (double a)
  
  Sets the maximum value for the valuator.

- double **minimum** () const
  
  Gets the minimum value for the valuator.

- void **minimum** (double a)
  
  Sets the minimum value for the valuator.

- void **precision** (int digits)
  
  Sets the step value to 1.0 / 10^digits.

- void **range** (double a, double b)
  
  Sets the minimum and maximum values for the valuator.

- void **round** (double)
  
  Round the passed value to the nearest step increment.

- double **step** () const
  
  Gets or sets the step value.

- void **step** (double a, int b)
  
  See double Fl_Valuator::step() const

- void **step** (double s)
  
  See double Fl_Valuator::step() const.

- void **step** (int a)
  
  See double Fl_Valuator::step() const

- double **value** () const
  
  Gets the floating point(double) value.

- int **value** (double)
  
  Sets the current value.
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

32.146.1 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 32.57 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.
32.146.2 Constructor & Destructor Documentation

32.146.2.1 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.

32.146.3 Member Function Documentation

32.146.3.1 format()

int Fl_Valuator::format (  
    char * buffer ) [virtual]
Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.  
The actual format used depends on the current step value. If the step value has been set to zero then a %g format  
is used. If the step value is non-zero, then a %.f format is used, where the precision is calculated to show sufficient  
digits for the current step value. An integer step value, such as 1 or 1.0, gives a precision of 0, so the formatted  
value will appear as an integer.  
This method is used by the Fl_Valuator... group of widgets to format the current value into a text string. The return  
value is the length of the formatted text. The formatted value is written into buffer. buffer should have space  
for at least 128 bytes.  
You may override this function to create your own text formatting.

32.146.3.2 increment()

double Fl_Valuator::increment (  
    double v,  
    int n )
Adds n times the step value to the passed value.  
If step was set to zero it uses fabs(maximum() - minimum()) / 100.

32.146.3.3 maximum() [1/2]

double Fl_Valuator::maximum ( ) const [inline]
Gets the maximum value for the valuator.

32.146.3.4 maximum() [2/2]

void Fl_Valuator::maximum (  
    double a ) [inline]
Sets the maximum value for the valuator.

32.146.3.5 minimum() [1/2]

double Fl_Valuator::minimum ( ) const [inline]
Gets the minimum value for the valuator.
32.146.3.6  minimum() [2/2]

void Fl_Valuator::minimum ( double a ) [inline]
Sets the minimum value for the valuator.

32.146.3.7  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. \(1.0/1 = 1\).

32.146.3.8  range()

void Fl_Valuator::range ( double a, double b ) [inline]
Sets the minimum and maximum values for the valuator. When the user manipulates the widget, the value is limited to this range. This clamping is done after rounding to the step value (this makes a difference if the range is not a multiple of the step). The minimum may be greater than the maximum. This has the effect of “reversing” the object so the larger values are in the opposite direction. This also switches which end of the filled sliders is filled. Some widgets consider this a “soft” range. This means they will stop at the range, but if the user releases and grabs the control again and tries to move it further, it is allowed. The range may affect the display. You must \text{redraw()} the widget after changing the range.

32.146.3.9  round()

double Fl_Valuator::round ( double v )
Round the passed value to the nearest step increment. Does nothing if step is zero.

32.146.3.10  step()

double Fl_Valuator::step ( ) const [inline]
Gets or sets the step value. As the user moves the mouse the value is rounded to the nearest multiple of the step value. This is done \text{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.

32.146.3.11  value() [1/2]

double Fl_Valuator::value ( ) const [inline]
Gets the floating point(double) value. See int value(double)
32.146.3.12 value() [2/2]

```cpp
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

32.147 Fl_Value_Input Class Reference

The Fl_Value_Input widget displays a numeric value.

```cpp
#include <Fl_Value_Input.H>
```

Inheritance diagram for Fl_Value_Input:

```
Fl_Widget
    Fl_Valuator
        Fl_Value_Input
```

Public Member Functions

- `Fl_Color cursor_color () const`
  
  Gets the color of the text cursor.

- `void cursor_color (Fl_Color n)`
  
  Sets the color of the text cursor.

- `Fl_Value_Input (int x, int y, int w, int h, const char *l=0)`
  
  Creates a new Fl_Value_Input widget using the given position, size, and label string.

- `int handle (int)`
  
  Handles the specified event.

- `void resize (int, int, int, int)`
  
  Changes the size or position of the widget.

- `int shortcut () const`
  
  Returns the current shortcut key for the Input.

- `void shortcut (int s)`
  
  Sets the shortcut key to `s`.

- `char soft () const`
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- `void soft (char s)`
  
  See void Fl_Value_Input::soft(char s)

- `Fl_Color textcolor () const`
  
  Gets the color of the text in the value box.

- `void textcolor (Fl_Color n)`
  
  Sets the color of the text in the value box.

- `Fl_Font textfont () const`

Generated by Doxygen
**Public Attributes**
- `Fl_Input` *input*

**Protected Member Functions**
- `void draw ()`
  
  *Draws the widget.*

**Additional Inherited Members**

### 32.147.1 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.

![Fl_Value_Input](image)

**Figure 32.58 Fl_Value_Input**

See also

`Fl_Widget::shortcut_label(int)`

### 32.147.2 Constructor & Destructor Documentation

#### 32.147.2.1 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.
### 32.147.3 Member Function Documentation

#### 32.147.3.1 cursor_color() [1/2]

```cpp
Fl_Color Fl_Value_Input::cursor_color ( ) const [inline]
```

Gets the color of the text cursor.
The text cursor is black by default.

#### 32.147.3.2 cursor_color() [2/2]

```cpp
void Fl_Value_Input::cursor_color ( Fl_Color n ) [inline]
```

Sets the color of the text cursor.
The text cursor is black by default.

#### 32.147.3.3 draw()

```cpp
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`.

#### 32.147.3.4 handle()

```cpp
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**

<table>
<thead>
<tr>
<th><strong>in event</strong></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`.

#### 32.147.3.5 resize()

```cpp
void Fl_Value_Input::resize ( )
```

Generated by Doxygen
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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y new position relative to the parent window</td>
</tr>
<tr>
<td>in</td>
<td>w,h new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`

Reimplemented from `Fl_Widget`.

### 32.147.3.6 shortcut() [1/2]

```cpp
int Fl_Value_Input::shortcut ( ) const [inline]
```

Returns the current shortcut key for the Input.

See also

- `Fl_Value_Input::shortcut(int)`

### 32.147.3.7 shortcut() [2/2]

```cpp
void Fl_Value_Input::shortcut ( int s ) [inline]
```

Sets the shortcut key to `s`.
Setting this overrides the use of `&` in the `label()`. The value is a bitwise OR of a key and a set of shift flags, for example `FL_ALT | 'a'`, `FL_ALT | (FL_F + 10)`, or just `a`. A value of 0 disables the shortcut.
The key can be any value returned by `Fl::event_key()`, but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by `Fl::event_state()`. If the bit is on that shift key must be pushed.
Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

### 32.147.3.8 soft()

```cpp
char Fl_Value_Input::soft ( ) const [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value. The default is true.

### 32.147.3.9 textcolor()

```cpp
Fl_Color Fl_Value_Input::textcolor ( ) const [inline]
```

Gets the color of the text in the value box.
32.147.3.10  textfont() [1/2]

\texttt{Fl\_Font Fl\_Value\_Input::textfont ( ) const [inline]}

Gets the typeface of the text in the value box.

32.147.3.11  textfont() [2/2]

\texttt{void Fl\_Value\_Input::textfont ( Fl\_Font s ) [inline]}

Sets the typeface of the text in the value box.

32.147.3.12  textsize() [1/2]

\texttt{Fl\_Fontsize Fl\_Value\_Input::textsize ( ) const [inline]}

Gets the size of the text in the value box.

32.147.3.13  textsize() [2/2]

\texttt{void Fl\_Value\_Input::textsize ( Fl\_Fontsize s ) [inline]}

Sets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl\_Value\_Input.H
- Fl\_Value\_Input.cxx

32.148  \texttt{Fl\_Value\_Output Class Reference}

The \texttt{Fl\_Value\_Output} widget displays a floating point value.

\#include \langle Fl\_Value\_Output.H\rangle

Inheritance diagram for \texttt{Fl\_Value\_Output}:

```
<table>
<thead>
<tr>
<th>Fl_Value_Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Valuator</td>
</tr>
<tr>
<td>Fl_Widget</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- \texttt{Fl\_Value\_Output (int x, int y, int w, int h, const char */=0)}
  
  Creates a new \texttt{Fl\_Value\_Output} widget using the given position, size, and label string.

- \texttt{int handle (int)}

  \textit{Handles the specified event.}

- \texttt{uchar soft () const}

  If "soft" is turned on, the user is allowed to drag the value outside the range.

- \texttt{void soft (uchar s)}
If "soft" is turned on, the user is allowed to drag the value outside the range.

- `Fl_Color textcolor () const`
  Sets the color of the text in the value box.
- `void textcolor (Fl_Color s)`
  Gets the color of the text in the value box.
- `Fl_Font textfont () const`
  Gets the typeface of the text in the value box.
- `void textfont (Fl_Font s)`
  Sets the typeface of the text in the value box.
- `Fl_Fontsize textsize () const`
  Gets the size of the text in the value box.
- `void textsize (Fl_Fontsize s)`

Protected Member Functions

- `void draw ()`
  Draws the widget.

Additional Inherited Members

32.148.1 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.

![Figure 32.59 Fl_Value_Output](image)

32.148.2 Constructor & Destructor Documentation

32.148.2.1 `Fl_Value_Output()`

`Fl_Value_Output::Fl_Value_Output (int X, int Y, int W, int H, const char * l = 0)`

Creates a new `Fl_Value_Output` widget using the given position, size, and label string. The default boxtype is `FL_NO_BOX`. Inherited destructor destroys the Valuator.

32.148.3 Member Function Documentation
32.148.3.1 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`.

32.148.3.2 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**

| 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`.

32.148.3.3 soft() [1/2]

```cpp
uchar Fl_Value_Output::soft ( ) const [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

32.148.3.4 soft() [2/2]

```cpp
void Fl_Value_Output::soft ( uchar s ) [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

32.148.3.5 textcolor() [1/2]

```cpp
Fl_Color Fl_Value_Output::textcolor ( ) const [inline]
```

Sets the color of the text in the value box.

---

*Generated by Doxygen*
32.148.3.6  textcolor() [2/2]

```c
void Fl_Value_Output::textcolor ( Fl_Color s ) [inline]
```

Gets the color of the text in the value box.

32.148.3.7  textfont() [1/2]

```c
Fl_Font Fl_Value_Output::textfont ( ) const [inline]
```

Gets the typeface of the text in the value box.

32.148.3.8  textfont() [2/2]

```c
void Fl_Value_Output::textfont ( Fl_Font s ) [inline]
```

Sets the typeface of the text in the value box.

32.148.3.9  textsize()

```c
Fl_Fontsize Fl_Value_Output::textsize ( ) const [inline]
```

Gets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Output.H
- Fl_Value_Output.cxx

32.149  Fl_Value_Slider Class Reference

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

```c
#include <Fl_Value_Slider.H>
```

Inheritance diagram for Fl_Value_Slider:
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

32.149.1 Detailed Description

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

![Figure 32.60 Fl_Value_Slider](image)

32.149.2 Constructor & Destructor Documentation

32.149.2.1 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.
32.149.3 Member Function Documentation

32.149.3.1 draw()

void Fl_Value_Slider::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Slider.

32.149.3.2 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.

32.149.3.3 textcolor() [1/2]

Fl_Color Fl_Value_Slider::textcolor ( ) const [inline]
Gets the color of the text in the value box.

32.149.3.4 textcolor() [2/2]

void Fl_Value_Slider::textcolor (Fl_Color s) [inline]
Sets the color of the text in the value box.
32.149.3.5  textfont() [1/2]

    Fl_Font Fl_Value_Slider::textfont ( ) const [inline]
Gets the typeface of the text in the value box.

32.149.3.6  textfont() [2/2]

    void Fl_Value_Slider::textfont ( Fl_Font s ) [inline]
Sets the typeface of the text in the value box.

32.149.3.7  textsize() [1/2]

    Fl_Fontsize Fl_Value_Slider::textsize ( ) const [inline]
Gets the size of the text in the value box.

32.149.3.8  textsize() [2/2]

    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

32.150  Fl_Widget Class Reference

Fl_Widget is the base class for all widgets in FLTK.
#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)
  Sets the current callback function for the widget.

- void callback (Fl_Callback +cb, void +p)
  Sets the current callback function for the widget.

- void callback (Fl_Callback0 +cb)
  Sets the current callback function for the widget.

- void callback (Fl_Callback1 +cb, long p=0)
  Sets the current callback function for the widget.

- unsigned int changed () const
  Checks if the widget value changed since the last callback.

- void clear_active ()
  Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()
  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)
  Clears or sets the damage flags.

- void clear_output ()
  Sets a widget to accept input.

- void clear_visible ()
  Hides the widget.

- void clear_visible_focus ()
  Disables keyboard focus navigation with this widget.

- Fl_Color color () const
  Gets the background color of the widget.

- void color (Fl_Color bg)
  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)
  Sets the background and selection color of the widget.

- Fl_Color color2 () const
  For back compatibility only.

- void color2 (unsigned a)
  For back compatibility only.

- int contains (const Fl_Widget *w) const
  Checks if w is a child of this widget.

- void copy_label (const char *new_label)
  Sets the current label.

- void copy_tooltip (const char *text)
  Sets the current tooltip text.

- uchar damage () const
  Returns non-zero if draw() needs to be called.

- void damage (uchar c)
  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)
  Sets the damage bits for an area inside the widget.

- int damage_resize (int, int, int)
  Internal use only.
• void `deactivate()`
  Deactivates the widget.

• `Fl_Image * deimage()`
  Gets the image that is used as part of the widget label 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.

- int `shortcut_label () const`
  
  Returns whether the widget’s label uses ‘&’ to indicate shortcuts.

- void `shortcut_label (int value)`
  
  Sets whether the widget’s label uses ‘&’ to indicate shortcuts.

- virtual void `show ()`
  
  Makes a widget visible.

- void `size (int W, int H)`
  
  Changes the size of the widget.

- int `take_focus ()`
  
  Gives the widget the keyboard focus.

- unsigned int `takesevents () const`
  
  Returns if the widget is able to take events.
• int test_shortcut ()
  Returns true if the widget's label contains the entered '&x' shortcut.

• const char * tooltip () const
  Gets the current tooltip text.

• void tooltip (const char *text)
  Sets the current tooltip text.

• Fl_Window * top_window () const
  Returns a pointer to the top-level window for the widget.

• Fl_Window * top_window_offset (int &xoff, int &yoff) const
  Finds the x/y offset of the current widget relative to the top-level window.

• uchar type () const
  Gets the widget type.

• void type (uchar t)
  Sets the widget type.

• int use_accents_menu ()
  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

• void * user_data () const
  Gets the user data for this widget.

• void user_data (void *v)
  Sets the user data for this widget.

• unsigned int visible () const
  Returns whether a widget is visible.

• unsigned int visible_focus () const
  Checks whether this widget has a visible focus.

• void visible_focus (int v)
  Modifies keyboard focus navigation.

• int visible_r () const
  Returns whether a widget and all its parents are visible.

• int w () const
  Gets the widget width.

• Fl_When when () const
  Returns the conditions under which the callback is called.

• void when (uchar i)
  Sets the flags used to decide when a callback is called.

• Fl_Window * window () const
  Returns a pointer to the nearest parent window up the widget hierarchy.

• int x () const
  Gets the widget position in its window.

• int y () const
  Gets the widget position in its window.

• virtual ~Fl_Widget ()
  Destroys the widget.

**Static Public Member Functions**

• static void default_callback (Fl_Widget *widget, void *data)
  The default callback for all widgets that don't set a callback.

• static unsigned int label_shortcut (const char *t)
  Returns the Unicode value of the '&x' shortcut in a given text.

• static int test_shortcut (const char *t, const bool require_alt=false)
  Returns true if the given text contains the entered '&x' shortcut.
Protected Types

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU = 1<<19,
  NEEDS_KEYBOARD = 1<<20, USERFLAG3 = 1<<29, USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}

flags possible values enumeration.

Protected Member Functions

• void clear_flag (unsigned int c)
  Clears a flag in the flags mask.
  
• void draw_backdrop () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
  
• void draw_box () const
  Draws the widget box according its box style.
  
• void draw_box (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.
  
• void draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.
  
• void draw_focus () const
  Draws a focus rectangle around the widget.
  
• void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.
  
• void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.
  
• void draw_label () const
  Draws the widget's label at the defined label position.
  
• void draw_label (int, int, int, int) const
  Draws the label in an arbitrary bounding box.
  
• Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.
  
• unsigned int flags () const
  Gets the widget flags mask.
  
• void h (int v)
  Internal use only.
  
• void set_flag (unsigned int c)
  Sets a flag in the flags mask.
  
• void w (int v)
  Internal use only.
  
• void x (int v)
  Internal use only.
  
• void y (int v)
  Internal use only.
### Friends
- void Fl::focus (Fl_Widget *)
- class Fl_Group

#### 32.150.1 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.

#### 32.150.2 Member Enumeration Documentation

##### 32.150.2.1 anonymous enum

An anonymous enumeration for flags possible values enumeration. See `activate()`, `output()`, `visible()`, `changed()`, `set_visible_focus()`

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INACTIVE</td>
<td>the widget can't receive focus, and is disabled but potentially visible</td>
</tr>
<tr>
<td>INVISIBLE</td>
<td>the widget is not drawn, but can receive a few special events</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>for output only</td>
</tr>
<tr>
<td>NOBORDER</td>
<td>don't draw a decoration (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>FORCE_POSITION</td>
<td>don't let the window manager position the window (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>NON_MODAL</td>
<td>this is a hovering toolbar window (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>SHORTCUT_LABEL</td>
<td>the label contains a shortcut we need to draw</td>
</tr>
<tr>
<td>CHANGED</td>
<td>the widget value changed</td>
</tr>
<tr>
<td>OVERRIDE</td>
<td>position window on top (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>VISIBLE_FOCUS</td>
<td>accepts keyboard focus navigation if the widget can have the focus</td>
</tr>
<tr>
<td>COPIED_LABEL</td>
<td>the widget label is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>CLIP_CHILDREN</td>
<td>all drawing within this widget will be clipped (<code>Fl_Group</code>)</td>
</tr>
<tr>
<td>MENU_WINDOW</td>
<td>a temporary popup window, dismissed by clicking outside (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>TOOLTIP_WINDOW</td>
<td>a temporary popup, transparent to events, and dismissed easily (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>MODAL</td>
<td>a window blocking input to all other winows (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>NO_OVERLAY</td>
<td>window not using a hardware overlay plane (<code>Fl_Menu_Window</code>)</td>
</tr>
<tr>
<td>GROUP_RELATIVE</td>
<td>Reserved, not implemented. DO NOT USE.</td>
</tr>
<tr>
<td>COPIED_TOOLTIP</td>
<td>the widget tooltip is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>FULLSCREEN</td>
<td>a fullscreen window (<code>Fl_Window</code>)</td>
</tr>
<tr>
<td>MAC_USE_ACCENTS_MENU</td>
<td>On the Mac OS platform, pressing and holding a key on the keyboard opens an accented-character menu window (<code>Fl_Input_</code>, <code>Fl_Text_Editor</code>)</td>
</tr>
<tr>
<td>NEEDS_KEYBOARD</td>
<td>set this on touch screen devices if a widget needs a keyboard when it gets Focus. See also <code>Fl_Screen_Driver::request_keyboard()</code></td>
</tr>
<tr>
<td>USERFLAG3</td>
<td>reserved for 3rd party extensions</td>
</tr>
<tr>
<td>USERFLAG2</td>
<td>reserved for 3rd party extensions</td>
</tr>
</tbody>
</table>

Generated by Doxygen
32.150.3 Constructor & Destructor Documentation

32.150.3.1 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>

32.150.3.2 ~Fl_Widget()

Fl_Widget::~Fl_Widget ( )  [virtual]

Destroys the widget.
Destroys the widget, taking care of throwing focus before if any.
Destroying single widgets is not very common. You almost always want to destroy the parent group instead, which will destroy all of the child widgets and groups in that group.

Since

FLTK 1.3, the widget's destructor removes the widget from its parent group, if it is member of a group.

Destruction removes the widget from any parent group! And groups when destroyed destroy all their children. This is convenient and fast.

32.150.4 Member Function Documentation

32.150.4.1 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()
32.150.4.2 active()

```cpp
unsigned int Fl_Widget::active ( ) const [inline]
```

Returns whether the widget is active.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the widget is inactive</td>
</tr>
</tbody>
</table>

See also

active_r(), activate(), deactivate()

32.150.4.3 active_r()

```cpp
int Fl_Widget::active_r ( ) const
```

Returns whether the widget and all of its parents are active.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if this or any of the parent widgets are inactive</td>
</tr>
</tbody>
</table>

See also

active(), activate(), deactivate()

32.150.4.4 align() [1/2]

```cpp
Fl_Align Fl_Widget::align ( ) const [inline]
```

Gets the label alignment.

Returns

- label alignment

See also

label(), align(Fl_Align), Fl_Align

32.150.4.5 align() [2/2]

```cpp
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

- **alignment** new label alignment
32.150.4.6 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 `flintptr_t` or similar to avoid 64-bit platform incompatibilities.

32.150.4.7 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()`

32.150.4.8 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.

32.150.4.9 as_group()

```cpp
virtual Fl_Group* Fl_Widget::as_group ( ) [inline], [virtual]
```

Generated by Doxygen
Returns an **Fl_Group** pointer if this widget is an **Fl_Group**.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from **Fl_Group**. If it returns non-NULL, then the widget in question is derived from **Fl_Group**, and you can use the returned pointer to access its children or other **Fl_Group**-specific methods.

Example:
```c
void my_callback (Fl_Widget *w, void *) {
    Fl_Group *g = w->as_group();
    if (g)
        printf ("This group has %d children\n", g->children());
    else
        printf ("This widget is not a group!\n");
}
```

**Return values**

| NULL | if this widget is not derived from **Fl_Group**. |

**Note**

This method is provided to avoid dynamic_cast.

**See also**

**Fl_Widget::as_window()**, **Fl_Widget::as_gl_window()**

Reimplemented in **Fl_Group**.

### 32.150.4.10 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**.

### 32.150.4.11 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**
### 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

### 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

### callback() [2/5]

**void Fl_Widget::callback (Fl_Callback * cb) [inline]**

Sets the current callback function for the widget. Each widget has a single callback.

**Parameters**

- **in cb** new callback

### callback() [3/5]

**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
32.150.4.16 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

- **in cb** new callback

32.150.4.17 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

- **in cb** new callback
- **in p** user data

32.150.4.18 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)

32.150.4.19 clear_active()

```cpp
void Fl_Widget::clear_active ( ) [inline]
```

Marks the widget as inactive without sending events or changing focus. This is mainly for specialized use, for normal cases you want deactivate().
See also
   deactivate()

32.150.4.20  clear_changed()

void Fl_Widget::clear_changed ( ) [inline]
Marks the value of the widget as unchanged.
See also
   changed(), set_changed()

32.150.4.21  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()

32.150.4.22  clear_output()

void Fl_Widget::clear_output ( ) [inline]
Sets a widget to accept input.
See also
   set_output(), output()

32.150.4.23  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.

32.150.4.24  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)

32.150.4.25  color() [1/3]

Fl_Color Fl_Widget::color ( ) const [inline]
 Gets the background color of the widget.

Returns

current background color

See also

color(Fl_Color), color(Fl_Color, Fl_Color)

32.150.4.26  color() [2/3]

void Fl_Widget::color ( Fl_Color bg ) [inline]
 Sets the background color of the widget.
The color is passed to the box routine. The color is either an index into an internal table of RGB colors or an RGB
color value generated using fl_rgb_color().
The default for most widgets is FL_BACKGROUND_COLOR. Use Fl::set_color() to redefine colors in the color map.

Parameters

- **in** bg background color

See also

color(), color(Fl_Color, Fl_Color), selection_color(Fl_Color)

32.150.4.27  color() [3/3]

void Fl_Widget::color ( Fl_Color bg, Fl_Color sel ) [inline]
 Sets the background and selection color of the widget.
The two color form sets both the background and selection colors.

Parameters

- **in** bg background color
- **in** sel selection color

See also

color(unsigned), selection_color(unsigned)

32.150.4.28  color2() [1/2]

Fl_Color Fl_Widget::color2 ( ) const [inline]


For back compatibility only.

**Deprecated** Use `selection_color()` instead.

### 32.150.4.29 color2() [2/2]

```cpp
void Fl_Widget::color2 (unsigned a) [inline]
```

For back compatibility only.

**Deprecated** Use `selection_color(unsigned)` instead.

### 32.150.4.30 contains()

```cpp
int Fl_Widget::contains (const Fl_Widget * w) const
```

Checks if `w` is a child of this widget.

**Parameters**

| in | w | potential child widget |

**Returns**

Returns 1 if `w` is a child of this widget, or is equal to this widget. Returns 0 if `w` is NULL.

### 32.150.4.31 copy_label()

```cpp
void Fl_Widget::copy_label (const char * new_label)
```

Sets the current label.

Unlike `label()`, this method allocates a copy of the label string instead of using the original string pointer. The internal copy will automatically be freed whenever you assign a new label or when the widget is destroyed.

**Parameters**

| in | new_label | the new label text |

**See also**

`label()`

### 32.150.4.32 copy_tooltip()

```cpp
void Fl_Widget::copy_tooltip (const char * text)
```

Sets the current tooltip text.

Unlike `tooltip()`, this method allocates a copy of the tooltip string instead of using the original string pointer. The internal copy will automatically be freed whenever you assign a new tooltip or when the widget is destroyed. If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group’s tooltip instead. To avoid this behavior, you can set the child’s tooltip to an empty string (“”).
Parameters

| in  | text | New tooltip text (an internal copy is made and managed) |

See also

tooltip(const char*), tooltip()

32.150.4.33  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)

32.150.4.34  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)

32.150.4.35  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

| in  | c | bitmask of flags to set |
| in  | x,y,w,h | size of damaged area |
32.150.4.36 deactivate()

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()

32.150.4.37 default_callback()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>the Fl_Widget given to the callback</td>
</tr>
<tr>
<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)

32.150.4.38 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

32.150.4.39 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

### 32.150.4.40  deimage() [3/4]

```cpp
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)

### 32.150.4.41  deimage() [4/4]

```cpp
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.

### 32.150.4.42  do_callback() [1/3]

```cpp
void Fl_Widget::do_callback ( ) [inline]
```

Calls the widget callback function with default arguments.

This is the same as calling

dcallback(this, user_data());

**See also**

callback()

do_callback(Fl_Widget *widget, void *data)

### 32.150.4.43  do_callback() [2/3]

```cpp
void Fl_Widget::do_callback ( Fl_Widget * widget,
long arg ) [inline]
```

Calls the widget callback function with arbitrary arguments.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>call the callback with widget as the first argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>call the callback with arg as the user data (second) argument</td>
</tr>
</tbody>
</table>

See also

- callback()
- do_callback(Fl_Widget *widget, void *data)

32.150.4.44 do_callback() [3/3]

```cpp
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>in</th>
<th>widget</th>
<th>call the callback with widget as the first argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>use arg as the user data (second) argument</td>
</tr>
</tbody>
</table>

See also

- default_callback()
- callback()
- class Fl_Widget_Tracker

32.150.4.45 draw()

```cpp
virtual void Fl_Widget::draw ( ) [pure virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```


Generated by Doxygen
void Fl_Widget::draw_focus() const [inline], [protected]
Draws a focus rectangle around the widget.
This method uses the widget's boxtype and coordinates and its background color color().

See also
Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int, Fl_Color) const

void Fl_Widget::draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const [inline], [protected]
Draws a focus rectangle around the widget.
This method uses the given boxtype and coordinates and the widget's background color color().

See also
Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int, Fl_Color) const

void Fl_Widget::draw_focus(Fl_Boxtype bt, int X, int Y, int W, int H, Fl_Color bg) const [protected]
Draws a focus box for the widget at the given position and size.
This method does nothing if

- the global option Fl::visible_focus() or
- the per-widget option visible_focus() is false (off).

This means that Fl_Widget::draw_focus() or one of the more specialized methods can be called without checking these visible focus options.

Note
This method must only be called if the widget has the focus. This is not tested internally.

The boxtype bt is used to calculate the inset so the focus box is drawn inside the box borders.
The default focus box drawing color is black. The background color bg is used to determine a better visible color if necessary by using fl_contrast() with the given background color.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in bt</td>
<td>Boxtype that needs to be considered (frame width)</td>
</tr>
<tr>
<td>in X,Y,W,H</td>
<td>Bounding box</td>
</tr>
<tr>
<td>in bg</td>
<td>Background color</td>
</tr>
</tbody>
</table>
See also

Fl_Widget::draw_focus()
Fl_Widget::draw_focus(Fl_Boxtype, int, int, int) const

32.150.4.49  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.

32.150.4.50  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

32.150.4.51  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.

32.150.4.52  h() [1/2]

int Fl_Widget::h ( ) const [inline]
Gets the widget height.
Returns
the height of the widget in pixels.

32.150.4.53  h() [2/2]

void Fl_Widget::h (  
   int v ) [inline], [protected]
Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

32.150.4.54  handle()

int Fl_Widget::handle (  
   int event ) [virtual]
Handles the specified event.
You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.
Parameters

| in  | event  | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

FI_Event


32.150.4.55 hide()

```cpp
void Fl_Widget::hide ( ) [virtual]
```

Makes a widget invisible.

See also

show(), visible(), visible_r()

Reimplemented in Fl_Window, Fl_Overlay_Window, Fl_Gl_Window, Fl_Double_Window, and Fl_Browser.

32.150.4.56 image() [1/4]

```cpp
Fl_Image* Fl_Widget::image ( ) [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

32.150.4.57 image() [2/4]

```cpp
const Fl_Image* Fl_Widget::image ( ) const [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

32.150.4.58 image() [3/4]

```cpp
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)

32.150.4.59 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

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.

32.150.4.60 inside()

int Fl_Widget::inside (const Fl_Widget * wgt) const [inline]
Checks if this widget is a child of wgt.
Returns 1 if this widget is a child of wgt, or is equal to wgt. Returns 0 if wgt is NULL.

Parameters

in wgt the possible parent widget.

See also

contains()

32.150.4.61 is_label_copied()

int Fl_Widget::is_label_copied ( ) const [inline]
Returns whether the current label was assigned with copy_label().
This can be useful for temporarily overwriting the widget's label and restoring it later.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>current label was assigned with label().</td>
</tr>
<tr>
<td>1</td>
<td>current label was assigned with copy_label().</td>
</tr>
</tbody>
</table>
### 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*)`

### label() [2/3]

```cpp
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()`

### label() [3/3]

```cpp
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)`

### label_shortcut()

```cpp
unsigned int Fl_Widget::label_shortcut(
    const char * t) [static]
```

Returns the Unicode value of the 'x' shortcut in a given text.

The given text `t` (usually a widget's label or a menu text) is searched for a 'x' shortcut label, and if found, the Unicode value (code point) of the 'x' shortcut is returned.

**Parameters**

- **t text or label to search for 'x' shortcut.**
Returns

Unicode (UCS-4) value of shortcut in t or 0.

Note

Internal use only.

32.150.4.66 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

32.150.4.67 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

\begin{tabular}{|c|}
\hline
\texttt{in c} & the new label color \\
\hline
\end{tabular}

32.150.4.68 labelfont() [1/2]

\texttt{Fl\_Font Fl\_Widget::labelfont ( ) const [inline]}

Gets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft®
Windows®). The function Fl::set_font() can define new typefaces.

Returns

current font used by the label

See also

Fl\_Font

32.150.4.69 labelfont() [2/2]

\texttt{void Fl\_Widget::labelfont ( Fl\_Font f ) [inline]}

Sets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft®
Windows®). The function Fl::set_font() can define new typefaces.

Parameters

\begin{tabular}{|c|}
\hline
\texttt{in f} & the new font for the label \\
\hline
\end{tabular}
**See also**

Fl_Font

### 32.150.4.70 labelsize() [1/2]

```cpp
Fl_Fontsize Fl_Widget::labelsize ( ) const [inline]
```

Gets the font size in pixels.
The default size is 14 pixels.

Returns

the current font size

### 32.150.4.71 labelsize() [2/2]

```cpp
void Fl_Widget::labelsize ( Fl_Fontsize pix ) [inline]
```

Sets the font size in pixels.

Parameters

- **in** `pix` the new font size

See also

Fl_Fontsize labelsize()

### 32.150.4.72 labeltype() [1/2]

```cpp
Fl_Labeltype Fl_Widget::labeltype ( ) const [inline]
```

Gets the label type.

Returns

the current label type.

See also

Fl_Labeltype

### 32.150.4.73 labeltype() [2/2]

```cpp
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
32.150.4.74 measure_label()

```cpp
void Fl_Widget::measure_label {
  int & ww,
  int & hh } const [inline]
```

Sets width ww and height hh accordingly with the label size. Labels with images will return w() and h() of the image. This calls fl_measure() internally. For more information about the arguments ww and hh and word wrapping

See also

```cpp
fl_measure(const char*, int&, int&, int)
```

32.150.4.75 output()

```cpp
unsigned int Fl_Widget::output ( ) const [inline]
```

Returns if a widget is used for output only. output() means the same as `active()` except it does not change how the widget is drawn. The widget will not receive any events. This is useful for making scrollbars or buttons that work as displays rather than input devices.

Return values

```
0 if the widget is used for input and output
```

See also

```cpp
set_output(), clear_output()
```

32.150.4.76 parent() [1/2]

```cpp
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

```cpp
Fl_Group::add(Fl_Widget*)
```

32.150.4.77 parent() [2/2]

```cpp
void Fl_Widget::parent {
  Fl_Group * p } [inline]
```

Internal use only - "for hacks only". It is **STRONGLY recommended** not to use this method, because it short-circuits Fl_Group's normal widget adding and removing methods, if the widget is already a child widget of another Fl_Group. Use Fl_Group::add(Fl_Widget*) and/or Fl_Group::remove(Fl_Widget*) instead.
32.150.4.78  position()

void Fl_Widget::position (  
    int X,  
    int Y ) [inline]

Repositions the window or widget.
positional(X, Y) is a shortcut for resize(X, Y, w(), h()).

Parameters

| in   | X, Y | new position relative to the parent window |

See also

    resize(int,int,int,int), size(int,int)

32.150.4.79  redraw()

void Fl_Widget::redraw ()

Schedules the drawing of the widget.
Marks the widget as needing its draw() routine called.

32.150.4.80  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.

32.150.4.81  resize()

void Fl_Widget::resize (  
    int x,  
    int y,  
    int w,  
    int h ) [virtual]

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
positional(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 in Fl_Value_Input, Fl_Tree, Fl_Overlay_Window, Fl_Input_, Fl_Help_View, Fl_Group, Fl_Gl_Window, Fl_Double_Window, Fl_Window, Fl_Tile, Fl_Text_Display, Fl_Table, Fl_Spinner, Fl_Scroll, Fl_Input_Choice, Fl_Flex, and Fl_Browser_.

Generated by Doxygen
32.150.4.82 selection_color() [1/2]

Fl_Color Fl_Widget::selection_color ( ) const [inline]

Gets the selection color.

Returns

the current selection color

See also

selection_color(Fl_Color), color(Fl_Color, Fl_Color)

32.150.4.83 selection_color() [2/2]

void Fl_Widget::selection_color ( Fl_Color a ) [inline]

Sets the selection color.

The selection color is defined for Forms compatibility and is usually used to color the widget when it is se-
lected, although some widgets use this color for other purposes. You can set both colors at once with
color(Fl_Color bg, Fl_Color sel).

Parameters

in a the new selection color

See also

selection_color(), color(Fl_Color, Fl_Color)

32.150.4.84 set_active()

void Fl_Widget::set_active ( ) [inline]

Marks the widget as active without sending events or changing focus.
This is mainly for specialized use, for normal cases you want activate().

See also

activate()

32.150.4.85 set_changed()

void Fl_Widget::set_changed ( ) [inline]

Marks the value of the widget as changed.

See also

changed(), clear_changed()

32.150.4.86 set_output()

void Fl_Widget::set_output ( ) [inline]

Sets a widget to output only.

See also

output(), clear_output()
32.150.4.87  set_visible()

void Fl_Widget::set_visible ( ) [inline]
Makes the widget visible.
You must still redraw the parent widget to see a change in the window. Normally you want to use the show() method instead.

32.150.4.88  set_visible_focus()

void Fl_Widget::set_visible_focus ( ) [inline]
Enables keyboard focus navigation with this widget.
Note, however, that this will not necessarily mean that the widget will accept focus, but for widgets that can accept focus, this method enables it if it has been disabled.

See also
visible_focus(), clear_visible_focus(), visible_focus(int)

32.150.4.89  shortcut_label() [1/2]

int Fl_Widget::shortcut_label ( ) const [inline]
Returns whether the widget's label uses '&' to indicate shortcuts.

See also
void shortcut_label(int value)

32.150.4.90  shortcut_label() [2/2]

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.

32.150.4.91  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()
### 32.150.4.92 size()

```cpp
void Fl_Widget::size (  
  int W,  
  int H ) [inline]
```

Changes the size of the widget.

size(W, H) is a shortcut for resize(x(), y(), W, H).

**Parameters**

| in | W/H | new size |

**See also**

position(int,int), resize(int,int,int)

---

### 32.150.4.93 take_focus()

```cpp
int Fl_Widget::take_focus ( )
```

Gives the widget the keyboard focus.

Tries to make this widget be the Fl::focus() widget, by first sending it an FL_FOCUS event, and if it returns non-zero, setting Fl::focus() to this widget. You should use this method to assign the focus to a widget.

**Returns**

true if the widget accepted the focus.

---

### 32.150.4.94 takesevents()

```cpp
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 |

---

### 32.150.4.95 test_shortcut() [1/2]

```cpp
int Fl_Widget::test_shortcut ( )
```

Returns true if the widget's label contains the entered '&x' shortcut.

This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The widget's label is searched for a '&x' shortcut, and if found, this is compared with the entered key value.

Fl::event_text() is used to get the entered key value.

**Returns**

true, if the entered text matches the widget's '&x' shortcut, false (0) otherwise.

**Note**

Internal use only.
32.150.4.96  test_shortcut() [2/2]

int Fl_Widget::test_shortcut (const char * t, const bool require_alt = false) [static]

Returns true if the given text t contains the entered '&x' shortcut.
This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The given text t (usually a widget's label or menu text) is searched for a '&x' shortcut, and if found, this is compared with the entered key value. Fl::event_text() is used to get the entered key value. Fl::event_state() is used to get the Alt modifier, if require←_alt is true.

Parameters

<table>
<thead>
<tr>
<th>t</th>
<th>text or label to search for '&amp;x' shortcut.</th>
</tr>
</thead>
<tbody>
<tr>
<td>require_alt</td>
<td>if true: match only if Alt key is pressed.</td>
</tr>
</tbody>
</table>

Returns

true, if the entered text matches the '&x' shortcut in t, false (0) otherwise.

Note
Internal use only.

32.150.4.97  tooltip() [1/2]

const char* Fl_Widget::tooltip () const [inline]

Gets the current tooltip text.

Returns

a pointer to the tooltip text or NULL

See also
tooltip(const char*), copy_tooltip(const char*)

32.150.4.98  tooltip() [2/2]

void Fl_Widget::tooltip (const char * text )

Sets the current tooltip text.

Sets a string of text to display in a popup tooltip window when the user hovers the mouse over the widget. The string is not copied, so make sure any formatted string is stored in a static, global, or allocated buffer. If you want a copy made and managed for you, use the copy_tooltip() method, which will manage the tooltip string automatically. If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group's tooltip instead. To avoid this behavior, you can set the child's tooltip to an empty string ("").

Parameters

| in text | New tooltip text (no copy is made) |

Generated by Doxygen
See also

copy_tooltip(const char*), tooltip()

32.150.4.99  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()

32.150.4.100  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)

32.150.4.101  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"?

32.150.4.102  type()  [2/2]

void Fl_Widget::type (  
    uchar t ) [inline]

Sets the widget type.
This is used for Forms compatibility.

32.150.4.103  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

32.150.4.104  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 |

32.150.4.105  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()

32.150.4.106  visible_focus() [1/2]
unsigned int Fl_Widget::visible_focus ( ) const [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()

32.150.4.107  visible_focus() [2/2]
void Fl_Widget::visible_focus ( 
    int v ) [inline]
Modifies keyboard focus navigation.

Parameters

| in  | v | set or clear visible focus |
See also

- set_visible_focus(), clear_visible_focus(), visible_focus()

### 32.150.4.108 visible_r()

```cpp
int Fl_Widget::visible_r ( ) const
```

Returns whether a widget and all its parents are visible.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The widget or any of its parents are invisible.</td>
</tr>
</tbody>
</table>

See also

- show(), hide(), visible()

### 32.150.4.109 w() [1/2]

```cpp
int Fl_Widget::w ( ) const [inline]
```

Gets the widget width.

**Returns**

- The width of the widget in pixels.

### 32.150.4.110 w() [2/2]

```cpp
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.

### 32.150.4.111 when() [1/2]

```cpp
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)

### 32.150.4.112 when() [2/2]

```cpp
void Fl_Widget::when ( uchar i ) [inline]
```

Sets the flags used to decide when a callback is called.

This controls when callbacks are done. The following values are useful, the default value is FL_WHEN_RELEASE:

- 0: The callback is not done, but changed() is turned on.
- FL_WHEN_CHANGED: The callback is done each time the text is changed by the user.

- FL_WHEN_RELEASE: The callback will be done when this widget loses the focus, including when the window is unmapped. This is a useful value for text fields in a panel where doing the callback on every change is wasteful. However the callback will also happen if the mouse is moved out of the window, which means it should not do anything visible (like pop up an error message). You might do better setting this to zero, and scanning all the items for changed() when the OK button on a panel is pressed.

- FL_WHEN_ENTER_KEY: If the user types the Enter key, the entire text is selected, and the callback is done if the text has changed. Normally the Enter key will navigate to the next field (or insert a newline for a Fl_Multiline_Input) - this changes the behavior.

- FL_WHEN_ENTER_KEY|FL_WHEN_NOT_CHANGED: The Enter key will do the callback even if the text has not changed. Useful for command fields. Fl_Widget::when() is a set of bitflags used by subclasses of Fl_Widget to decide when to do the callback.

If the value is zero then the callback is never done. Other values are described in the individual widgets. This field is in the base class so that you can scan a panel and do_callback() on all the ones that don't do their own callbacks in response to an "OK" button.

**Parameters**

```
in  i  set of flags
```

### 32.150.4.113 window()

```cpp
Fl_Widget * 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()

### 32.150.4.114 x() [1/2]

```cpp
int Fl_Widget::x ( ) const [inline]
```

Gets the widget position in its window.

**Returns**

the x position relative to the window

### 32.150.4.115 x() [2/2]

```cpp
void Fl_Widget::x ( int v ) [inline], [protected]
```
Internal use only. Use `position(int,int), size(int,int)` or `resize(int,int,int,int)` instead.

### 32.150.4.116 y() [1/2]

```cpp
int Fl_Widget::y ( ) const [inline]
```

Gets the widget position in its window.

**Returns**

the y position relative to the window

### 32.150.4.117 y() [2/2]

```cpp
void Fl_Widget::y ( int v ) [inline], [protected]
```

Internal use only. Use `position(int,int), size(int,int)` or `resize(int,int,int,int)` instead.

The documentation for this class was generated from the following files:

- Fl_Widget.H
- Fl.cxx
- fl_boxtype.cxx
- fl_labeltype.cxx
- fl_shortcut.cxx
- Fl_Tooltip.cxx
- Fl_Widget.cxx
- Fl_Window.cxx

### 32.151 FL_Widget_Surface Class Reference

A surface on which any FLTK widget can be drawn.

```
#include <Fl_Widget_Surface.H>
```

Inheritance diagram for Fl_Widget_Surface:

```
Fl_Surface_Device
         |             |
         |             |
Fl_Widget_Surface
```

#### 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)**
  
  Computes the coordinates of the current origin of graphics functions.

- **virtual void origin (int x, int y)**
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- **void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)**
  
  Draws a rectangular part of an on-screen window.
• virtual int printable_rect (int *w, int *h)
  Computes the width and height of the drawable area of the drawing surface.
• virtual void translate (int x, int y)
  Translates the current graphics origin accounting for the current rotation.
• virtual void untranslate ()
  Undoes the effect of a previous translate() call.

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

32.151.1 Detailed Description

A surface on which any FLTK widget can be drawn.

32.151.2 Constructor & Destructor Documentation

32.151.2.1 Fl_Widget_Surface()

Fl_Widget_Surface::Fl_Widget_Surface ( Fl_Graphics_Driver * d ) [protected]
The constructor.
Parameters

\[ d \] can be nul.

32.151.3 Member Function Documentation

32.151.3.1 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

\[ \text{in} \quad \text{widget} \quad \text{Any FLTK widget (e.g., standard, custom, window).} \]
32.151.3.2 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.

32.151.3.3 origin() [1/2]

```cpp
void Fl_Widget_Surface::origin (    
    int * x,                                                   
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

| out  | x,y | If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin. |


32.151.3.4 origin() [2/2]

```cpp
void Fl_Widget_Surface::origin (    
    int x,                                                   
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. `Origin()` calls are not affected by `rotate()` calls (for classes derived from `Fl_Paged_Device`).

Parameters

| in   | x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |


32.151.3.5 print_window_part()

```cpp
void Fl_Widget_Surface::print_window_part (    
    Fl_Window * win,                                            
    int x,                                                   
    int y,                                                   
```

Generated by Doxygen
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>win</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>x</td>
<td>The rectangle left</td>
</tr>
<tr>
<td>y</td>
<td>The rectangle top</td>
</tr>
<tr>
<td>w</td>
<td>The rectangle width</td>
</tr>
<tr>
<td>h</td>
<td>The rectangle height</td>
</tr>
<tr>
<td>delta_x, delta_y</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>

32.151.3.6 printable_rect()

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


32.151.3.7 translate()

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

32.152 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.

#include <Fl.H>
Public Member Functions

- int deleted ()
  Returns 1, if the watched widget has been deleted.
- int exists ()
  Returns 1, if the watched widget exists (has not been deleted).
- Fl_Widget_Tracker (Fl_Widget ∗wi)
  The constructor adds a widget to the watch list.
- Fl_Widget ∗ widget ()
  Returns a pointer to the watched widget.
- ∼Fl_Widget_Tracker ()
  The destructor removes a widget from the watch list.

32.152.1 Detailed Description

This class should be used to control safe widget deletion. You can use an Fl_Widget_Tracker object to watch another widget, if you need to know 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();
  }
}
```

32.152.2 Member Function Documentation

32.152.2.1 deleted()

int Fl_Widget_Tracker::deleted () [inline]
Returns 1, if the watched widget has been deleted.
This is a convenience method. You can also use something like
if (wp.widget() == 0) // ...
where wp is an Fl_Widget_Tracker object.

32.152.2.2 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.
32.152.2.3  \texttt{widget()}

\texttt{Fl_Widget* Fl_Widget_Tracker::widget () [inline]}

Returns a pointer to the watched widget.
This pointer is \texttt{NULL}, if the widget has been deleted.
The documentation for this class was generated from the following files:

- \texttt{Fl.H}
- \texttt{Fl.cxx}

32.153  \texttt{Fl_Window} Class Reference

This widget produces an actual window.

\#include <Fl_Window.H>

Inheritance diagram for \texttt{Fl_Window}:

- \texttt{Fl_Widget}
- \texttt{Fl_Group}
- \texttt{Fl_Window}
  \hspace{1cm}
  - \texttt{Fl_Double_Window}
  - \texttt{Fl_Gl_Window}
  - \texttt{Fl_Single_Window}
  \hspace{1cm}
  - \texttt{Fl_Cairo_Window}
  - \texttt{Fl_Overlay_Window}
  - \texttt{Fl_Glut_Window}
  - \texttt{Fl_Menu_Window}

\textbf{Public Types}

- \texttt{typedef struct HICON__ * HICON}

\textbf{Public Member Functions}

- virtual class \texttt{Fl_Double_Window * as_double_window ()}
  \textit{Return non-null if this is an Fl_Double_Window object.}
- virtual class \texttt{Fl_Overlay_Window * as_overlay_window ()}
  \textit{Return non-null if this is an Fl_Overlay_Window object.}
- virtual \texttt{Fl_Window * as_window ()}
  \textit{Returns an Fl_Window pointer if this widget is an Fl_Window.}
- unsigned int \texttt{border () const}
  \textit{Returns whether the window possesses a border.}
- void \texttt{border (int b)}
  \textit{Sets whether or not the window manager border is around the window.}
- void \texttt{clear_border ()}
  \textit{Fast inline function to turn the window manager border off.}
- void \texttt{clear_modal_states ()}
  \textit{Cleans the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.}
- void \texttt{copy_label (const char *a)}
  \textit{Sets the window titlebar label to a copy of a character string.}
- void \texttt{cursor (const Fl_RGB_Image *, int, int)}
  \textit{Changes the cursor for this window.}
- void \texttt{cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)}
For back compatibility only.

- **void cursor(Fl_Cursor)**
  Changes the cursor for this window.

- **int decorated_h() const**
  Returns the window height including any window title bar and any frame added by the window manager.

- **int decorated_w() const**
  Returns the window width including any frame added by the window manager.

- **void default_cursor(Fl_Cursor)**
  For back compatibility only.

- **void default_cursor(Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)**
  For back compatibility only.

- **void default_cursor(Fl_Cursor c, Fl_Color=FL_WHITE)**
  Sets the default window cursor.

- **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(const Fl_Widget &p, int offscreen=0)**
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot(const Fl_Widget *, int offscreen=0)**
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- **void hotspot(int x, int y, int offscreen=0)**
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- **const void *icon() const**
  Gets the current icon window target dependent data.

- **void icon(const Fl_RGB_Image *)**
  Sets or resets a single window icon.

- **void icon(const void *ic)**
  Platform-specific method to set the window icon usable on Windows and X11 only.

- **void iconize()**
  Iconifies the window.

- **const char *iconlabel() const**
  See void Fl_Window::iconlabel(const char *)

- **void iconlabel(const char *)**
Sets the icon label.

- **void icons (const Fl_RGB_Image *[], int)**
  
  Sets the window icons.

- **void icons (HICON big_icon, HICON small_icon)**
  
  Sets the window icons using HICON handles (Windows platform only).

- **const char * label () const**

  See void Fl_Window::label(const char*)

- **void label (const char *)**

  Sets the window title bar label.

- **void label (const char *label, const char *iconlabel)**

  Sets the icon label.

- **void make_current ()**

  Sets things up so that the drawing functions in `<FL/fl_draw.H>` will go into this window.

- **unsigned int menu_window () const**

  Returns true if this window is a menu window.

- **unsigned int modal () const**

  Returns true if this window is modal.

- **unsigned int non_modal () const**

  Returns true if this window is modal or non-modal.

- **fl_uintptr_t os_id ()**

  Returns a platform-specific identification of a shown window, or 0 if not shown.

- **unsigned int override () const**

  Returns non zero if 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 screen_num (int screen_num)**

  Set the number of the screen where to map the window.

- **void set_menu_window ()**

  Marks the window as a menu window.

- **void set_modal ()**

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

- **void set_non_modal ()**

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

- **void set_override ()**

  Activates the flags NOBORDER|FL_OVERRIDE.

- **void set_tooltip_window ()**

  Marks the window as a tooltip window.

- **const Fl_Image * shape ()**

  Returns the image controlling the window shape or NULL.

- **void shape (const Fl_Image &b)**

  Set the window’s shape with an Fl_Image.

- **void shape (const Fl_Image *img)**

  Assigns a non-rectangular shape to the window.

- **virtual void show ()**

  Puts the window on the screen.

- **void show (int argc, char **argv)**

  Generated by Doxygen
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_icons (HICON big_icon, HICON small_icon)`
  Sets the default window icons (Windows platform only).

- `static const char * default_xclass ()`
  Returns the default xclass.

- `static void default_xclass (const char *)`
  Sets the default window xclass.

- `static bool is_a_rescale ()`
  Returns true when a window is being rescaled.

### Protected Member Functions

- `void default_size_range ()`
  Protected method to calculate the default size range of a window.

- `void draw ()`
  Draws the widget.

- `virtual void flush ()`
  Forces the window to be drawn, this window is also made current and calls `draw()`.

- `int force_position () const`
  Returns the internal state of the window's FORCE_POSITION flag.

- `void force_position (int force)`
  Sets an internal flag that tells FLTK and the window manager to honor position requests.

- `void free_icons ()`
Deletes all icons previously attached to the window.

- int is_resizable()

Protected method to determine whether a window is resizable.

### 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

### 32.153.1 Detailed Description

This widget produces an actual window. This can either be a main window, with a border and title and all the window management controls, or a "subwindow" inside a window. This is controlled by whether or not the window has a parent().

Once you create a window, you usually add children Fl_Widget's to it by using window->add(child) for each new widget. See Fl_Group for more information on how to add and remove children.

There are several subclasses of Fl_Window that provide double-buffering, overlay, menu, and OpenGL support. The window's callback is done if the user tries to close a window using the window manager and Fl::modal() is zero or equal to the window. Fl_Window has a default callback that calls Fl_Window::hide().

### 32.153.2 Constructor & Destructor Documentation

#### 32.153.2.1 Fl_Window() [1/2]

Fl_Window::Fl_Window (  
  int w,  
  int h,  
  const char * title = 0 )

Creates a window from the given 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)
32.153.2.2 Fl_Window() [2/2]

```c
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

```c
Fl_Window(int w, int h, const char *title)
```

32.153.2.3 ~Fl_Window()

```c
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.

32.153.3 Member Function Documentation

32.153.3.1 as_window()

```c
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**

```c
NULL if this widget is not derived from Fl_Window.
```

**Note**

This method is provided to avoid dynamic_cast.

See also

```c
Fl_Widget::as_group(), Fl_Widget::as_gl_window()
```

Reimplemented from Fl_Widget.

32.153.3.2 border()

```c
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.*

32.153.3.3 clear_border()

```c
void Fl_Window::clear_border ( ) [inline]
```
Fast inline function to turn the window manager border off. It only works before `show()` is called.

### 32.153.3.4 `clear_modal_states()`

```cpp
void Fl_Window::clear_modal_states ( ) [inline]
```

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window. Note that there are three states for a window: modal, non-modal, and normal.

You cannot 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

```cpp
void set_modal(), void set_non_modal()
```

### 32.153.3.5 `current()`

```cpp
Fl_Window * Fl_Window::current ( ) [static]
```

Returns the last window that was made current.

See also

```cpp
Fl_Window::make_current()
```

### 32.153.3.6 `cursor()` [1/3]

```cpp
void Fl_Window::cursor ( const Fl_RGB_Image * image,
                         int hotx,
                         int hoty )
```

Changes the cursor for this window. This always calls the system. If you are changing the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor is different.

The default cursor will be used if the provided image cannot be used as a cursor.

See also

```cpp
cursor(Fl_Cursor), default_cursor()
```

### 32.153.3.7 `cursor()` [2/3]

```cpp
void Fl_Window::cursor ( Fl_Cursor c,
                        Fl_Widget * w )
```

Generated by Doxygen
32.153.3.8 cursor() [3/3]

```cpp
void Fl_Window::cursor (Fl_Cursor c)
```

Changes the cursor for this window.
This always calls the system. If you are changing the cursor a lot you may want to keep track of how you set it in a
static variable and call this only if the new cursor is different.
The type Fl_Cursor is an enumeration defined in `<FL/Enumerations.H>`.
See also
cursor(const Fl_RGB_Image*, int, int), default_cursor()

32.153.3.9 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, or if window is not yet mapped.

Note
Under X11, FLTK is able to compute the size of window titlebars and borders only if these decoration ele-
ments are strictly X11-based. When that's not the case, decorated_h() returns the same value as h() and
decorated_w() as w(), and FLTK cannot access window decorations.
Under X11 again, the values returned by decorated_h() and decorated_w() may not be reliable during a
resize operation. The size of decoration elements of a window is best computed when the window is first
mapped.

32.153.3.10 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, or if window is not yet mapped.

See also
decorated_h().

32.153.3.11 default_cursor() [1/2]

```cpp
void Fl_Window::default_cursor (Fl_Cursor c,
Fl_Color ,
Fl_Color = FL_WHITE )
```

For back compatibility only.
same as Fl_Window::default_cursor(Fl_Cursor)

32.153.3.12 default_cursor() [2/2]

```cpp
void Fl_Window::default_cursor (Fl_Cursor c)
```

Sets the default window cursor.
This is the cursor that will be used after the mouse pointer leaves a widget with a custom cursor set.

Generated by Doxygen
See also
cursor(const Fl_RGB_Image*, int, int), default_cursor()

### 32.153.3.13 default_icon()

```c
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)`

**Note**

See Window icons for the Wayland platform.

### 32.153.3.14 default_icons() [1/2]

```c
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)`

**Note**

See Window icons for the Wayland platform.
32.153.3.15  **default_icons()** [2/2]

```cpp
static void Fl_Window::default_icons (  
    HICON big_icon,  
    HICON small_icon ) [static]
```

Sets the default window icons (Windows platform only).
Convenience function to set the default icons using Windows' native HICON icon handles.
The given icons are copied. You can free the icons immediately after this call.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>big_icon</th>
<th>default large icon for all windows subsequently created</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>small_icon</td>
<td>default small icon for all windows subsequently created</td>
</tr>
</tbody>
</table>

**See also**

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::default_icons(const Fl_RGB_Image *)[[], int]`
- `Fl_Window::icon(const Fl_RGB_Image *)`
- `Fl_Window::icons(const Fl_RGB_Image *)[[], int]`
- `Fl_Window::icons(HICON, HICON)`

32.153.3.16  **default_size_range()**

```cpp
void Fl_Window::default_size_range ( ) [protected]
```

Protected method to calculate the default size range of a window.
This method is called internally prior to showing a window to ensure that the window's size range values are calculated if a `resizable()` widget has been set but `size_range()` has not been called explicitly.
This method does nothing if `size_range()` has been called before.
Otherwise FLTK tries to figure out the window's size range from the setting of the window's `resizable()` widget as follows and roughly in the given order.

1. If `resizable()` is NULL (this is the default) then the window cannot be resized and the resize border and max-size control will not be displayed for the window.
2. If either dimension of `resizable()` is zero, then the window cannot resize in that direction.
3. The `resizable()` widget is clipped to the window area.
4. The non-resizable portion of the window is calculated as the difference of the window's size and the clipped `resizable()` widget's size.
5. If either dimension of the clipped `resizable()` widget is greater than 100, then 100 is considered its minimum width/height. This allows the resizable widget to shrink below its original size.
6. Finally the minimum width/height of the window is set to the non-resizable portion plus the width/height of the `resizable()` widget as calculated above.

In simple words:

- It is assumed that the `resizable()` widget can be indefinitely enlarged and/or shrunk to a minimum width/height of 100 unless it is smaller than that, which is then considered the minimum.
- The window's `size_range()` minimum values are set to the sum of the non-resizable portion of the window and the previously calculated minimum size of the `resizable()` widget.
Examples:

```cpp
Fl_Window win(400, 400);
win.resizable(win);
// win.size_range(100, 100, 0, 0);
The minimum size of the resizable is 100, hence the minimum size of the total window is also 100 in both directions.
```

```cpp
Fl_Window win(400, 400);
Fl_Box box(20, 20, 360, 360);
win.resizable(box);
// win.size_range(140, 140, 0, 0);
The calculated minimum width and height would be 20 + 100 + 20 in both dimensions.
```

```cpp
Fl_Window win(400, 400);
Fl_Box box(200, 0, 500, 300); // note: width 500 too large: clipped
win.resizable(box);
// win.size_range(300, 200, 0, 0);
The width of the resizable is clipped to 200, hence the minimum size of the total window is also 200 (fix) + 100 (min. resizable) in x direction. The minimum value in y direction is 100 (resizable) + 100 (fixed part).
The calculation is based on clipping the resizable widget to the window area to prevent programming errors and the assumption that the resizable widget can be shrunk to 100x100 or its original size, whichever is smaller.
If this is not what you want, please use `Fl_Window::size_range()` explicitly so you can set any appropriate range.

### 32.153.3.17 default_xclass()

```cpp
const char * Fl_Window::default_xclass ( ) [static]
```

Returns the default xclass.

See also

```cpp
Fl_Window::default_xclass(const char * )
```

### 32.153.3.18 default_xclass()

```cpp
void Fl_Window::default_xclass ( const char * xc ) [static]
```

Sets the default window xclass.
The default xclass is used for all windows that don't have their own xclass set before `show()` is called. You can change the default xclass whenever you want, but this only affects windows that are created (and shown) after this call.
The given string `xc` is copied. You can use a local variable or free the string immediately after this call.
If you don't call this, the default xclass for all windows will be "FLTK". You can reset the default xclass by specifying `NULL` for `xc`.
If you call `Fl_Window::xclass(const char * )` for any window, then this also sets the default xclass, unless it has been set before.

Parameters

```cpp
in  xc default xclass for all windows subsequently created
```

See also

```cpp
Fl_Window::xclass(const char * )
```

### 32.153.3.19 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:
32.153 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.

32.153.3.20 force_position() [1/2]

```cpp
template <int Fl_Window::force_position ( ) const [inline], [protected]

Returns the internal state of the window's FORCE_POSITION flag.
Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>flag is set</td>
</tr>
<tr>
<td>0</td>
<td>otherwise</td>
</tr>
</tbody>
</table>

See also
```

32.153.3.21 force_position() [2/2]

```cpp
void Fl_Window::force_position ( int force ) [inline], [protected]

Sets an internal flag that tells FLTK and the window manager to honor position requests.
This is used internally and should not be needed by user code.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in force</td>
<td>1 to set the FORCE_POSITION flag, 0 to clear it</td>
</tr>
</tbody>
</table>

32.153.3.22 free_icons()

```cpp
void Fl_Window::free_icons ( ) [protected]

Deletes all icons previously attached to the window.
See also
```

32.153.3.23 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.
```

32.153.3.24 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

\begin{verbatim}
void Fl_Window::fullscreen_screens()
\end{verbatim}

### 32.153.3.25 fullscreen_screens()

\begin{verbatim}
void Fl_Window::fullscreen_screens {
    int top,
    int bottom,
    int left,
    int right
}
\end{verbatim}

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 \texttt{top}, the bottom of the screen with index \texttt{bottom}, etc. If this method is never called, or if any argument is \texttt{< 0}, then the window will be resized to fill the screen it is currently on.

See also

\begin{verbatim}
void Fl_Window::fullscreen()
\end{verbatim}

### 32.153.3.26 handle()

\begin{verbatim}
int Fl_Window::handle ( int event ) [virtual]
\end{verbatim}

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 \texttt{0} if the widget does not use the event or \texttt{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**

\begin{verbatim}
in event the kind of event received
\end{verbatim}

**Return values**

\begin{verbatim}
0 if the event was not used or understood
1 if the event was used and can be deleted
\end{verbatim}

See also

FL_Event


### 32.153.3.27 hide()

\begin{verbatim}
void Fl_Window::hide () [virtual]
\end{verbatim}
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_Overlay_Window, Fl_Gl_Window, and Fl_Double_Window.

32.153.3.28  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()

32.153.3.29  icon() [1/3]

const void * Fl_Window::icon () const

Gets the current icon window target dependent data.

Deprecated in 1.3.3

32.153.3.30  icon() [2/3]

void Fl_Window::icon (  
    const Fl_RGB_Image * icon )

Sets or resets a single window icon.
A window icon can be changed while the window is shown, but this may be platform and/or window manager
dependent. To be sure that the window displays the correct window icon you should always set the icon before the
window is shown.
If a window icon has not been set for a particular window, then the default window icon (see links below) or the
system default icon will be used.
This method makes an internal copy of the icon pixel buffer, so once set, the Fl_RGB_Image instance can be
freed by the caller.

Parameters

| in | icon | icon for this window, NULL to reset window icon. |

See also

    Fl_Window::default_icon(const Fl_RGB_Image *)
    Fl_Window::default_icons(const Fl_RGB_Image *[], int)
    Fl_Window::icons(const Fl_RGB_Image *[], int)

Note

    See Window icons for the Wayland platform.
32.153.3.31 icon() [3/3]

```cpp
void Fl_Window::icon (const void * ic )
```

Platform-specific method to set the window icon usable on Windows and X11 only.

See Setting the Icon of a Window for its use under X11, and Setting the Icon of a Window under Windows.

**Deprecated** in 1.3.3 in favor of platform-independent methods `Fl_Window::icon(const Fl_RGB_Image *icon)` and `Fl_Window::icons(const Fl_RGB_Image *icons[], int count)`.

32.153.3.32 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.

32.153.3.33 icons() [1/2]

```cpp
void Fl_Window::icons (const Fl_RGB_Image * icons[], int count )
```

Sets the window icons.

You may set multiple window icons with different sizes. Dependent on the platform and system settings the best (or the first) icon will be chosen.

The given images in `icons` are copied. You can use a local variable or free the images immediately after this call. If `count` is zero, current icons are removed. If `count` is greater than zero (must not be negative), then `icons[]` must contain at least `count` valid image pointers (not NULL). Otherwise the behavior is undefined.

**Parameters**

<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 *)`

**Note**

See Window icons for the Wayland platform.

32.153.3.34 icons() [2/2]

```cpp
void Fl_Window::icons (HICON big_icon, HICON small_icon )
```

Sets the window icons using HICON handles (Windows platform only).

The given icons are copied. You can free the icons immediately after this call.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>big_icon</th>
<th>large window icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>small_icon</td>
<td>small window icon</td>
</tr>
</tbody>
</table>

### 32.153.3.35 is_resizable()

```cpp
int Fl_Window::is_resizable ( ) [protected]
```

Protected method to determine whether a window is resizable. If `size_range()` has not yet been called this method calculates the default size range values by calling `default_size_range()`. This method is for internal use only. The returned value is a bit mask and non-zero if the window is resizable in at least one direction.

**Returns**

- non-zero if the window is resizable

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the window is not resizable</td>
</tr>
<tr>
<td>1</td>
<td>the window is resizable in horizontal direction (w)</td>
</tr>
<tr>
<td>2</td>
<td>the window is resizable in vertical direction (h)</td>
</tr>
<tr>
<td>3</td>
<td>the window is resizable in both directions (w and h)</td>
</tr>
</tbody>
</table>

**See also**

- `default_size_range()`

### 32.153.3.36 make_current()

```cpp
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.

### 32.153.3.37 modal()

```cpp
unsigned int Fl_Window::modal ( ) const [inline]
```

Returns true if this window is modal.

### 32.153.3.38 os_id()

```cpp
fl_uintptr_t Fl_Window::os_id ( )
```

Returns a platform-specific identification of a shown window, or 0 if not shown.

**Note**

This identification may differ from the platform-specific reference of an `Fl_Window` object used by functions `fl_x11_xid()`, `fl_mac_xid()`, `fl_x11_find()`, and `fl_mac_find()`.

- X11 platform: the window's XID.
- macOS platform: The window number of the window’s window device.
- other platforms: 0.

### 32.153.3.39 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_Overlay_Window`, `Fl_Gl_Window`, and `Fl_Double_Window`.

### 32.153.3.40 screen_num()

```cpp
void Fl_Window::screen_num {
    int screen_num )
```

Set the number of the screen where to map the window.

Call this and set also the window's desired position before `show()`ing the window. This can be necessary when a system has several screens with distinct scaling factor values because the window's `x()` and `y()` may not suffice to uniquely identify one screen. To see that, consider a system with two screens where the screen at left is A pixel-wide and has a scale factor of 1 whereas the screen at right has a scale factor of 2. For the sake of simplicity, consider only the X coordinates of windows. FLTK coordinates translate directly to pixel coordinates on the left screen, whereas FLTK coordinates multiplied by 2 correspond to pixel coordinates on the right screen. Consequently, FLTK coordinates between A/2 + 1 and A-1 can map to both screens. Both window coordinates and screen number are necessary to uniquely identify where a window is to be mapped.

### 32.153.3.41 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.

### 32.153.3.42 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()`.
32.153.3.43  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.

32.153.3.44  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.

32.153.3.45  shape() [1/2]

void Fl_Window::shape (const Fl_Image & img )

Set the window's shape with an Fl_Image.

See also

void shape(const Fl_Image * img)

32.153.3.46  shape() [2/2]

void Fl_Window::shape (const Fl_Image * img )

Assigns a non-rectangular shape to the window.
This function gives an arbitrary shape (not just a rectangular region) to an Fl_Window. An Fl_Image of any dimen-
sion can be used as mask; it is rescaled to the window's dimension as needed.
The layout and widgets inside are unaware of the mask shape, and most will act as though the window's rectangular
bounding box is available to them. It is up to you to make sure they adhere to the bounds of their masking shape.
The img argument can be an Fl_Bitmap, Fl_Pixmap, Fl_RGB_Image or Fl_Shared_Image:

• With Fl_Bitmap or Fl_Pixmap, the shaped window covers the image part where bitmap bits equal one, or
where the pixmap is not fully transparent.
• With an Fl_RGB_Image with an alpha channel (depths 2 or 4), the shaped window covers the image part that
is not fully transparent.
• With an Fl_RGB_Image of depth 1 (gray-scale) or 3 (RGB), the shaped window covers the non-black image
part.
• With an Fl_Shared_Image, the shape is determined by rules above applied to the underlying image. The
shared image should not have been scaled through Fl_Image::scale().

Platform details:

• On the unix/linux platform, the SHAPE extension of the X server is required. This function does control the
shape of Fl_Gl_Window instances.
• On the Windows platform, this function does nothing with class Fl_Gl_Window.
- On the Mac platform, OS version 10.4 or above is required. An 8-bit shape-mask is used when `img` is an `FL_RGB_Image`: with depths 2 or 4, the image alpha channel becomes the shape mask such that areas with alpha = 0 are out of the shaped window; with depths 1 or 3, white and black are in and out of the shaped window, respectively, and other colors give intermediate masking scores. This function does nothing with class `Fl_Gl_Window`.

The window borders and caption created by the window system are turned off by default. They can be re-enabled by calling `Fl_Window::border(1)`.

A usage example is found at example/shapedwindow.cxx.

Version

1.3.3

32.153.3.47 show() [1/2]

```cpp
template void Fl_Window::show ( ) [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

If the window is already shown then it is restored and raised to the top. This is really convenient because your program can call `show()` at any time, even if the window is already up. It also means that `show()` serves the purpose of `raise()` in other toolkits.

`Fl_Window::show(int argc, char **argv)` is used for top-level windows and allows standard arguments to be parsed from the command-line.

**Note**

For some obscure reasons `Fl_Window::show()` resets the current group by calling `Fl_Group::current(0)`. The comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have unwanted side effects if you `show()` an unrelated window (maybe for an error message or warning) while building a window or any other group widget.

**Todo**

Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

**See also**

`Fl_Window::show(int argc, char **argv)`

Reimplemented from `Fl_Widget`.

Reimplemented in `Fl_Single_Window`, `Fl_Overlay_Window`, `Fl_Gl_Window`, and `Fl_Double_Window`.

32.153.3.48 show() [2/2]

```cpp
template 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>
virtual void Fl_Window::show()

### 32.153.3.49 shown()

```cpp
template <class...> int Fl_Window::shown ( ) [inline]
```

This function returns non-zero if `show()` has been called (but not `hide()`).
You can tell if a window is iconified with `(w->shown()) && !w->visible()`.

### 32.153.3.50 size_range()

```cpp
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.
It is undefined what happens if the current window size does not fit in the constraints passed to `size_range()`.
We recommend to call `size_range()` if you have a `resizable()` widget in a main window.
If this function is not called, FLTK tries to figure out the range. Please see the protected method `default_size_range()` for details.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>minWidth</code></td>
<td>The smallest the window can be. Either value must be greater than 0.</td>
</tr>
<tr>
<td><code>minHeight</code></td>
<td></td>
</tr>
<tr>
<td><code>maxWidth</code></td>
<td>The largest the window can be. If either is equal to the minimum then you cannot resize in that direction. If either is zero then FLTK picks a maximum size in that direction such that the window will fill the screen.</td>
</tr>
<tr>
<td><code>maxHeight</code></td>
<td></td>
</tr>
<tr>
<td><code>deltaX</code></td>
<td>These are size increments. The window will be constrained to widths of <code>minWidth + N * deltaX</code>, where N is any non-negative integer. If these are less or equal to 1 they are ignored (this is always ignored on Windows).</td>
</tr>
<tr>
<td><code>deltaY</code></td>
<td></td>
</tr>
<tr>
<td><code>aspectRatio</code></td>
<td>A flag that indicates that the window should preserve its aspect ratio. This only works if both the maximum and minimum have the same aspect ratio (ignored on Windows and by many X window managers).</td>
</tr>
</tbody>
</table>

### 32.153.3.51 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

```cpp
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_forExpose(); // 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()`.

### 32.153.3.52 xclass() [1/2]

```cpp
const char * Fl_Window::xclass () const
```

Returns the xclass for this window, or a default.

See also

```cpp
Fl_Window::default_xclass(const char *)
Fl_Window::xclass(const char *)
```

### 32.153.3.53 xclass() [2/2]

```cpp
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

```cpp
Fl_Window::default_xclass(const char *)
```

### 32.153.4 Member Data Documentation
32.153.1 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

32.154 Fl_Wizard Class Reference

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes “tabs” under program control.

#include <Fl_Wizard.H>

Inheritance diagram for Fl_Wizard:

```
Fl_Widget
  ↓
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

32.154.1 Detailed Description

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes “tabs” under program control.

Its primary purpose is to support “wizards” that step a user through configuration or troubleshooting tasks.

As with Fl_Tabs, wizard panes are composed of child (usually Fl_Group) widgets. Navigation buttons must be added separately.
32.154.2 Constructor & Destructor Documentation

32.154.2.1 Fl_Wizard()

Fl_Wizard::Fl_Wizard (  
  int xx,  
  int yy,  
  int ww,  
  int hh,  
  const char ∗ l = 0 )  

The constructor creates the Fl_Wizard widget at the specified position and size.
The inherited destructor destroys the widget and its children.

32.154.3 Member Function Documentation

32.154.3.1 next()

void Fl_Wizard::next ( )  
This method shows the next child of the wizard.
If the last child is already visible, this function does nothing.
The documentation for this class was generated from the following files:

• Fl_Wizard.H  
• Fl_Wizard.cxx

32.155 Fl_XBM_Image Class Reference

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

#include <Fl_XBM_Image.H>

Inheritance diagram for Fl_XBM_Image:

```
Fl_XBM_Image  
  |  
  Fl_Bitmap  
  |  
  Fl_Image  
```

Public Member Functions

• Fl_XBM_Image (const char *filename)

  The constructor loads the named XBM file from the given name filename.

Additional Inherited Members

32.155.1 Detailed Description

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

32.155.2 Constructor & Destructor Documentation


32.155.2.1 Fl_XBM_Image()

Fl_XBM_Image::Fl_XBM_Image (const char * name)
The constructor loads the named XBM file from the given name filename. The destructor frees all memory and server resources that are used by the image. The documentation for this class was generated from the following files:

- Fl_XBM_Image.H
- Fl_XBM_Image.cxx

32.156 Fl_XColor Struct Reference

Public Attributes

- unsigned char b
- unsigned char g
- unsigned char mapped
- unsigned long pixel
- unsigned char r

The documentation for this struct was generated from the following file:

- Fl_XColor.H

32.157 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_XPM_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

32.157.1 Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

32.157.2 Constructor & Destructor Documentation
32.157.2.1  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

32.158  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

32.158.1  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

32.159  Fl_Preferences::Name Class Reference

'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on  
the fly.  
#include <Fl_Preferences.H>

Public Member Functions

•  Name (const char *format,...)  
  Creates a group name or entry name on the fly.  
•  Name (unsigned int n)  
  Creates a group name or entry name on the fly.  
•  operator const char * ()  
  Return the Name as a "C" string.

32.159.1  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.
32.159.2 Constructor & Destructor Documentation

32.159.2.1 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], "" );
```

32.159.2.2 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++ )
prefs.get( Fl_Preferences::Name( "File%d", i ), prevFile[i], "" );
```

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

32.160 Fl_Preferences::Node Class Reference

Public Member Functions

- void add (const char *line)
- Node * addChild (const char *path)
- const char * child (int ix)
- Node * childNode (int ix)
- void clearDirtyFlags ()
- void deleteAllChildren ()
- void deleteAllEntries ()
- char deleteEntry (const char *name)
- char dirty ()
- Entry & entry (int i)
- Node * find (const char *path)
- RootNode * findRoot ()
- const char * get (const char *name)
- int getEntry (const char *name)
- const char * name ()
- int nChildren ()
- int nEntry ()
- Node (const char *path)
- Node * parent ()
- const char * path ()
- char remove ()
- Node * search (const char *path, int offset=0)
- void set (const char *line)
- void set (const char *name, const char *value)
- void setParent (Node *parent)
- void setRoot (RootNode *r)
- int write (FILE *f)

Static Public Attributes
- static int lastEntrySet = -1

The documentation for this class was generated from the following files:
- Fl_Preferences.H
- Fl_Preferences.cxx

32.161 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

32.161.1 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

32.162 Fl_Preferences::RootNode Class Reference

Public Member Functions
- char * filename ()
- char getPath (char *path, int pathlen)
- int read ()
- Root root ()
- RootNode (Fl_Preferences *)
- RootNode (Fl_Preferences *, const char *path, const char *vendor, const char *application)
- RootNode (Fl_Preferences *, Root root, const char *vendor, const char *application)
- int write ()

The documentation for this class was generated from the following files:
- Fl_Preferences.H
- Fl_Preferences.cxx

32.163 Fl_Scroll::ScrollInfo Struct Reference

Structure to manage scrollbar and widget interior sizes.
#include <Fl_Scroll.H>
Public Attributes

- `Fl_Region_LRTB child`
  child bounding box: left/right/top/bottom
- `int hneeded`
  horizontal scrollbar visibility
- `Fl_Scrollbar_Data hscroll`
  horizontal scrollbar region + values
- `Fl_Region_XYWH innerbox`
  widget's inner box, excluding scrollbars
- `Fl_Region_XYWH innerchild`
  widget's inner box, including scrollbars
- `int scrollsize`
  the effective scrollbar thickness (local or global)
- `int vneeded`
  vertical scrollbar visibility
- `Fl_Scrollbar_Data vscroll`
  vertical scrollbar region + values

32.163.1 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`

32.164 Fl_Text_Display::Style_Table_Entry Struct Reference

This structure associates the color, font, and font size of a string to draw with an attribute mask matching `attr`.

```c
#include <Fl_Text_Display.H>
```

Public Attributes

- `unsigned attr`
  further attributes for the text style (see `ATTR_BGCOLOR`, etc.)
- `Fl_Color bgcolor`
  text background color if `ATTR_BGCOLOR` or `ATTR_BGCOLOR_EXT` is set
- `Fl_Color color`
  text color
- `Fl_Font font`
  text font
- `Fl_Fontsize size`
  text font size
32.164.1 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 33

File Documentation

33.1 Enumerations.H File Reference

This file contains type definitions and general enumerations.

```c
#include <FL/fl_config.h>
#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h>
```

Macros

Mouse and Keyboard Events

This and the following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events.

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_Alt_L 0xffe9`
  The left alt key.
- `#define FL_Alt_R 0xffea`
  The right alt key.
- `#define FL_Back 0xEF26 /* Like back on a browser */`
- `#define FL_BackSpace 0xff08`
  The backspace key.
- `#define FL_Button 0xfee8`
  A mouse button; use Fl_Button + n for mouse button n.
- `#define FL_Caps_Lock 0xffe5`
  The caps lock key.
- `#define FL_Control_L 0xffe3`
  The lefthand control key.
- `#define FL_Control_R 0xffe4`
  The righthand control key.
- `#define FL_Delete 0xffff`
  The delete key.
- `#define FL_Down 0xff54`
  The down arrow key.
- `#define FL_Eisu 0xff2f`
  The Eisu key of JIS keyboards.
- `#define FL_End 0xff57`
  The end key.
• #define FL_Enter 0xff0d
  The enter key.
• #define FL_Escape 0xff1b
  The escape key.
• #define FL_F 0xffbd
  One of the function keys; use FL_F + n for function key n.
• #define FL_F_Last 0xffe0
  The last function key; use to range-check function keys.
• #define FL_Favorites 0xEF30 /
  ∗ Show favorite locations ∗ 
• #define FL_Forward 0xEF27 /
  ∗ Like forward on a browser ∗ 
• #define FL_Help 0xff68
  The 'help' key on Mac keyboards.
• #define FL_Home 0xff50
  The home key.
• #define FL_Home_Page 0xEF18 /
  ∗ Display user's home page ∗ 
• #define FL_Insert 0xff63
  The insert key.
• #define FL_iso_Key 0xff0c
  The additional key of ISO keyboards.
• #define FL_JIS_Underscore 0xff31
  The underscore key of JIS keyboards.
• #define FL_Kana 0xff2e
  The Kana key of JIS keyboards.
• #define FL_KP 0xff80
  One of the keypad numbers; use FL_KP + 'n' for digit n.
• #define FL_KP_Enter 0xff8d
  The enter key on the keypad, same as Fl_KP+'r'.
• #define FL_KP_Last 0xffbd
  The last keypad key; use to range-check keypad.
• #define FL_Left 0xff51
  The left arrow key.
• #define FL_Mail 0xEF19 /
  ∗ Invoke user's mail program ∗ 
• #define FL_Menu 0xff67
  The menu key.
• #define FL_Meta_L 0xffe7
  The left meta/Windows key.
• #define FL_Meta_R 0xffe8
  The right meta/Windows key.
• #define FL_Num_Lock 0xff7f
  The num lock key.
• #define FL_Page_Down 0xff56
  The page-down key.
• #define FL_Page_Up 0xff55
  The page-up key.
• #define FL_Pause 0xff13
  The pause key.
• #define FL_Print 0xff61
  The print (or print-screen) key.
• #define FL_Refresh 0xEF29 /
  ∗ Refresh the page ∗ 
• #define FL_Right 0xff53
  The right arrow key.
• #define FL_Scroll_Lock 0xff14
  The scroll lock key.
• #define FL_Search 0xEF1B /
  ∗ Search ∗ 
• #define FL_Shift_L 0xffe1
  The lefthand shift key.
• #define FL_Shift_R 0xffe2
  The righthand shift key.
• #define FL_Sleep 0xEF2F /* Put system to sleep */
• #define FL_Stop 0xEF28 /* Stop current operation */
• #define FL_Tab 0xff09
  The tab key.
• #define FL_Up 0xff52
  The up arrow key.
• #define FL_Volume_Down 0xEF11 /* Volume control down */
• #define FL_Volume_Mute 0xEF12 /* Mute sound from the system */
• #define FL_Volume_Up 0xEF13 /* Volume control up */
• #define FL_Yen 0xff30
  The Yen key of JIS keyboards.

Mouse Buttons

These constants define the button numbers for FL_PUSH and FL_RELEASE events.

See also

Fl::event_button()

• #define FL_LEFT_MOUSE 1
  The left mouse button.
• #define FL_MIDDLE_MOUSE 2
  The middle mouse button.
• #define FL_RIGHT_MOUSE 3
  The right mouse button.

Event States

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

• #define FL_ALT 0x00080000
  One of the alt keys is down.
• #define FL_BUTTON(n) (0x00800000<<n)
  Mouse button n (n > 0) is pushed.
• #define FL_BUTTON1 0x01000000
  Mouse button 1 is pushed.
• #define FL_BUTTON2 0x02000000
  Mouse button 2 is pushed.
• #define FL_BUTTON3 0x04000000
  Mouse button 3 is pushed.
• #define FL_BUTTONS 0x7f000000
  Any mouse button is pushed.
• #define FL_CAPS_LOCK 0x00020000
  The caps lock is on.
• #define FL_CTRL 0x00040000
  One of the ctrl keys is down.
• #define FL_KEY_MASK 0x0000ffff
  All keys are 16 bit for now.
• #define FL_META 0x00400000
  One of the meta/Windows keys is down.
• #define FL_NUM_LOCK 0x00100000
  The num lock is on.
• #define FL_SCROLL_LOCK 0x00800000
  The scroll lock is on.
• #define FL_SHIFT 0x00010000
  One of the shift keys is down.
Enumerations

When Conditions

- enum Fl_When {
  FL_WHEN_NEVER = 0, FL_WHEN_CHANGED = 1, FL_WHEN_NOT_CHANGED = 2, FL_WHEN_RELEASE = 4,
  FL_WHEN_RELEASE_ALWAYS = 6, FL_WHEN_ENTER_KEY = 8, FL_WHEN_ENTER_KEY_ALWAYS = 10,
  FL_WHEN_ENTER_KEY_CHANGED = 11
}  
These constants determine when a callback is performed.

Version Numbers

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled. The following constants are defined:

- #define FL_ABI_VERSION FL_API_VERSION  
The FLTK ABI (Application Binary Interface) version number as an int.

- #define FL_API_VERSION (FL_MAJOR_VERSION * 10000 + FL_MINOR_VERSION * 100 + FL_PATCH_VERSION)  
The FLTK API version number as an int.

- 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.

- #define FL_MAJOR_VERSION 1  
The major release version of this FLTK library.

- #define FL_MINOR_VERSION 4  
The minor release version for this library.

- #define FL_PATCH_VERSION 0  
The patch version for this library.

- #define FL_VERSION  
The FLTK version number as a double.

Box Types

FLTK standard box types
This enum defines the standard box types included with FLTK.

Note

The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. _FL_.  
_SHADOW_BOX_. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

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

Todo  Description of box types is incomplete. See below for the defined enum Fl_Boxtype.

See also

src/Fl_get_system_colors.cxx

• Fl_Boxtype fl_box (Fl_Boxtype b)

Get the filled version of a frame.

• enum Fl_Boxtype {  
FL_NO_BOX = 0 , FL_FLAT_BOX , FL_UP_BOX , FL_DOWN_BOX , FL_UP_FRAME , FL_DOWN_FRAME , FL_THIN_UP_BOX , FL_THIN_DOWN_BOX , FL_THIN_UP_FRAME , FL_THIN_DOWN_FRAME , FL_ENGRAVED_BOX , FL_EMBOSSED_BOX , FL_ENGRAVED_FRAME , FL_EMBOSSED_FRAME , FL_BORDER_BOX , FL_BORDER_FRAME , FL_SHADOW_BOX , FL_SHADOW_FRAME , FL_ROUNDED_BOX , FL_ROUNDED_FRAME , FL_RFLAT_BOX , FL_ROUND_UP_BOX , FL_ROUND_DOWN_BOX , FL_ROUND_UP_FRAME , FL_ROUND_DOWN_FRAME , FL_OVAL_BOX , FL_OVAL_FRAME , FL_OFLAT_BOX , FL_PLASTIC_UP_BOX , FL_PLASTIC_DOWN_BOX , FL_PLASTIC_UP_FRAME , FL_PLASTIC_DOWN_FRAME , FL_PLASTIC_THIN_UP_BOX , FL_PLASTIC_THIN_DOWN_BOX , FL_PLASTIC_ROUND_UP_BOX , FL_PLASTIC_ROUND_DOWN_BOX , FL_GTK_UP_BOX , FL_GTK_DOWN_BOX , FL_GTK_UP_FRAME , FL_GTK_DOWN_FRAME , FL_GTK_THIN_UP_BOX , FL_GTK_THIN_DOWN_BOX , FL_GTK_THIN_UP_FRAME , FL_GTK_THIN_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME , FL_GTK_ROUND_UP.Boxe
FLTK standard box types.

- #define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
- FL_LABELTYPE FL_EXPORT fl_define_FL_DIAMOND_BOX()
  Initializes the internal table entry for FL_EMBOSSED_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.Export FL_Boxtype fl_define_FL_GTK_UP_BOX()
  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.
- FL_LABELTYPE FL_EXPORT fl_define_FL_MULTI_LABEL()
  Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
- 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_RFLAT_BOX()
- FL.Export FL_Boxtype fl_define_FL_ROUND_UP_BOX()
- FL.Export FL_Boxtype fl_define_FL_ROUNDED_BOX()
- FL.Export FL_Boxtype fl_define_FL_RSHADOW_BOX()
- FL.Export FL_Boxtype fl_define_FL_SHADOW_BOX()
  Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.

- #define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX
- #define FL_DIAMOND_DOWN_BOX (FL_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)
- #define FL_GLEAM_DOWN_BOX (FL_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  Get the "pressed" or "down" version of a box.
- #define FL_GLEAM_DOWN_FRAME (FL_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  Draws a label with embossed text.
- #define FL_GLEAM_ROUND_DOWN_BOX (FL_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
  Draws a label with engraved text.
- #define FL_GTK_DOWN_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+1)
  Get the unfilled, frame only version of a box.
- #define FL_GTK_DOWN_FRAME (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+1)
- #define FL_GTK_DOWN_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+3)
- #define FL_GTK_DOWN_FRAME (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+3)
- #define FL_GTK_ROUND_DOWN_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)
  #define FL_GTK_ROUND_DOWN_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)
  #define FL_GTK_THIN_DOWN_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+5)
  #define FL_GTK_THIN_DOWN_FRAME (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+5)
  #define FL_GTK_THIN_UP_BOX (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+4)
  #define FL_GTK_THIN_UP_FRAME (FL_Boxtype)(fl_define_FL_GTK_UP_BOX()+6)

Generated by Doxygen
• #define FL_GTK_UP_BOX fl_define_FL_GTK_UP_BOX()
• #define FL_GTK_UP_FRAME (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+2)
• #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.
• enum Fl_Labeltype {
  FL_NORMAL_LABEL = 0 , FL_NO_LABEL , FL_SHADOW_LABEL , FL_ENGRAVED_LABEL ,
  FL_EMBOSSED_LABEL , FL_MULTI_LABEL , FL_ICON_LABEL , FL_IMAGE_LABEL ,
  FL_FREE_LABELTYPE }
  The labeltype() method sets the type of the label.
• #define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()
  Draws a label that can comprise several parts like text and images.
• #define FL_OFLAT_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+3)
• #define FL_OSHADOW_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+1)
• #define FL_OVAL_BOX fl_define_FL_OVAL_BOX()
• #define FL_OVAL_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)
• #define FL_PLASTIC_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+1)
• #define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+3)
• #define FL_PLASTIC_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)
• #define FL_PLASTIC_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
• #define FL_PLASTIC_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+5)
• #define FL_PLASTIC_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+4)
• #define FL_PLASTIC_UP_BOX fl_define_FL_PLASTIC_UP_BOX()
• #define FL_PLASTIC_UP_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+2)
• #define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()
• #define FL_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_ROUND_UP_BOX()+2)
• #define FL_ROUND_UP_BOX fl_define_FL_ROUND_UP_BOX()
• #define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()
• #define FL_ROUNDED_FRAME (Fl_Boxtype)(fl_define_FL_ROUNDED_BOX()+2)
• #define FL_RSHADOW_BOX fl_define_FL_RSHADOW_BOX()
• #define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()
• #define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)
• #define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()
  Draws a label with shadows behind the text.
• #define FL_SYMBOL_LABEL FL_NORMAL_LABEL
  Sets the current label type and returns its corresponding Fl_Labeltype value.

Colors

The Fl_Color type holds an FLTK color value. Colors are either 8-bit indexes into a virtual colormap or 24-bit RGB color values. (See Colors for the default FLTK colormap)
Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBI.

Fl_Color => 0xrrrggbhii
  | | | |
  | | +--- index between 0 and 255
  | | ------ blue color component (8 bit)
  | +------- green component (8 bit)
  +--------- red component (8 bit)

A color can have either an index or an rgb value. Colors with rgb set and an index >0 are reserved for special use.
• const Fl_Color FL_BACKGROUND2_COLOR = 7
  the default background color for text, list, and valuator widgets
• const Fl_Color FL_BACKGROUND_COLOR = 49
• const Fl_Color FL_BLACK = 56
• const Fl_Color FL_BLUE = 216
  typedef unsigned int Fl_Color
  An FLTK color value; see also Colors

• FL_EXPORT Fl_Color fl_color_average (Fl_Color c1, Fl_Color c2, float weight)
  Returns the weighted average color between the two given colors.
• #define FL_COLOR_CUBE (Fl_Color)56
  Fl_Color fl_color_cube (int r, int g, int b)
  Returns a color out of the color cube.
• FL_EXPORT Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg)
  Returns a color that contrasts with the background color.
• const Fl_Color FL_CYAN = 223
• const Fl_Color FL_DARK1 = 47
• const Fl_Color FL_DARK2 = 45
• const Fl_Color FL_DARK3 = 39
• const Fl_Color FL_DARK_BLUE = 136
• const Fl_Color FL_DARK_CYAN = 140
• const Fl_Color FL_DARK_GREEN = 60
• const Fl_Color FL_DARK_MAGENTA = 152
• const Fl_Color FL_DARK_RED = 72
• const Fl_Color FL_DARK_YELLOW = 76
  Fl_Color fl_darker (Fl_Color c)
  Returns a darker version of the specified color.
• const Fl_Color FL_FOREGROUND_COLOR = 0
  the default foreground color (0) used for labels and text
• #define FL_FREE_COLOR (Fl_Color)16
  Colors numbered between FL_FREE_COLOR and FL_FREE_COLOR + FL_NUM_FREE_COLOR - 1 are free for
  the user to be given any value using Fl::set_color().
• #define FL_GRAY FL_BACKGROUND_COLOR
• const Fl_Color FL_GRAY0 = 32
  #define FL_GRAY_RAMP (Fl_Color)32
  Fl_Color fl_gray_ramp (int i)
  Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1).
• const Fl_Color FL_GREEN = 63
• FL_EXPORT Fl_Color fl_inactive (Fl_Color c)
  Returns the inactive, dimmed version of the given color.
• const Fl_Color FL_INACTIVE_COLOR = 8
  the inactive foreground color
• const Fl_Color FL_LIGHT1 = 50
• const Fl_Color FL_LIGHT2 = 52
• const Fl_Color FL_LIGHT3 = 54
  Fl_Color fl_lighter (Fl_Color c)
  Returns a lighter version of the specified color.
• const Fl_Color FL_MAGENTA = 248
• #define FL_NUM_BLUE 5
• #define FL_NUM_FREE_COLOR 16
• #define FL_NUM_GRAY 24
• #define FL_NUM_GREEN 8
• #define FL_NUM_RED 5
• const Fl_Color FL_WHITE = 255
• const Fl_Color FL_YELLOW = 95

Cursors

• enum { FL_READ = 1 , FL_WRITE = 4 , FL_EXCEPT = 8 }
  FD “when” conditions.
• enum Fl_Cursor {
  FL_CURSOR_DEFAULT = 0 , FL_CURSOR_ARROW = 35 , FL_CURSOR_CROSS = 66 , FL_CURSOR_WAIT = 76 ,
  FL_CURSOR_INSERT = 77 , FL_CURSOR_HAND = 31 , FL_CURSOR_HELP = 47 , FL_CURSOR_MOVE = 27 ,
  FL_CURSOR_NS = 78 , FL_CURSOR_WE = 79 , FL_CURSOR_NWS = 80 , FL_CURSOR_NESW = 81 ,
  FL_CURSOR_N = 70 , FL_CURSOR_NE = 69 , FL_CURSOR_NW = 68 , FL_CURSOR_NWSE = 80 , FL_CURSOR_NESW = 81 ,
  FL_CURSOR_S = 9 , FL_CURSOR_SW = 7 , FL_CURSOR_S = 9 , FL_CURSOR_SE = 8 , FL_CURSOR_NONE =255 }
  The following constants define the mouse cursors that are available in FLTK.
• enum Fl_Damage {
  FL_DAMAGE_CHILD = 0x01 , FL_DAMAGE_EXPOSE = 0x02 , FL_DAMAGE_SCROLL = 0x04 ,
  FL_DAMAGE_OVERLAY = 0x08 ,
  FL_DAMAGE_USER1 = 0x10 , FL_DAMAGE_USER2 = 0x20 , FL_DAMAGE_ALL = 0x80 }
  Damage masks.
• #define FL_IMAGE_WITH_ALPHA 0x40000000
• enum Fl_Mode {
  FL_RGB = 0 , FL_INDEX = 1 , FL_SINGLE = 0 , FL_DOUBLE = 2 ,
  FL_ACCUM = 4 , FL_ALPHA = 8 , FL_DEPTH = 16 , FL_STENCIL = 32 ,
  FL_RGB8 = 64 , FL_MULTISAMPLE = 128 , FL_STEREO = 256 , FL_FAKE_SINGLE = 512 ,
  FL_OPENGL3 = 1024 }
  visual types and Fl_Gl_Window::mode() (values match Glut)

Alignment Flags

Flags to control the label alignment.
This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER (0) for most widgets, which centers the label inside the widget.
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):

<table>
<thead>
<tr>
<th>TOP_LEFT</th>
<th>TOP</th>
<th>TOP_RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT_TOP</td>
<td>CENTER</td>
<td>RIGHT</td>
</tr>
<tr>
<td>LEFT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEFT_BOTTOM</td>
<td></td>
<td>RIGHT_BOTTOM</td>
</tr>
</tbody>
</table>

Inside alignments (FL_ALIGN_INSIDE is set):

<table>
<thead>
<tr>
<th>TOP_LEFT</th>
<th>TOP</th>
<th>TOP_RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generated by Doxygen
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
   FTK type for alignment control.
• const Fl_Align FL_ALIGN_BOTTOM = 0x0002
   Align the label at the bottom of the widget.
• const Fl_Align FL_ALIGN_BOTTOM_LEFT = FLALIGN_BOTTOM | FL_ALIGN_LEFT
• const Fl_Align FL_ALIGN_BOTTOM_RIGHT = FLALIGN_BOTTOM | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_CENTER = 0x0000
   Align the label horizontally in the middle.
• 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_IMAGE_BACKDROP = 0x0200
   If the label contains an image, draw the image or deimage in the background.
• const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320
   Mask value to test for image alignment flags.
• const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100
   If the label contains an image, draw the text to the right of the image.
• const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000
   If the label contains an image, draw the text below the image.
• const Fl_Align FL_ALIGN_INSIDE = 0x0010
   Draw the label inside of the widget.
• const Fl_Align FL_ALIGN_LEFT = 0x0004
   Align the label at the left of the widget.
• const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d
   Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.
• const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007
   Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.
• const Fl_Align FL_ALIGN NOWRAP = 0x0000
   Nothing, same as FL_ALIGN_CENTER, for back compatibility.
• const Fl_Align FL_ALIGNPOSITIONMASK = 0x000f
   Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.
• const Fl_Align FL_ALIGN_RIGHT = 0x0008
  Align the label to the right of the widget.
• const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e
  Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.
• const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b
  Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.
• const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120
  If the label contains an image, draw the text to the left of the image.
• const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
  If the label contains an image, draw the text on top of the image.
• const Fl_Align FL_ALIGN_TOP = 0x0001
  Align the label at the top of the widget.
• const Fl_Align FL_ALIGN_TOP_LEFT = FL_ALIGN_TOP | FL_ALIGN_LEFT
• const Fl_Align FL_ALIGN_TOP_RIGHT = FL_ALIGN_TOP | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_WRAP = 0x0080
  Wrap text that does not fit the width of the widget.

Font Numbers
The following constants define the standard FLTK fonts:

• const Fl_Font FL_BOLD = 1
  add this to helvetica, courier, or times
• const Fl_Font FL_BOLD_ITALIC = 3
  add this to helvetica, courier, or times
• const Fl_Font FL_COURIER = 4
  Courier normal.
• const Fl_Font FL_COURIER_BOLD = 5
  Courier bold.
• const Fl_Font FL_COURIER_BOLD_ITALIC = 7
  Courier bold-italic.
• const Fl_Font FL_COURIER_ITALIC = 6
  Courier italic.
• typedef int Fl_Font
  A font number is an index into the internal font table.
• typedef int Fl_Fontsize
  Size of a font in pixels.
• const Fl_Font FL_FREE_FONT = 16
  first one to allocate
• const Fl_Font FL_HELVETICA = 0
  Helvetica (or Arial) normal (0)
• const Fl_Font FL_HELVETICA_BOLD = 1
  Helvetica (or Arial) bold.
• const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3
  Helvetica (or Arial) bold-italic.
• const Fl_Font FL_HELVETICA_ITALIC = 2
  Helvetica (or Arial) oblique.
• const Fl_Font FL_ITALIC = 2
  add this to helvetica, courier, or times
• FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE
  normal font size
• const Fl_Font FL_SCREEN = 13

Generated by Doxygen
Default monospaced screen font.

- const Fl_Font FL_SCREEN_BOLD = 14

Default monospaced bold screen font.

- const Fl_Font FL_SYMBOL = 12

Standard symbol font.

- const Fl_Font FL_TIMES = 8

Times roman.

- const Fl_Font FL_TIMES_BOLD = 9

Times roman bold.

- const Fl_Font FL_TIMES_BOLD_ITALIC = 11

Times roman bold-italic.

- const Fl_Font FL_TIMES_ITALIC = 10

Times roman italic.

- const Fl_Font FL_ZAPF_DINGBATS = 15

Zapf-dingbats font.

### 33.1.1 Detailed Description

This file contains type definitions and general enumerations.

### 33.1.2 Macro Definition Documentation

#### 33.1.2.1 FL_ABI_VERSION

```c
#define FL_ABI_VERSION FL_API_VERSION
```

The FLTK ABI (Application Binary Interface) version number as an int.

FL_ABI_VERSION is an int that describes the major, minor, and patch ABI version numbers in the same format as FL_API_VERSION.

The ABI version number FL_ABI_VERSION is usually the same as the API version FL_API_VERSION with the last two digits set to '00'.

FLTK retains the ABI (Application Binary Interface) during patch releases of the same major and minor versions.

Examples:

<table>
<thead>
<tr>
<th>FLTK Version</th>
<th>FL_API_VERSION</th>
<th>FL_ABI_VERSION</th>
<th>FL_VERSION (deprecated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.0</td>
<td>10300</td>
<td>10300</td>
<td>1.0300</td>
</tr>
<tr>
<td>1.3.4</td>
<td>10304</td>
<td>10300</td>
<td>1.0304</td>
</tr>
</tbody>
</table>

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

#### 33.1.2.2 FL_API_VERSION

```c
#define FL_API_VERSION (FL_MAJOR_VERSION * 10000 + FL_MINOR_VERSION * 100 + FL_PATCH_VERSION)
```

The FLTK API version number as an int.

FL_API_VERSION is an int that describes the major, minor, and patch version numbers.

Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.

The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.
Note

FL_API_VERSION is intended to replace the deprecated double FL_VERSION.

See also

Fl::api_version()

33.1.2.3 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.

33.1.2.4 FL_MAJOR_VERSION

#define FL_MAJOR_VERSION 1

The major release version of this FLTK library.

See also

FL_VERSION

33.1.2.5 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.

33.1.2.6 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

33.1.2.7 FL_PATCH_VERSION

#define FL_PATCH_VERSION 0

The patch version for this library.
FLTK remains binary compatible between patches.

33.1.2.8 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.
33.1.2.9 FL_VERSION

#define FL_VERSION

Value:

\[
\text{(double)FL_MAJOR_VERSION + (double)FL_MINOR_VERSION * 0.01 + (double)FL_PATCH_VERSION * 0.0001)}
\]

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 \((\text{double})\text{FL_API_VERSION} / 10000\).

See also

- Fl::version() (deprecated as well)
- FL_API_VERSION
- Fl::api_version()

33.1.3 Typedef Documentation

33.1.3.1 Fl_Fontsize

typedef int Fl_Fontsize

Size of a font in pixels. This is the approximate height of a font in pixels.

33.1.4 Enumeration Type Documentation

33.1.4.1 anonymous enum

anonymous enum

FD "when" conditions.

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerant</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>

33.1.4.2 Fl_Boxtype

enum Fl_Boxtype

FLTK standard box types. This enum defines the standard box types included with FLTK.
33.1 Enumerations.H File Reference

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!

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NO_BOX</td>
<td>nothing is drawn at all, this box is invisible</td>
</tr>
<tr>
<td>FL_FLAT_BOX</td>
<td>a flat box</td>
</tr>
<tr>
<td>FL_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_UP_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_EMBOSSED_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_EMBOSSED_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_BORDER_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>_FL_SHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_SHADOW_BOX</td>
</tr>
<tr>
<td>_FL_SHADOW_FRAME</td>
<td>see figure Standard Box Types, use FL_SHADOW_FRAME</td>
</tr>
<tr>
<td>_FL_ROUNDED_BOX</td>
<td>see figure Standard Box Types, use FL_ROUNDED_BOX</td>
</tr>
<tr>
<td>_FL_RSHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_RSHADOW_BOX</td>
</tr>
<tr>
<td>_FL_ROUNDED_FRAME</td>
<td>see figure Standard Box Types, use FL_ROUNDED_FRAME</td>
</tr>
<tr>
<td>_FL_RFLAT_BOX</td>
<td>see figure Standard Box Types, use FL_RFLAT_BOX</td>
</tr>
<tr>
<td>_FL_ROUND_UP_BOX</td>
<td>see figure Standard Box Types, use FL_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_ROUND_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_UP_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_UP_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OVAL_BOX</td>
<td>see figure Standard Box Types, use FL_OVAL_BOX</td>
</tr>
<tr>
<td>_FL_OSHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_OSHADOW_BOX</td>
</tr>
<tr>
<td>_FL_OVAL_FRAME</td>
<td>see figure Standard Box Types, use FL_OVAL_FRAME</td>
</tr>
<tr>
<td>_FL_OFLAT_BOX</td>
<td>see figure Standard Box Types, use FL_OFLAT_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_BOX</td>
<td>plastic version of FL_UP_BOX, use FL_PLASTIC_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_BOX</td>
<td>plastic version of FL_DOWN_BOX, use FL_PLASTIC_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_FRAME</td>
<td>plastic version of FL_UP_FRAME, use FL_PLASTIC_UP_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_FRAME</td>
<td>plastic version of FL_DOWN_FRAME, use FL_PLASTIC_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_UP_BOX</td>
<td>plastic version of FL_THIN_UP_BOX, use FL_PLASTIC_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_DOWN_BOX</td>
<td>plastic version of FL_THIN_DOWN_BOX, use FL_PLASTIC_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_UP_BOX</td>
<td>plastic version of FL_ROUND_UP_BOX, use FL_PLASTIC_ROUND_UP_BOX</td>
</tr>
</tbody>
</table>

Generated by Doxygen
### Enumerator

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FL_PLASTIC_ROUND_DOWN_BOX</td>
<td>plastic version of FL_ROUND_DOWN_BOX, use FL_PLASTIC_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_BOX</td>
<td>gtk+ version of FL_UP_BOX, use FL_GTK_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_BOX</td>
<td>gtk+ version of FL_DOWN_BOX, use FL_GTK_DOWN BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_FRAME</td>
<td>gtk+ version of FL_UP_FRAME, use FL_GTK_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_FRAME</td>
<td>gtk+ version of FL_DOWN_FRAME, use FL_GTK_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_BOX</td>
<td>gtk+ version of FL_THIN_UP_BOX, use FL_GTK_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_BOX</td>
<td>gtk+ version of FL_THIN_DOWN_BOX, use FL_GTK_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_FRAME</td>
<td>gtk+ version of FL_THIN_UP_FRAME, use FL_GTK_THIN_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_FRAME</td>
<td>gtk+ version of FL_THIN_DOWN_FRAME, use FL_GTK_THIN_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_ROUND_UP_BOX</td>
<td>gtk+ version of FL_ROUND_UP_BOX, use FL_GTK_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_ROUND_DOWN_BOX</td>
<td>gtk+ version of FL_ROUND_DOWN_BOX, use FL_GTK_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_UP_BOX</td>
<td>gleam version of FL_UP_BOX, use FL_GLEAM_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_BOX</td>
<td>gleam version of FL_DOWN_BOX, use FL_GLEAM_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_UP_FRAME</td>
<td>gleam version of FL_UP_FRAME, use FL_GLEAM_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_FRAME</td>
<td>gleam version of FL_DOWN_FRAME, use FL_GLEAM_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_UP_BOX</td>
<td>gleam version of FL_THIN_UP_BOX, use FL_GLEAM_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_DOWN_BOX</td>
<td>gleam version of FL_THIN_DOWN_BOX, use FL_GLEAM_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_UP_BOX</td>
<td>gleam version of FL_ROUND_UP_BOX, use FL_GLEAM_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_DOWN_BOX</td>
<td>gleam version of FL_ROUND_DOWN_BOX, use FL_GLEAM_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>FL_FREE_BOXTYPE</td>
<td>the first free box type for creation of new box types</td>
</tr>
</tbody>
</table>

### 33.1.4.3 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.
33.1 Enumerations.H File Reference

### 33.1.4.4 Fl_Damage

enum Fl_Damage
Damage masks.

**Enumerator**

<table>
<thead>
<tr>
<th>Fl_Damage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_DAMAGE_CHILD</td>
<td>A child needs to be redrawn.</td>
</tr>
<tr>
<td>FL_DAMAGE_EXPOSE</td>
<td>The window was exposed.</td>
</tr>
<tr>
<td>FL_DAMAGE_SCROLL</td>
<td>The Fl_Scroll widget was scrolled.</td>
</tr>
<tr>
<td>FL_DAMAGE_OVERLAY</td>
<td>The overlay planes need to be redrawn.</td>
</tr>
<tr>
<td>FL_DAMAGE_USER1</td>
<td>First user-defined damage bit.</td>
</tr>
<tr>
<td>FL_DAMAGE_USER2</td>
<td>Second user-defined damage bit.</td>
</tr>
<tr>
<td>FL_DAMAGE_ALL</td>
<td>Everything needs to be redrawn.</td>
</tr>
</tbody>
</table>

### 33.1.4.5 Fl_Event

enum Fl_Event
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

Events can also come from other programs like the window manager.

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

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

See also

- Fl: :event_text(), Fl: :event_key(), class Fl::

Generated by Doxygen
### Enumerator

<table>
<thead>
<tr>
<th>FL_NO_EVENT</th>
<th>No event.</th>
</tr>
</thead>
<tbody>
<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 <code>Fl::event_button()</code>. You find out the mouse position by calling <code>Fl::event_x()</code> and <code>Fl::event_y()</code>. A widget indicates that it &quot;wants&quot; the mouse click by returning non-zero from its <code>Fl_Widget::handle()</code> method. It will then become the <code>Fl::pushed()</code> widget and will get FL_DRAG and the matching FL_RELEASE events. If <code>Fl_Widget::handle()</code> 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 <code>Fl::event_button()</code>. 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 <code>Fl::belowmouse()</code> 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 <code>Fl::event_state()</code>. The mouse position is in <code>Fl::event_x()</code> and <code>Fl::event_y()</code>. 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 <code>Fl::focus()</code> 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 <code>Fl::event_key()</code> 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 <code>Fl::focus()</code> widget when another widget gets the focus or the window loses focus.</td>
</tr>
</tbody>
</table>
## Enumerations

### FL_KEYDOWN
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.

### FL_KEYBOARD
Equivalent to FL_KEYDOWN.
See also FL_KEYDOWN

### FL_KEYUP
Key release event.
See also FL_KEYDOWN

### FL_CLOSE
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().

### 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.

### 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.

### FL_DEACTIVATE
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).

### FL_ACTIVATE
This widget is now active, due to Fl_Widget::activate() being called on it or one of its parents.
### Enumerator

<table>
<thead>
<tr>
<th>Enum Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_HIDE</td>
<td>This widget is no longer visible, due to <code>Fl_Widget::hide()</code> being called on it or one of its parents, or due to a parent window being minimized. <code>Fl_Widget::visible()</code> may still be true after this, but the widget is visible only if visible() is true for it and all its parents (use <code>Fl_Widget::visible_r()</code> to check this).</td>
</tr>
<tr>
<td>FL_SHOW</td>
<td>This widget is visible again, due to <code>Fl_Widget::show()</code> being called on it or one of its parents, or due to a parent window being restored. Child <code>Fl_Windows</code> respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass <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>

### 33.1.4.6 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>Enumerators</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_FREE_LABELTYPE</td>
<td>first free labeltype to use for creating own labeltypes</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>
</tbody>
</table>

See also

Fl_Multi_Label

33.1.4.7 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>Enumerators</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>= (FL_WHEN_ENTER_KEY</td>
</tr>
</tbody>
</table>
33.1.5 Function Documentation

33.1.5.1 fl_box()

`fl_box` (``Fl_Boxtype``) [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.

33.1.5.2 fl_color_cube()

`fl_color_cube` (``Fl_Color``, ``int``, ``int``, ``int``) [inline]

Returns a color out of the color cube. 
`r` must be in the range 0 to FL_NUM_RED (5) minus 1, 
`g` must be in the range 0 to FL_NUM_GREEN (8) minus 1, 
`b` must be in the range 0 to FL_NUM_BLUE (5) minus 1.
To get the closest color to a 8-bit set of R,G,B values use:

```
fl_color_cube(R * (FL_NUM_RED - 1) / 255,
               G * (FL_NUM_GREEN - 1) / 255,
               B * (FL_NUM_BLUE - 1) / 255);
```

33.1.5.3 fl_define_FL_EMBOSSED_LABEL()

`fl_define_FL_EMBOSSED_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value. Internal use only.

33.1.5.4 fl_define_FL_ENGRAVED_LABEL()

`fl_define_FL_ENGRAVED_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value. Internal use only.

33.1.5.5 fl_define_FL_ICON_LABEL()

`fl_define_FL_ICON_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value. Internal use only.

33.1.5.6 fl_define_FL_IMAGE_LABEL()

`fl_define_FL_IMAGE_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value. Internal use only.

33.1.5.7 fl_define_FL_MULTI_LABEL()

`fl_define_FL_MULTI_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value. Internal use only.

33.1.5.8 fl_define_FL_SHADOW_LABEL()

`fl_define_FL_SHADOW_LABEL` (``Fl_Labeltype``) [FL_EXPORT]

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
Internal use only.

### 33.1.5.9 fl_down()

```c
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.

### 33.1.5.10 fl_frame()

```c
Fl_Boxtype fl_frame (Fl_Boxtype b) [inline]
```

Get the unfilled, frame only version of a box. If no frame version of a given box exists, the behavior of this function is undefined and some random box or frame is returned.

### 33.1.5.11 fl_gray_ramp()

```c
Fl_Color fl_gray_ramp (int i) [inline]
```

Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1). FL_NUM_GRAY is defined to be 24 in the current FLTK release. To get the closest FLTK gray value to an 8-bit grayscale color I use:

```
fl_gray_ramp(I * (FL_NUM_GRAY - 1) / 255)
```

### 33.1.6 Variable Documentation

#### 33.1.6.1 FL_ALIGN_LEFT

```c
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.

#### 33.1.6.2 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.

#### 33.1.6.3 FL_NORMAL_SIZE

```c
FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE [extern]
```

Normal font size

### 33.2 filename.H File Reference

File names and URI utility functions.

```
#include "Fl_Export.H"
#include <FL/platform_types.h>
```
Macros

• #define FL_PATH_MAX 2048
  all path buffers should use this length

Typedefs

• typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
  File sorting function.

Functions

• 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).

33.2.1 Detailed Description

File names and URI utility functions.

33.3 Fl.cxx File Reference

Implementation of the member functions of class Fl.
#include <FL/Fl.H>
#include <FL/platform.H>
#include "FL_Screen_Driver.H"
#include "Fl_Window_Driver.H"
#include "Fl_System_Driver.H"
#include "Fl_Timeout.h"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <ctype.h>

Generated by Doxygen
#include <stdlib.h>
#include "flstring.h"

Macros

- \#define FOREVER 1e20

Functions

- bool fl_clipboard_notify_empty (void)
- void fl_close_display ()
- void fl_disable_wayland ()
  
  Prevent the FLTK library from using its wayland backend.
- const char * fl_filename_name (const char *name)
  
  Gets the file name from a path.
- Fl_Window * fl_find (Window xid)
  
  Returns the Fl_Window that corresponds to the given window reference, or NULL if not found.
- void fl_fix_focus ()
- void fl_open_callback (void (*)(const char *cb))(const char *))
  
  Register a function called for each file dropped onto an application icon.
- void fl_open_display ()
  
  Opens the display.
- int fl_send_system_handlers (void *e)
- void fl_throw_focus (Fl_Widget *o)
- void fl_trigger_clipboard_notify (int source)
- FL_EXPORT Window fl_xid_(const Fl_Window *w)

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

33.3.1 Detailed Description
Implementation of the member functions of class Fl.

33.3.2 Function Documentation

33.3.2.1 fl_disable_wayland()

void fl_disable_wayland ()

Prevent the FLTK library from using its wayland backend.
Call this early in your main(), before fl_open_display() runs, or any window is created, or the screen is accessed.
This function has no effect on non-Wayland platforms.
33.3.2.2 fl_find()

```c
FL_Window* fl_find(
    Window xid
)
```

Returns the Fl_Window that corresponds to the given window reference, or NULL if not found.

**Deprecated** Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wld_window*), fl_win32_find(HWND) or fl_mac_find(FLWindow*) with FLTK 1.4.0 and above.

33.3.2.3 fl_open_display()

```c
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.

33.4 Fl.H File Reference

**Fl** static class.

```c
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/platform_types.h>
#include <FL/fl_casts.H>
#include <FL/Fl_Cairo.H>
#include "fl_utf8.h"
#include "Enumerations.H"
#include <string.h>
```

**Classes**

- **class Fl**
  
  *The Fl is the FLTK global (static) class containing state information and global methods for the current application.*

- **class Fl_Widget_Tracker**
  
  *This class should be used to control safe widget deletion.*

**Macros**

- **#define Fl_Object Fl_Widget**
  
  *for back compatibility - use Fl_Widget!*

**Typedefs**

- **typedef void(* Fl_Abort_Handler) (const char *format,...)**
  
  *Signature of set_abort functions passed as parameters.*

- **typedef int(* Fl_Args_Handler) (int argc, char **argv, int &i)**
  
  *Signature of args functions passed as parameters.*

- **typedef void(* Fl_Atclose_Handler) (Fl_Window *window, void *data)**
  
  *Signature of set_atclose functions passed as parameters.*

- **typedef void(* Fl_Awake_Handler) (void *data)**
  
  *Signature of some wakeup callback functions passed as parameters.*

- **typedef void(* Fl_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color)**
  
  *Signature of some box drawing functions passed as parameters.*

- **typedef void(* Fl_Clipboard_Notify_Handler) (int source, void *data)**
  
  *Signature of some clipboard notify functions passed as parameters.*
33.4 Fl.H File Reference

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.

Functions

- `FL_EXPORT void fl_disable_wayland()`
  Prevent the FLTK library from using its wayland backend.

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

33.4.1 Detailed Description

`Fl` static class.

33.4.2 Function Documentation

33.4.2.1 fl_disable_wayland()

`FL_EXPORT void fl_disable_wayland();`
Prevent the FLTK library from using its wayland backend.
Call this early in your `main()`, before `fl_open_display()` runs, or any window is created, or the screen is accessed.
This function has no effect on non-Wayland platforms.
33.5  fl_arc.cxx File Reference

Utility functions for drawing arcs and circles.
#include <FL/fl_draw.H>
#include <FL/math.h>

33.5.1 Detailed Description

Utility functions for drawing arcs and circles.

33.6  fl_ask.cxx File Reference

Utility functions for common dialogs.
#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Input_.H>
#include "flstring.h"
#include "Fl_Screen_Driver.H"
#include <FL/fl_ask.H>
#include "Fl_Message.h"
#include <stdio.h>
#include <stdarg.h>

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_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2, ...)
  Shows a dialog displaying the printf style fmt message.
- const char * fl_input (const char *fmt, const char *defstr, ...)
  Shows an input dialog displaying the fmt message with variable arguments.
- Fl_String fl_input_str (int maxchar, const char *fmt, const char *defstr, ...)
  Shows an input dialog displaying the fmt message with variable arguments.
- void fl_message (const char *fmt, ...)
  Shows an information message dialog box.
- int fl_message_hotspot ()
  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- void fl_message_hotspot (int enable)
  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- Fl_Widget * fl_message_icon ()
  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char *str)
  Sets the icon label of the dialog window used in many common dialogs.
• **void fl_message_position** (const int x, const int y, const int center)
  
  *Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().*

• **void fl_message_position** (Fl_Widget *widget)
  
  *Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().*

• **int fl_message_position** (int *x, int *y)
  
  *Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().*

• **void fl_message_title** (const char *title)
  
  *Sets the title of the dialog window used in many common dialogs.*

• **void fl_message_title_default** (const char *title)
  
  *Sets the default title of the dialog window used in many common dialogs.*

• **const char * fl_password** (const char *fmt, const char *defstr,...)
  
  *Shows an input dialog displaying the fmt message with variable arguments.*

• **Fl_String fl_password_str** (int maxchar, const char *fmt, const char *defstr,...)
  
  *Shows an input dialog displaying the fmt message with variable arguments.*

### Variables

- **const char * fl_cancel** = "Cancel"
  
  *string pointer used in common dialogs, you can change it to another language*

- **const char * fl_close** = "Close"
  
  *string pointer used in common dialogs, you can change it to another language*

- **FL_Font fl_message_font_** = FL_HELVETICA

- **FL_Fontsize fl_message_size_** = -1

- **const char * fl_no** = "No"
  
  *string pointer used in common dialogs, you can change it to another language*

- **const char * fl_ok** = "OK"
  
  *string pointer used in common dialogs, you can change it to another language*

- **const char * fl_yes** = "Yes"
  
  *string pointer used in common dialogs, you can change it to another language*

### 33.6.1 Detailed Description

Utility functions for common dialogs.

This file defines the functions

- fl_alert()
- fl_beep()
- fl_message()
- fl_ask()
- fl_choice()
- fl_input()
- fl_input_str()
- fl_password()
- fl_password_str()

and some more functions to change their behavior (positioning, window title, and more).

Since FLTK 1.4.0 a big part of these functions is implemented in class Fl_Message.
33.7  fl_ask.H File Reference

API for common dialogs.
#include <FL/Enumerations.H>
#include <FL/Fl_String.H>
#include <FL/fl_attr.h>

Enumerations

• enum Fl_Beep {
  FL_BEEP_DEFAULT = 0 , FL_BEEP_MESSAGE , FL_BEEP_ERROR , FL_BEEP_QUESTION ,
  FL_BEEP_PASSWORD , FL_BEEP_NOTIFICATION
}

Different system beeps available.

Functions

• FL_EXPORT void FL_EXPORT void fl_alert (const char *,...) __fl_attr((__format__(__printf__

• FL_EXPORT void FL_EXPORT void FL_EXPORT int fl_ask (const char *,...) __fl_attr((__format__(__printf__

• FL_EXPORT void fl_beep (int type=FL_BEEP_DEFAULT)

Emits a system beep message.

• FL_EXPORT int fl_choice (const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__

• FL_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT int FL_EXPORT Fl_String

• FL_EXPORT void fl_message_font (Fl_Font f, Fl_Fontsize s)

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

• FL_EXPORT void fl_message_hotspot (int enable)

Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

• FL_EXPORT int fl_message_hotspot (void)

Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

• FL_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT int FL_EXPORT Fl_String

• FL_EXPORT void fl_message_position (const int x, const int y, const int center=0)

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

• FL_EXPORT void fl_message_position (Fl_Widget &widget)

 Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

• FL_EXPORT int fl_message_position (int *x=0, int *y=0)

Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

• FL_EXPORT void fl_message_title (const char *title)
Sets the title of the dialog window used in many common dialogs.

- FL_EXPORT void fl_message_title_default (const char *title)

Sets the default title of the dialog window used in many common dialogs.

- FL_EXPORT int FL_EXPORT const char FL_EXPORT const char * fl_password (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- FL_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT Fl_String FL_EXPORT Fl_String fl_password_str (int maxchar, const char *label, const char *deflt=0,...) __fl_attr(__format__(__printf__

Variables

- FL_EXPORT void FL_EXPORT void FL_EXPORT int __deprecated__
- FL_EXPORT const char * fl_cancel
  string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * fl_close
  string pointer used in common dialogs, you can change it to another language
- FL_EXPORT Fl_Font fl_message_font
- FL_EXPORT Fl_Fontsize fl_message_size
- FL_EXPORT const char * fl_no
  string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * fl_ok
  string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * fl_yes
  string pointer used in common dialogs, you can change it to another language

33.7.1 Detailed Description

API for common dialogs.

33.7.2 Enumeration Type Documentation

33.7.2.1 Fl_Beep

enum Fl_Beep
Different system beeps available.

See also
fl_beep(int)

Enumerator

<table>
<thead>
<tr>
<th>Fl_Beep</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>

33.7.3 Function Documentation

Generated by Doxygen
33.7.3.1 fl_message_position()

```c
void fl_message_position (Fl_Widget & widget) [inline]
```

See also

`fl_message_position(Fl_Widget *widget)`.

33.8 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, int, Fl_Color)`
  Draws a box of type FL_NO_BOX.
33.8.1 Detailed Description

drawing code for common box types.

33.8.2 Function Documentation

33.8.2.1 fl_internal_boxtype()

void fl_internal_boxtype (  
    Fl_Boxtype t,  
    Fl_Box_Draw_F * f )

Sets the drawing function for a given box type.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>f</td>
<td>box drawing function</td>
</tr>
</tbody>
</table>

33.8.2.2 fl_rectbound()

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.

33.9 fl_color.cxx File Reference

Color handling.

#include <FL/Fl.H>
#include <FL/Fl_Device.H>
#include <FL/Fl_Graphics_Driver.H>
#include "fl_cmap.h"

## Functions

- **Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)**
  Returns the weighted average color between the two given colors.
- **Fl_Color fl_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

- **FL_EXPORT unsigned fl_cmap [256]**

### 33.9.1 Detailed Description
Color handling.

### 33.9.2 Variable Documentation

#### 33.9.2.1 fl_cmap

**FL_EXPORT unsigned fl_cmap[256]**

Initial value:
```
{
}
```

### 33.10 Fl_Color_Chooser.H File Reference

**Fl_Color_Chooser widget**
_INCLUDED &lt;FL/Fl_Group.H&gt; 
_INCLUDED &lt;FL/Fl_Box.H&gt; 
_INCLUDED &lt;FL/Fl_Return_Button.H&gt; 
_INCLUDED &lt;FL/Fl_Choice.H&gt; 
_INCLUDED &lt;FL/Fl_Value_Input.H&gt; 

## Classes

- **class Fl_Color_Chooser**
  The **Fl_Color_Chooser** widget provides a standard RGB color chooser.

### 33.10.1 Detailed Description

**Fl_Color_Chooser widget**

### 33.11 Fl_compose.cxx File Reference

Utility functions to support text input.
_INCLUDED &lt;FL/Fl.H&gt; 
_INCLUDED "Fl_Screen_Driver.H"
33.11.1 Detailed Description
Utility functions to support text input.

33.12 fl_curve.cxx File Reference
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.
#include <FL/fl_draw.H>
#include <math.h>

33.12.1 Detailed Description
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path. Incremental math implementation: I very much doubt this is optimal! From Foley/vanDam page 511. If anybody has a better algorithm, please send it!

33.13 Fl_Device.H File Reference
declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.
#include <FL/FL_Plugin.H>
#include <FL/platform_types.h>

Classes

• class Fl_Device_Plugin
  
  This plugin socket allows the integration of new device drivers for special window or screen types.

• class Fl_Display_Device
  
  A display to which the computer can draw.

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

33.13.1 Detailed Description
declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

33.14 Fl_Double_Window.cxx File Reference
Fl_Double_Window implementation.
#include <FL/Fl.H>
#include <FL/platform.H>
#include <FL/Fl_Double_Window.H>
#include <FL/fl_draw.H>
#include "Fl_Window_Driver.H"

33.14.1 Detailed Description
Fl_Double_Window implementation.
33.15  fl_draw.H File Reference

utility header to pull drawing functions together
#include <FL/Enumerations.H>
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl_Rect.H>

Enumerations

- enum {
  FL_SOLID = 0, FL_DASH = 1, FL_DOT = 2, FL_DASHDOT = 3,
  FL_DASHDOTDOT = 4, FL_CAP_FLAT = 0x100, FL_CAP_ROUND = 0x200, FL_CAP_SQUARE = 0x300,
  FL_JOIN_MITER = 0x1000, FL_JOIN_ROUND = 0x2000, FL_JOIN_BEVEL = 0x3000 }

Functions

- FL_EXPORT int fl_add_symbol (const char *name, void(*drawit)(Fl_Color), int scalable)
  Adds a symbol to the system.
- int fl_antialias ()
  Return whether line drawings are currently antialiased.
- void fl_antialias (int state)
  Turn antialiased line drawings ON or OFF, if supported by platform.
- void fl_arc (double x, double y, double r, double start, double end)
  Add a series of points to the current path on the arc of a circle.
- void fl_arc (int x, int y, int w, int h, double a1, double a2)
  Draw ellipse sections using integer coordinates.
- void fl_begin_complex_polygon ()
  Start drawing a complex filled polygon.
- void fl_begin_line ()
  Start drawing a list of lines.
- void fl_begin_loop ()
  Start drawing a closed sequence of lines.
- FL_EXPORT void fl_begin_offscreen (Fl_Offscreen ctx)
  Send all subsequent drawing commands to this offscreen buffer.
- void fl_begin_points ()
  Start drawing a list of points.
- void fl_begin_polygon ()
  Start drawing a convex filled polygon.
- char fl_can_do_alpha_blending ()
  Check whether platform supports true alpha blending for RGBA images.
- FL_EXPORT Fl_RGB_Image * fl_capture_window (Fl_Window *win, int x, int y, int w, int h)
  Captures the content of a rectangular zone of a mapped window.
- 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.
- void fl_clip (int x, int y, int w, int h)
  Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated).
- int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)
  Intersect a rectangle with the current clip region and return the bounding box of the result.
- FL_Region fl_clip_region ()
  Return the current clipping region.
• void fl_clip_region (Fl_Region r)
  Replace the top of the clipping stack with a clipping region of any shape.

• Fl_Color fl_color ()
  Return the last fl_color() that was set.

• void fl_color (Fl_Color c)
  Set the color for all subsequent drawing operations.

• void fl_color (int c)
  for back compatibility - use fl_color(Fl_Color c) instead

• void fl_color (uchar r, uchar g, uchar b)
  Set the color for all subsequent drawing operations.

• void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  Copy a rectangular area of the given offscreen buffer into the current drawing destination.

• FL_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)
  Add a series of points on a Bézier curve to the path.

• FL_EXPORT void fl_delete_offscreen (Fl_Offscreen ctx)
  Deletion of an offscreen graphics buffer.

• int fl_descent ()
  Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.

• void fl_draw (const char *str, int n, int x, int y)
  Draws starting at the given x, y location a UTF-8 string of length n bytes.

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

• FL_EXPORT void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)
  Fancy string drawing function which is used to draw all the labels.

• FL_EXPORT void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, void(*callthis)(const char *, int, int, int), Fl_Image *img=0, int draw_symbols=1)
  The same as fl_draw(const char *, int, int, int, Fl_Align, Fl_Image *, int) with the addition of the callthis parameter, which is a pointer to a text drawing function such as fl_draw(const char *, int, int, int) to do the real work.

• void fl_draw (int angle, const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Imagew *img=0, int draw_symbols=1)
  Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

• FL_EXPORT void fl_draw (int angle, const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Imagew *img=0, int draw_symbols=1)
  Draw a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

• FL_EXPORT void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)
  Draws a box using given type, position, size and color.

• void fl_draw_check (Fl_Rect bb, Fl_Color col)
  Draw a check mark inside the given bounding box.

• void fl_draw_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)
  Draw an 8-bit per color RGB or luminance image.

• void fl_draw_image (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=3)
  Draw an image using a callback function to generate image data.

• void fl_draw_image_mono (const uchar *buf, int X, int Y, int W, int H, int D=1, int L=0)
  Draw a gray-scale (1 channel) image.

• void fl_draw_image_mono (Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D=1)
  Draw a gray-scale image using a callback function to generate image data.
• int fl_draw_pixmap (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_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.

• 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 ()
  End complex filled polygon, and draw.

• void fl_end_line ()
  End list of lines, and draw.

• void fl_end_loop ()
  End closed sequence of lines, and draw.

• FL_EXPORT void fl_end_offscreen ()
  Quit sending drawing commands to the current offscreen buffer.

• void fl_end_points ()
  End list of points, and draw.

• void fl_end_polygon ()
  End convex filled polygon, and draw.

• 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_Font fl_font ()
  Return the face set by the most recent call to fl_font().

• FL_EXPORT void fl_font (Fl_Font face, Fl_Fontsize fsize)
  Sets the current font, which is then used in various drawing routines.

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

• FL_EXPORT void fl_frame2 (const char *s, int x, int y, int w, int h)
  Draws a series of line segments around the given box.

• void fl_gap ()
  Separate loops of the path.

• int fl_height ()
  Return 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)
  Convert text from Windows/X11 latin1 character set to local encoding.

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

• void fl_line (int x, int y, int x1, int y1)
  Draw a line from (x,y) to (x1,y1)

• void fl_line (int x, int y, int x1, int y1, int x2, int y2)
  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)

• void fl_line_style (int style, int width=0, char *dashes=0)
  Set how to draw lines (the "pen").

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

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

• void fl_loop (int x, int y, int x1, int y1, int x2, int y2)
  Outline a 3-sided polygon with lines.
• void fl_loop (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Outline a 4-sided polygon with lines.

• FL_EXPORT const char * fl_mac_roman_to_local (const char *t, int n=-1)
  Convert 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_measurePixmap (char *const *data, int &w, int &h)
  Get the dimensions of a pixmap.

• FL_EXPORT int fl_measurePixmap (const char *const *cdata, int &w, int &h)
  Get the dimensions of a pixmap.

• void fl_mult_matrix (double a, double b, double c, double d, double x, double y)
  Concatenate another transformation onto the current one.

• int fl_not_clipped (int x, int y, int w, int h)
  Does the rectangle intersect the current clip region?

• FL_EXPORT unsigned int fl_oldShortcut (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.

• FL_EXPORT float fl_override_scale ()
  Removes any GUI scaling factor in subsequent drawing operations.

• 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)
  Draw a single pixel at the given coordinates.

• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2)
  Fill a 3-sided polygon.

• void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  Fill a 4-sided polygon.

• void fl_pop_clip ()
  Restore the previous clip region.

• void fl_pop_matrix ()
  Restore the current transformation matrix from the stack.

• void fl_push_clip (int x, int y, int w, int h)
  Intersect the current clip region with a rectangle and push this new region onto the stack.

• void fl_push_matrix ()
  Save the current transformation matrix on the stack.

• void fl_push_no_clip ()
  Push an empty clip region onto the stack so nothing will be clipped.

• 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 (Fl_Rect r)
  Draw a 1-pixel border inside the given bounding box.

• void fl_rect (int x, int y, int w, int h)
  Draw a 1-pixel border inside the given bounding box.

• void fl_rect (int x, int y, int w, int h, Fl_Color c)
  Draw with passed color a 1-pixel border inside the given bounding box.

• void fl_rectf (Fl_Rect bb, uchar r, uchar g, uchar b)
  Color a rectangle with "exactly" the passed r, g, b color.

• void fl_rectf (Fl_Rect r)
Color with current color a rectangle that exactly fills the given bounding box.

- `void fl_rectf (Fl_Rect r, Fl_Color c)`
  - Color with passed color a rectangle that exactly fills the given bounding box.
  - `void fl_rectf (int x, int y, int w, int h)`
  - Color with current color a rectangle that exactly fills the given bounding box.
  - `void fl_rectf (int x, int y, int w, int h, Fl_Color c)`
  - Color with passed color a rectangle that exactly fills the given bounding box.
  - `void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)`
    - Color a rectangle with "exactly" the passed r, g, b color.
  - `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)`
    - Resets marked text.
  - `void fl_restore_clip ()`
    - Undo any clobbering of the clip region done by your program.
  - `FL_EXPORT void fl_restore_scale (float s)`
    - Restores the GUI scaling factor and the clipping region in subsequent drawing operations.
  - `void fl_rotate (double d)`
    - Concatenate rotation transformation onto the current one.
  - `void fl_rti_draw (const char *str, int n, int x, int y)`
    - Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.
  - `void fl_scale (double x)`
    - Concatenate scaling transformation onto the current one.
  - `void fl_scale (double x, double y)`
    - Concatenate scaling transformation onto the current one.
  - `FL_EXPORT void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void (draw_area)(void *, int, int, int), void *data)`
    - Scroll a rectangle and draw the newly exposed portions.
  - `FL_EXPORT void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)`
    - Inform text input methods about the current text insertion cursor.
  - `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_EXPORT void fl_text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h)`
    - Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().
  - `FL_EXPORT void fl_text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h)`
    - Determine the minimum pixel dimensions of a nul-terminated string using the current fl_font().
  - `double fl_transform_dx (double x, double y)`
    - Transform distance using current transformation matrix.
  - `double fl_transform_dy (double x, double y)`
    - Transform distance using current transformation matrix.
  - `double fl_transform_x (double x, double y)`
    - Transform coordinate using the current transformation matrix.
  - `double fl_transform_y (double x, double y)`
    - Transform coordinate using the current transformation matrix.
  - `void fl_transformed_vertex (double xf, double yf)`
    - Add coordinate pair to the vertex list without further transformations.
33.16 Fl_Image.H File Reference

- void fl_translate (double x, double y)
  
  Concatenate translation transformation onto the current one.
- void fl_vertex (double x, double y)
  
  Add a single vertex to the current path.
- FL_EXPORT double fl_width (const char *txt)
  
  Return the typographical width of a null-terminated string using the current font face and size.
- double fl_width (const char *txt, int n)
  
  Return the typographical width of a sequence of n characters using the current font face and size.
- double fl_width (unsigned int c)
  
  Return the typographical width of a single character using the current font face and size.
- void fl_xyline (int x, int y, int x1)
  
  Draw a horizontal line from (x,y) to (x1,y).
- void fl_xyline (int x, int y, int x1, int y2)
  
  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).
- void fl_xyline (int x, int y, int x1, int y2, int x3)
  
  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).
- void fl_yxline (int x, int y, int y1)
  
  Draw a vertical line from (x,y) to (x,y1).
- void fl_yxline (int x, int y, int y1, int x2)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).
- void fl_yxline (int x, int y, int y1, int x2, int y3)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

Variables

- FL_EXPORT char fl_draw_shortcut

33.15.1 Detailed Description

utility header to pull drawing functions together

33.16 Fl_Image.H File Reference

Fl_image, Fl_RGB_image classes.
#include "Enumerations.H"
#include "Fl_Widget.H"

Classes

- class Fl_image
  
  Base class for image caching, scaling and drawing.
- 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.
33.16.1 Detailed Description

\texttt{Fl\_Image, Fl\_RGB\_Image} classes.

33.16.2 Enumeration Type Documentation

33.16.2.1 \texttt{Fl\_RGB\_Scaling}

\begin{verbatim}
enum Fl_RGB_Scaling

The scaling algorithm to use for RGB images.

Enumerator
\begin{itemize}
\item FL_RGB_SCALING_NEAREST default RGB image scaling algorithm
\item FL_RGB_SCALING_BILINEAR more accurate, but slower RGB image scaling algorithm
\end{itemize}
\end{verbatim}

33.17 \texttt{Fl\_MenuItem.H} File Reference

\begin{verbatim}
#include "Fl\_Widget.H"
#include "Fl\_Image.H"
#include <FL/platform_types.h>
\end{verbatim}

Classes

- struct \texttt{Fl\_MenuItem}

  The \texttt{Fl\_MenuItem} structure defines a single menu item that is used by the \texttt{Fl\_Menu} class.

Typedefs

- typedef \texttt{Fl\_MenuItem Fl\_Menu}

Enumerations

- enum {

  FL_MENU_INACTIVE = 1 , FL_MENU_TOGGLE = 2 , FL_MENU_VALUE = 4 , FL_MENU_RADIO = 8 ,
  FL_MENU_INVISIBLE = 0x10 , FL_SUBMENU_POINTER = 0x20 , FL_SUBMENU = 0x40 , FL_MENU_DIVIDER
  = 0x80 ,
  FL_MENU_HORIZONTAL = 0x100 , FL_MENU_RESERVED = 0xffffff00 }

- enum {

  FL_PUP_NONE = 0 , FL_PUP_GREY = FL_MENU_INACTIVE , FL_PUP_GRAY = FL_MENU_INACTIVE ,
  FL_MENU_BOX = FL_MENU_TOGGLE ,
  FL_PUP_BOX = FL_MENU_TOGGLE , FL_PUP_CHECK = FL_MENU_VALUE , FL_PUP_RADIO = FL_MENU_RADIO ,
  FL_PUP_INVISIBLE = FL_MENU_INVISIBLE , FL_PUP_SUBMENU = FL_SUBMENU_POINTER }

Functions

- FL\_EXPORT \texttt{Fl\_Shortcut fl\_old\_shortcut (const char *)}

  Emulation of XForms named shortcuts.

33.17.1 Enumeration Type Documentation
33.18 Fl_Native_File_Chooser.H File Reference

**Fl_Native_File_Chooser** widget.

```c
#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>
```

### Detailed Description

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

33.19 Fl_Paged_Device.cxx File Reference

implementation of class **Fl_Paged_Device**.

```c
#include <FL/Fl_Paged_Device.H>
#include <FL/Fl.H>
#include <FL/fl_draw.H>
```

### Detailed Description

implementation of class **Fl_Paged_Device**.

33.20 Fl_Paged_Device.H File Reference

declaration of class **Fl_Paged_Device**.

```c
#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.

33.20.1 Detailed Description

description of class Fl_Paged_Device.

33.21 Fl_PostScript.H File Reference

declaration of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface.
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>

Classes

- class Fl_EPS_File_Surface
  Encapsulated PostScript drawing surface.

- class Fl_PostScript_File_Device
  To send graphical output to a PostScript file.

Typedefs

- typedef int(* Fl_PostScript_Close_Command)(FILE *)
  Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.

33.21.1 Detailed Description

description of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface.

33.21.2 Typedef Documentation

33.21.2.1 Fl_PostScript_Close_Command
typedef int(* Fl_PostScript_Close_Command)(FILE *)
Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.
A non-null return value indicates output error.

See also
  Fl_PostScript_File_Device::close_command() and Fl_EPS_File_Surface::Fl_EPS_File_Surface().
33.22 Fl_Printer.H File Reference

declaration of class Fl_Printer.
#include <FL/Fl_Paged_Device.H>

Classes

• class Fl_Printer
  
  OS-independent print support.

33.22.1 Detailed Description

declaration of class Fl_Printer.

33.23 fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.
#include <FL/platform.H>
#include <FL/Fl_Graphics_Driver.H>

33.23.1 Detailed Description

Drawing and clipping routines for rectangles.

33.24 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 *name, uchar *header, int headerlen)
  
  Test function (typedef) for adding new shared image formats.

Functions

• FL_EXPORT void fl_register_images ()
  
  Register the known image formats.

33.24.1 Detailed Description

Fl_Shared_Image class.

33.24.2 Typedef Documentation
33.24.2.1 Fl_Shared_Handler

typedef Fl_Image *(*Fl_Shared_Handler) (const char *name, uchar *header, int headerlen)

Test function (typedef) for adding new shared image formats.
This defines the function type you can use to add a handler for unknown image formats that can be opened and loaded as an Fl_Shared_Image.

`fl_register_images()` adds all image formats known to FLTK. Call Fl_Shared_Image::add_handler() to add your own check function to the list of known image formats.

Your function will be passed the filename (`name`), some header bytes already read from the image file and the size `headerlen` of the data read. The max value of size is implementation dependent. If your handler function needs to check more bytes you must open the image file yourself.

The provided buffer `header` must not be overwritten.
If your handler function can identify the file type you must open the file and return a valid Fl_Image or derived type, otherwise you must return NULL. Example:

```c
static Fl_Image *check_my_image(const char *name, uchar *header, int headerlen) {
    // (test image type using header and headerlen)
    if (known) {
        // (load image data from file `p name`)
        return new Fl_RGB_Image(data, ...);
    } else
        return 0;
}
```

// add your handler:
Fl_Shared_Image::add_handler(check_my_image);

Parameters

| in   | name       | filename to be checked and opened if applicable |
| in   | header     | portion of the file that has already been read |
| in   | headerlen  | length of provided header data                |

Returns

.valid Fl_Image or NULL.

See also

Fl_Shared_Image::add_handler()

33.24.3 Function Documentation

33.24.3.1 fl_register_images()

FL_EXPORT void fl_register_images ()

Register the known image formats.
This function is provided in the fltk_images library and registers all of the "extra" image file formats known to FLTK that are not part of the core FLTK library.
You may add your own image formats with Fl_Shared_Image::add_handler().

33.25 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.
33.25.1 Detailed Description
The `fl_show_colormap()` function hides the implementation classes used to provide the popup window and color selection mechanism.

33.26 Fl_String.cxx File Reference
Basic Fl_String class for FLTK.
```cpp
#include <FL/Fl_String.H>
#include <stdio.h>
#include <string.h>
```

33.26.1 Detailed Description
Basic Fl_String class for FLTK.

33.27 Fl_String.H File Reference
Basic Fl_String class for FLTK.

Classes
- class Fl_String
  Fl_String is the basic string class for FLTK.

33.27.1 Detailed Description
Basic Fl_String class for FLTK.

33.28 fl_string_functions.h File Reference
Public header for FLTK's own platform agnostic string handling.
```cpp
#include "Fl_Export.H"
#include "fl_types.h"
```

Functions
- FL_EXPORT char * `fl_strdup` (const char *s)
  Cross platform interface to POSIX function strdup().

33.28.1 Detailed Description
Public header for FLTK's own platform agnostic string handling.

33.29 Fl_Sys_Menu_Bar.H File Reference
Definition of class Fl_Sys_Menu_Bar.
```cpp
#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.

33.29.1 Detailed Description

Definition of class Fl_Sys_Menu_Bar.

33.30 Fl_Timeout.cxx File Reference

#include "Fl_Timeout.h"
#include "Fl_System_Driver.H"
#include <stdio.h>

Typedefs

- typedef struct FlTimeStamp FlTimeStamp_t

33.31 Fl_Timeout.h File Reference

Fl_Timeout handling.
#include <FL/Fl.H>

Classes

- class Fl_Timeout
  Class Fl_Timeout handles all timeout related functions.

Macros

- #define FL_TIMEOUT_DEBUG 0

33.31.1 Detailed Description

Fl_Timeout handling.
This file contains implementations of:

- Fl::add_timeout()
- Fl::repeat_timeout()
- Fl::remove_timeout()
- Fl::has_timeout()

and related methods of class Fl_Timeout.

33.32 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.

33.32.1 Detailed Description

This file contains the definitions of the Fl_Tree class.

33.32.2 Enumeration Type Documentation

33.32.2.1 Fl_Tree_Reason

enum Fl_Tree_Reason

The reason the callback was invoked.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_REASON_NONE</td>
<td>unknown reason</td>
</tr>
<tr>
<td>FL_TREE_REASON_SELECTED</td>
<td>an item was selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_DESELECTED</td>
<td>an item was de-selected</td>
</tr>
<tr>
<td>FL_TREE_REASON_RESELECTED</td>
<td>an item was re-selected (double-clicked). See Fl_Tree_Item_Reselect_Mode to 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>

33.33 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>
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item_Array.H>
#include <FL/Fl_Tree_Prefs.H>

Classes

- class Fl_Tree_Item

Generated by Doxygen
33.33 Tree widget item.

33.33.1 Detailed Description
This file contains the definitions for Fl_Tree_Item.

33.34 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

33.34.1 Detailed Description
This file defines a class that manages an array of Fl_Tree_Item pointers.

33.35 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 =2 }
  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.

33.35.1 Detailed Description

This file contains the definitions for Fl_Tree's preferences.

```
Fl_Tree_Prefs
;
    .........
    ;
Fl_Tree
    |
    | Fl_Tree_Item
```

33.35.2 Enumeration Type Documentation

33.35.2.1 Fl_Tree_Connector

`enum Fl_Tree_Connector`

Defines the style of connection lines between items.

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_CONNECTOR_NONE Use no lines connecting items.</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_DOTTED Use dotted lines connecting items (default)</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_SOLID Use solid lines connecting items.</td>
</tr>
</tbody>
</table>

33.35.2.2 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().

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_ITEM_DRAW_DEFAULT If widget() defined, draw in place of label, and widget() tracks item height (default)</td>
</tr>
<tr>
<td>FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET If widget() defined, include label to the left of the widget.</td>
</tr>
<tr>
<td>FL_TREE_ITEM_HEIGHT_FROM_WIDGET If widget() defined, widget()'s height controls item's height.</td>
</tr>
</tbody>
</table>

33.35.2.3 Fl_Tree_Item_Reselect_Mode

`enum Fl_Tree_Item_Reselect_Mode`

Defines the ways an item can be (re) selected via item_reselect_mode().

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECTABLE_ONCE Item can only be selected once (default)</td>
</tr>
<tr>
<td>FL_TREE_SELECTABLE_ALWAYS Enables FL_TREE_REASON_RESELECTED events for callbacks.</td>
</tr>
</tbody>
</table>
33.35.2.4 Fl_Tree_Select

enum Fl_Tree_Select
Tree selection style.

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECT_NONE</td>
<td>Nothing selected when items are clicked.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE</td>
<td>Single item selected when item is clicked (default).</td>
</tr>
<tr>
<td>FL_TREE_SELECT_MULTI</td>
<td>Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE_DRAAGGABLE</td>
<td>Single items may be selected, and they may be reordered by mouse drag.</td>
</tr>
</tbody>
</table>

33.35.2.5 Fl_Tree_Sort

enum Fl_Tree_Sort
Sort order options for items added to the tree.

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</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>

33.36 fl_types.h File Reference

This file contains simple "C"-style type definitions.

Typedefs

**Miscellaneous**

- typedef unsigned int Fl_Shortcut
  16-bit Unicode character + 8-bit indicator for keyboard flags.
- typedef unsigned char uchar
- typedef unsigned long ulong

33.36.1 Detailed Description

This file contains simple "C"-style type definitions.

33.36.2 Typedef Documentation

33.36.2.1 Fl_Shortcut

typedef unsigned int Fl_Shortcut
16-bit Unicode character + 8-bit indicator for keyboard flags.
Note
This should be 24-bit Unicode character + 8-bit indicator for keyboard flags. The upper 8 bits are currently unused but reserved.

Due to compatibility issues this type and all FLTK shortcuts can only be used with 16-bit Unicode characters (U+0000 .. U+FFFF) and not with the full range of unicode characters (U+0000 .. U+10FFFF). This is caused by the bit flags FL_SHIFT, FL_CTRL, FL_ALT, and FL_META being all in the range 0x010000 .. 0x400000.

Todo Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on all programs being coded with symbolic names and not hard coded bit values.

33.37 fl_utf8.h File Reference

header for Unicode and UTF-8 character handling
#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <sys/stat.h>

Functions

• FL_EXPORT int fl_access (const char ∗f, int mode)
  Cross-platform function to test a file's 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)

• FL_EXPORT FILE ∗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.
- 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 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_utf8froma (char ∗dst, unsigned dstlen, const char ∗src, unsigned srclen)
  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
- FL_EXPORT unsigned fl_utf8fromwc (char ∗dst, unsigned dstlen, const wchar_t ∗src, unsigned srclen)
  Convert "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_utf8to_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_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_utf8tocwc (const char ∗src, unsigned srclen, wchar_t ∗dst, unsigned dstlen)
  Converts a UTF-8 string into a wide character string.
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.

33.37.1 Detailed Description
header for Unicode and UTF-8 character handling

33.38 fl_vertex.cxx File Reference
Portable drawing code for drawing arbitrary shapes with simple 2D transformations.
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>

33.38.1 Detailed Description
Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

33.39 Fl_Widget.H File Reference
Fl_Widget, Fl_Label classes.
#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.

### 33.39.1 Detailed Description

*Fl_Widget, Fl_Label* classes.

### 33.39.2 Macro Definition Documentation

#### 33.39.2.1 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.

### 33.40 Fl_Window.H File Reference

*Fl_Window* widget.

```c
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Bitmap.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

### 33.40.1 Detailed Description

*Fl_Window* widget.

### 33.41 fl_write_png.cxx File Reference

PNG image support functions.

```c
#include <config.h>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_RGB_Image.H>
```
Functions

- int fl_write_png (const char *filename, const char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.
- int fl_write_png (const char *filename, const unsigned char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.
- int fl_write_png (const char *filename, Fl_RGB_Image *img)
  Write an RGB(A) image to a PNG image file.

33.41.1 Detailed Description

PNG image support functions.

33.41.2 Function Documentation

33.41.2.1 fl_write_png() [1/3]

int fl_write_png(
    const char * filename,
    const char * pixels,
    int w,
    int h,
    int d,
    int ld
)

Write raw image data to a PNG image file.
This is a very basic and restricted function to create a PNG image file from raw image data, e.g. a screenshot.
The image data must be aligned w/o gaps after each row (ld = 0 or ld = w * d) or ld must be the total length of each row, i.e. w * d + gapsize. If ld == 0 then ld = w * d is assumed.
The total data size must be (w * d + gapsize) * h = ld' * h where ld' = w * d if ld == 0.
For further restrictions and return values please see fl_write_png(const char * filename, Fl_RGB_Image *img).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in filename</td>
<td>Output filename, extension should be <code>.png</code></td>
</tr>
<tr>
<td>in pixels</td>
<td>Image data</td>
</tr>
<tr>
<td>in w</td>
<td>Image data width</td>
</tr>
<tr>
<td>in h</td>
<td>Image data height</td>
</tr>
<tr>
<td>in d</td>
<td>Image depth: 1 = GRAY, 2 = GRAY + alpha, 3 = RGB, 4 = RGBA</td>
</tr>
<tr>
<td>in ld</td>
<td>Line delta: default (0) = w * d</td>
</tr>
</tbody>
</table>

Returns

success (0) or error code, see ...

See also

fl_write_png(const char * filename, Fl_RGB_Image * img)
33.41.2.2  \texttt{fl\_write\_png()} [2/3]

\begin{verbatim}
int fl_write_png ( 
    const char * filename, 
    const unsigned char * pixels, 
    int w, 
    int h, 
    int d, 
    int ld )
\end{verbatim}

Write raw image data to a PNG image file.

See also

\begin{verbatim}
fl_write_png(const char *filename, const char *pixels, int w, int h, int d, int ld)
\end{verbatim}

33.41.2.3  \texttt{fl\_write\_png()} [3/3]

\begin{verbatim}
int fl_write_png ( 
    const char * filename, 
    Fl_RGB_Image * img )
\end{verbatim}

Write an RGB(A) image to a PNG image file.

This is a very basic and restricted function to create a PNG image file from an RGB image (\texttt{Fl\_RGB\_Image}).
The image data must be aligned w/o gaps, i.e. \texttt{ld()} \textbf{MUST} be zero or equal to data\_w() * data\_h().
The image file is always written with the original image size data\_w() and data\_h(), even if the image has been scaled.
Image depth 1 (gray), 2 (gray + alpha channel), 3 (RGB) and 4 (RGBA) are supported.

\begin{note}
Currenty there is no error handling except for errors when opening the file. This may be changed in the future.
\end{note}

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{filename} & Output filename, extension should be ‘.png’ \\
\hline
\textbf{in} & \textbf{img} & RGB image to be written \\
\hline
\end{tabular}

Returns

success (0) or error code: negative values are errors

Return values

\begin{tabular}{|c|c|}
\hline
\textbf{0} & success, file has been written \\
\hline
\textbf{-1} & png or zlib library not available \\
\hline
\textbf{-2} & file open error \\
\hline
\end{tabular}

See also

\begin{verbatim}
fl_write_png(const char *, int, int, int, const unsigned char *)
\end{verbatim}

33.42  \texttt{gl\_h File Reference}

This file defines wrapper functions for OpenGL in FLTK.

\#include "Enumerations.H"
\#include <GL/gl.h>
Functions

- FL_EXPORT void gl_color (Fl_Color i)
  
  Sets the current OpenGL color to an FLTK color.

- void gl_color (int c)
  
  back compatibility

- FL_EXPORT int gl_descent ()
  
  Returns the current font's descent.

- FL_EXPORT void gl_draw (const char * )
  
  Draws a null-terminated string in the current font at the current position.

- FL_EXPORT void gl_draw (const char *, float x, float y)
  
  Draws a null-terminated string in the current font at the given position.

- FL_EXPORT void gl_draw (const char *, int n)
  
  Draws an array of n characters of the string in the current font at the current position.

- FLEXPORT void gl_draw (const char *, int n, float x, float y)
  
  Draws n characters of the string in the current font at the given position.

- FL_EXPORT void gl_draw (const char *, int n, int x, int y)
  
  Draws n characters of the string in the current font at the given position.

- FL_EXPORT void gl_draw (const char *, int x, int y)
  
  Draws a null-terminated string in the current font at the given position.

- FL_EXPORT void gl_draw (const char *, int x, int y, int w, int h, Fl_Align)
  
  Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X.

- FL_EXPORT void gl_draw_image (const uchar *, int x, int y, int w, int h, int d=3, int ld=0)

- FL_EXPORT void gl_finish ()
  
  Releases an OpenGL context.

- FL_EXPORT void gl_font (int fontid, int size)
  
  Sets the current OpenGL font to the same font as calling fl_font().

- FL_EXPORT int gl_height ()
  
  Returns the current font's height.

- FL_EXPORT void gl_measure (const char *, int &x, int &y)
  
  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 int gl_texture_pile_height ()
  
  Returns the current maximum height of the pile of pre-computed string textures.

- FL_EXPORT void gl_texture_pile_height (int max)
  
  Changes the maximum height of the pile of pre-computed string textures.

- FL_EXPORT void gl_texture_reset ()
  
  To call after GL operations that may invalidate textures used to draw text in GL scenes (e.g., switch between FL_← DOUBLE / FL_SINGLE modes).

- 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.
33.42.1 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).

33.42.2 Function Documentation

33.42.2.1 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!

33.42.2.2 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)

33.42.2.3 gl_draw() [2/7]

FL_EXPORT void gl_draw ( const char * str, float x, float y )
Draws a nul-terminated string in the current font at the given position.
See also

   gl_texture_pile_height(int)

33.42.2.4 gl_draw() [3/7]

FL_EXPORT void gl_draw ( const char * str, int n )
Draws an array of n characters of the string in the current font at the current position.
See also

   gl_texture_pile_height(int)
33.42.2.5  `gl_draw()` [4/7]

```
FL_EXPORT void gl_draw {
    const char * str,
    int n,
    float x,
    float y
}
```

Draws n characters of the string in the current font at the given position.

See also
```
  gl_texture_pile_height(int)
```

33.42.2.6  `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)
```

33.42.2.7  `gl_draw()` [6/7]

```
FL_EXPORT void gl_draw {
    const char * str,
    int x,
    int y
}
```

Draws a null-terminated string in the current font at the given position.

See also
```
  gl_texture_pile_height(int)
```

33.42.2.8  `gl_draw()` [7/7]

```
FL_EXPORT void gl_draw {
    const char * str,
    int x,
    int y,
    int w,
    int h,
    Fl_Align align
}
```

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X, and aligned with the edges or center. Exactly the same output as `fl_draw()`.

33.42.2.9  `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)

33.42.10  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.

33.42.11  gl_rectf()

void gl_rectf {
    int x,
    int y,
    int w,
    int h ) [inline]

Fills the given rectangle with the current color.
See also

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

33.42.12  gl_texture_pile_height() [1/2]

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

Fl::draw_GL_text_with_textures(int)

33.42.13  gl_texture_pile_height() [2/2]

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

Fl::draw_GL_text_with_textures(int)

33.43  mac.H File Reference

Mac OS X-specific symbols.
Classes

• class Fl_Mac_App_Menu

Functions

• FL_Window * fl_mac_find (FLWindow *)
  Returns the FL_Window corresponding to the given macOS-specific window reference.
• CGContextRef fl_mac_gc ()
  Returns the macOS-specific graphics context for the current window.
• void fl_mac_set_about (FL_Callback *cb, void *user_data, int shortcut=0)
  Attaches a callback to the "About myprog" item of the system application menu.
• FLWindow * fl_mac_xid (const Fl_Window *)
  Returns the macOS-specific window reference corresponding to the given Fl_Window object.

Variables

• int fl_mac_os_version
  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13).

33.43.1 Detailed Description
Mac OS X-specific symbols.

33.44 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).

33.44.1 Function Documentation

33.44.1.1 fl_casenumericsort()

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()
33.44.1.2 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**

| -1  | A < B |
| 0   | A == B |
| +1  | A > B |

**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()`
33.45 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.

Typedefs

- `typedef opaque fl_intptr_t
  An integral type large enough to store a pointer or a long value.
- `typedef opaque FL_Offscreen
  Platform-specific value representing an offscreen drawing buffer.
- `typedef struct opaque * FL_Region
  Pointer to a platform-specific structure representing a collection of rectangles.
- `typedef opaque FL_SOCKET
  socket or file descriptor
- `typedef opaque fl_uintptr_t
  An unsigned integral type large enough to store a pointer or an unsigned long value.
- `typedef struct opaque * GLContext
  Pointer to a platform-specific structure representing the window's OpenGL rendering context.

33.45.1 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.

33.45.2 Typedef Documentation

33.45.2.1 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.

33.45.2.2 FL_Offscreen

typedef opaque FL_Offscreen
Platform-specific value representing an offscreen drawing buffer.

Note
This value can be safely cast to these types on each platform:

- X11: Pixmap
- Wayland: struct fl_wld_buffer *
- Windows: HBITMAP
- macOS: CGContextRef
### 33.45.2.3 Fl_Region

typedef struct opaque* Fl_Region

Pointer to a platform-specific structure representing a collection of rectangles.

Note

This pointer can be safely cast to these types on each platform:

- X11: Region as defined by X11
- Wayland: struct flCairoRegion *
- Windows: HRGN
- macOS: struct flCocoaRegion *

### 33.45.2.4 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.

### 33.45.2.5 GLContext

typedef struct opaque* GLContext

Pointer to a platform-specific structure representing the window's OpenGL rendering context.

Note

This pointer can be safely cast to these types on each platform:

- X11: GLXContext
- Wayland: EGLContext
- Windows: HGLRC
- macOS: NSOpenGLContext *

### 33.46 vsnprintf.c File Reference

Portable vsnprintf() implementation.

```c
#include <stdio.h>
#include "flstring.h"
```

**Functions**

- `int fl_snprintf (char *str, size_t size, const char *fmt,...)`
- `int fl_vsnprintf (char *buffer, size_t bufsize, const char *format, va_list ap)`

FLTK's platform independent wrapper for the vsnprintf() C library function.

### 33.46.1 Detailed Description

Portable vsnprintf() implementation.

### 33.46.2 Function Documentation
33.46.2.1 fl_vsnprintf()

```c
int fl_vsnprintf (char * buffer, size_t bufsize, const char * format, va_list ap)
```

FLTKit's platform independent wrapper for the vsnprintf() C library function. This function guarantees:

- access to vsnprintf(), even on systems that don't have it (FLTKit's own built-in code is used)
- Guarantees NUL termination. Even if string expands larger than the buffer, a terminating NUL is included, unlike some implementations of vsnprintf(), notably Microsoft Visual Studio (pre-2015), which can leave the string unterminated when truncated.

If the build environment for FLTKit has vsnprintf(), fl_vsnprintf() is just a wrapper around the compiler's provided function. Otherwise, if the function is NOT available, FLTKit's own built-in version is provided. 

The FLTKit built in provides these style options:

- %[ -#+] 
- * – padding width
- .* – precision width
- Data types: h, l, ll, L
- Floating point formats: E, G, e, f, g
- Integer formats: B, X, b, d, i, o, u, x
- Pointer format: p
- String/char: c, s, n

33.47 wayland.H File Reference

Definitions of functions specific to the Wayland platform.

**Typedefs**

- typedef struct _cairo cairo_t
- typedef void * EGLContext

**Functions**

- FL_EXPORT cairo_t * fl_wl_cairo ()
  - Returns the cairo context associated to the current window.
- FL_EXPORT struct wl_display * fl_wl_display ()
  - Returns the Wayland display in use.
- FL_EXPORT Fl_Window * fl_wl_find (struct wld_window *)
  - Returns the Fl_Window corresponding to a given the platform-specific window reference.
- FL_EXPORT EGLContext fl_wl_glcontext (GLContext rc)
  - Returns the EGLContext corresponding to the given GLContext.
- FL_EXPORT struct wl_surface * fl_wl_surface (struct wld_window *xid)
  - Returns the wl_surface associated to a shown window.
- FL_EXPORT struct wld_window * fl_wl_xid (const Fl_Window *win)
  - Returns a platform-specific reference associated to a shown window.
33.47.1 Detailed Description
Definitions of functions specific to the Wayland platform.

33.48 win32.H File Reference
Definitions of functions specific to the Windows platform.

Functions
- HINSTANCE fl_win32_display ()
  Returns the Windows-specific display in use
- Fl_Window * fl_win32_find (HWND)
  Returns the Fl_Window corresponding to the given Windows-specific window reference.
- HDC fl_win32_gcontext ()
  Returns the Windows-specific graphics context for the current window
- HGLRC fl_win32_gcontext (GLContext rc)
  Returns the Windows-specific GL rendering context corresponding to the given GLContext
- HWND fl_win32_xid (const Fl_Window * win)
  Returns the Windows-specific window reference corresponding to the given Fl_Window object.

33.48.1 Detailed Description
Definitions of functions specific to the Windows platform.

33.49 x11.H File Reference
Definitions of functions specific to the X11 platform.

Functions
- Display * fl_x11_display ()
  Returns the X11 Display in use.
- Fl_Window * fl_x11_find (Window xid)
  Returns the Fl_Window corresponding to the given Window reference.
- GC fl_x11_gcontext ()
  Returns the X11-specific currently active graphics context.
- Window fl_x11_xid (const Fl_Window * win)
  Returns the Window reference for the given Fl_Window, or zero if not shown ()

33.49.1 Detailed Description
Definitions of functions specific to the X11 platform.

33.49.2 Function Documentation
33.49.2.1  fl_x11_find()

```c
Fl_Window* fl_x11_find ( 
    Window xid )
```

Returns the `Fl_Window` corresponding to the given Window reference.

33.49.2.2  fl_x11_gc()

```c
GC fl_x11_gc ( )
```

Returns the X11-specific currently active graphics context.

33.49.2.3  fl_x11_xid()

```c
Window fl_x11_xid ( 
    const Fl_Window * win )
```

Returns the Window reference for the given `Fl_Window`, or zero if not `shown()`.
Index

_FL_DIAMOND_DOWN_BOX
Enumerations.H, 1063

_FL_DIAMOND_UP_BOX
Enumerations.H, 1069

_FL_ENGRAVED_LABEL
Enumerations.H, 1069

_FL_EMBOSSED_LABEL
Enumerations.H, 1069

_FL_GLEAM_DOWN_BOX
Enumerations.H, 1064

_FL_GLEAM_DOWN_FRAME
Enumerations.H, 1064

_FL_GLEAM_ROUND_DOWN_BOX
Enumerations.H, 1064

_FL_GLEAM_ROUND_UP_BOX
Enumerations.H, 1064

_FL_GLEAM_THIN_DOWN_BOX
Enumerations.H, 1064

_FL_GLEAM_THIN_UP_BOX
Enumerations.H, 1064

_FL_GTK_DOWN_BOX
Enumerations.H, 1064

_FL_GTK_DOWN_FRAME
Enumerations.H, 1064

_FL_GTK_ROUND_DOWN_BOX
Enumerations.H, 1064

_FL_GTK_ROUND_UP_BOX
Enumerations.H, 1064

_FL_GTK_THIN_DOWN_BOX
Enumerations.H, 1064

_FL_GTK_THIN_DOWN_FRAME
Enumerations.H, 1064

_FL_GTK_THIN_UP_BOX
Enumerations.H, 1064

_FL_GTK_THIN_UP_FRAME
Enumerations.H, 1064

_FL_ICON_LABEL
Enumerations.H, 1069

_FL_IMAGE_LABEL
Enumerations.H, 1069

_FL_MULTI_LABEL

Enumerations.H, 1064

_FL_OFLAT_BOX
Enumerations.H, 1063

_FL_OSHADOW_BOX
Enumerations.H, 1063

_FL_OVAL_BOX
Enumerations.H, 1063

_FL_OVAL_FRAME
Enumerations.H, 1063

_FL_PLASTIC_DOWN_BOX
Enumerations.H, 1063

_FL_PLASTIC_DOWN_FRAME
Enumerations.H, 1063

_FL_PLASTIC_ROUND_DOWN_BOX
Enumerations.H, 1064

_FL_PLASTIC_ROUND_UP_BOX
Enumerations.H, 1064

_FL_PLASTIC_THIN_DOWN_BOX
Enumerations.H, 1063

_FL_PLASTIC_THIN_UP_BOX
Enumerations.H, 1063

_FL_PLASTIC_UP_BOX
Enumerations.H, 1063

_FL_PLASTIC_UP_FRAME
Enumerations.H, 1063

_FL_RFLAT_BOX
Enumerations.H, 1063

_FL_RGBA
Enumerations.H, 1063

_FL_RGBAextracomment
Enumerations.H, 1063

_FL_RGBA俐
Enumerations.H, 1063

_FL_RGBA俐
Enumerations.H, 1063

_FL_RGBA俐
Enumerations.H, 1063

_FL_RSHADOW_BOX
Enumerations.H, 1063

_FL_SHADOW_BOX
Enumerations.H, 1063

_FL_SHADOW_FRAME
Enumerations.H, 1063

_FL_SHADOW_LABEL
Enumerations.H, 1069

remove
Fl_Browser, 389

~Fl_Double_Window
Fl_Double_Window, 467

~Fl_EPS_File_Surface
Fl_EPS_File_Surface, 470
INDEX 1121

FL_Input, 568
FL_Simple_Terminal, 738
FL_Text_Buffer, 806

appendfile
FL_Text_Buffer, 806

arg
FL, 363

args
FL, 364

argument
FL_Menu_Item, 621
FL_Widget, 987
array
FL_Group, 524
FL_RGB_Image, 709
FL_Table, 780
as_gl_window
FL_Gl_Window, 512
FL_Widget, 987
as_group
FL_Group, 524
FL_Widget, 987
as_shared_image
FL_Image, 550
FL_Shared_Image, 729
as_svg_image
FL_RGB_Image, 707
FL_SVG_Image, 763
as_window
FL_Widget, 988
FL_Window, 1025

atclose
Windows handling functions, 242

ATTR_BGCOLOR
FL_Text_Display, 826

ATTR_BGCOLOR_EXT
FL_Text_Display, 827

ATTR_BGCOLOR_EXT_COMPLEX
FL_Text_Display, 827

ATTR_GRAMMAR
FL_Text_Display, 827

ATTR_LINES_MASK
FL_Text_Display, 827

ATTR_SPELLING
FL_Text_Display, 827

ATTR_STRIKE_THROUGH
FL_Text_Display, 827

ATTR_UNDERLINE
FL_Text_Display, 827

autosize
FL_Chart, 431, 432

awake
Multithreading support functions, 303, 304

b
FL_Color_Chooser, 453
FL_Rect, 700
B0
FL_Paged_Device, 648

B1
FL_Paged_Device, 648

B10
FL_Paged_Device, 648

B2
FL_Paged_Device, 648

B3
FL_Paged_Device, 648

B4
FL_Paged_Device, 648

B5
FL_Paged_Device, 648

B6
FL_Paged_Device, 648

B7
FL_Paged_Device, 648

B8
FL_Paged_Device, 648

B9
FL_Paged_Device, 648

background
FL, 364

background2
FL, 365

bbox
FL_Browser, 410
FL_Scroll, 716

begin
FL_Group, 525

begin_job
FL_Paged_Device, 649
FL_PostScript_File_Device, 664, 665
FL_Printer, 692

begin_page
FL_Paged_Device, 649
FL_PostScript_File_Device, 664, 665
FL_Printer, 692

belowmouse
Events handling functions, 246

BLOCK_CURSOR
FL_Text_Display, 826

border
FL_Window, 1025

BOTH
FL_Browser, 409

BOTH_ALWAYS
FL_Browser, 409

bottomline
FL_Browser, 389

bounds
FL_Chart, 432
FL_Group, 525
FL_Slider, 747

box
FL_Widget, 988

box_border_radius_max
FL, 365

box_color

Generated by Doxygen
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear_modal_states</td>
<td>1026</td>
</tr>
<tr>
<td>clear_output</td>
<td>991</td>
</tr>
<tr>
<td>clear_rect</td>
<td>830</td>
</tr>
<tr>
<td>clear_submenu</td>
<td>602</td>
</tr>
<tr>
<td>clear_visible</td>
<td>769</td>
</tr>
<tr>
<td>clear_visible_focus</td>
<td>991</td>
</tr>
<tr>
<td>clear_widget_pointer</td>
<td>991</td>
</tr>
<tr>
<td>client_area</td>
<td>789</td>
</tr>
<tr>
<td>CLIP_CHILDREN</td>
<td>526</td>
</tr>
<tr>
<td>clipboard_contains</td>
<td>257</td>
</tr>
<tr>
<td>close</td>
<td>470</td>
</tr>
<tr>
<td>close_dialog</td>
<td>759</td>
</tr>
<tr>
<td>closedeicon</td>
<td>903</td>
</tr>
<tr>
<td>closeicon</td>
<td>959</td>
</tr>
<tr>
<td>closeicon</td>
<td>904</td>
</tr>
<tr>
<td>closeicon</td>
<td>959</td>
</tr>
<tr>
<td>col_header</td>
<td>782</td>
</tr>
<tr>
<td>col_resize</td>
<td>782</td>
</tr>
<tr>
<td>col_resize_min</td>
<td>782</td>
</tr>
<tr>
<td>col_to_x</td>
<td>830</td>
</tr>
<tr>
<td>col_width</td>
<td>782</td>
</tr>
<tr>
<td>col_width_all</td>
<td>783</td>
</tr>
<tr>
<td>color</td>
<td>886</td>
</tr>
<tr>
<td>Color &amp; Font functions</td>
<td>992</td>
</tr>
<tr>
<td>fl_color</td>
<td>266</td>
</tr>
<tr>
<td>fl_color_average</td>
<td>267</td>
</tr>
<tr>
<td>fl_contrast</td>
<td>267</td>
</tr>
<tr>
<td>fl_font</td>
<td>267</td>
</tr>
<tr>
<td>fl_height</td>
<td>268</td>
</tr>
<tr>
<td>fl_latin1_to_local</td>
<td>268</td>
</tr>
<tr>
<td>fl_local_to_latin1</td>
<td>268</td>
</tr>
<tr>
<td>fl_local_to_mac_roman</td>
<td>269</td>
</tr>
<tr>
<td>fl_mac_roman_to_local</td>
<td>269</td>
</tr>
<tr>
<td>fl_show_colormap</td>
<td>269</td>
</tr>
<tr>
<td>fl_size</td>
<td>270</td>
</tr>
<tr>
<td>fl_text_extents</td>
<td>270, 271</td>
</tr>
<tr>
<td>fl_width</td>
<td>271</td>
</tr>
<tr>
<td>free_color</td>
<td>271</td>
</tr>
<tr>
<td>get_color</td>
<td>271, 272</td>
</tr>
<tr>
<td>get_font</td>
<td>272</td>
</tr>
<tr>
<td>get_font_name</td>
<td>272</td>
</tr>
<tr>
<td>get_font_sizes</td>
<td>272</td>
</tr>
<tr>
<td>set_color</td>
<td>272, 273</td>
</tr>
<tr>
<td>set_font</td>
<td>273</td>
</tr>
<tr>
<td>set_fonts</td>
<td>274</td>
</tr>
<tr>
<td>color2</td>
<td>992, 993</td>
</tr>
<tr>
<td>Color &amp; Font functions</td>
<td>264</td>
</tr>
<tr>
<td>color_average</td>
<td>550</td>
</tr>
<tr>
<td>Fl_Image</td>
<td>653</td>
</tr>
<tr>
<td>Fl_RGB_Image</td>
<td>707</td>
</tr>
<tr>
<td>Fl_Shared_Image</td>
<td>729</td>
</tr>
<tr>
<td>Fl_SVG_Image</td>
<td>763</td>
</tr>
<tr>
<td>Fl_Tiled_Image</td>
<td>873</td>
</tr>
<tr>
<td>COLUMN</td>
<td>493</td>
</tr>
<tr>
<td>column_char</td>
<td>390</td>
</tr>
<tr>
<td>column_width</td>
<td>391</td>
</tr>
<tr>
<td>Common Dialog Classes and Functions</td>
<td>326</td>
</tr>
<tr>
<td>error</td>
<td>342</td>
</tr>
<tr>
<td>fatal</td>
<td>342</td>
</tr>
<tr>
<td>fl_alert</td>
<td>328</td>
</tr>
<tr>
<td>fl_ask</td>
<td>329</td>
</tr>
<tr>
<td>fl_bEEP</td>
<td>329</td>
</tr>
<tr>
<td>fl_choice</td>
<td>329</td>
</tr>
<tr>
<td>fl_choice_n</td>
<td>330</td>
</tr>
<tr>
<td>fl_color_chooser</td>
<td>331, 332</td>
</tr>
<tr>
<td>fl_dir_chooser</td>
<td>333</td>
</tr>
<tr>
<td>fl_file_chooser</td>
<td>334</td>
</tr>
<tr>
<td>fl_file_chooser_callback</td>
<td>334</td>
</tr>
<tr>
<td>fl_file_chooser_ok_label</td>
<td>335</td>
</tr>
<tr>
<td>fl_input</td>
<td>335</td>
</tr>
<tr>
<td>fl_input_str</td>
<td>335</td>
</tr>
<tr>
<td>fl_message</td>
<td>336</td>
</tr>
<tr>
<td>fl_message_hotspot</td>
<td>336, 337</td>
</tr>
<tr>
<td>fl_message_icon</td>
<td>337</td>
</tr>
<tr>
<td>fl_message_icon_label</td>
<td>337</td>
</tr>
<tr>
<td>fl_message_position</td>
<td>338, 340</td>
</tr>
<tr>
<td>fl_message_title</td>
<td>340</td>
</tr>
<tr>
<td>fl_message_title_default</td>
<td>341</td>
</tr>
<tr>
<td>fl_password</td>
<td>341</td>
</tr>
<tr>
<td>fl_password_str</td>
<td>341</td>
</tr>
<tr>
<td>warning</td>
<td>343</td>
</tr>
<tr>
<td>compare</td>
<td>730</td>
</tr>
<tr>
<td>compose</td>
<td>246</td>
</tr>
<tr>
<td>Events handling functions</td>
<td>246</td>
</tr>
<tr>
<td>compose_reset</td>
<td>246</td>
</tr>
</tbody>
</table>
connectorstyle
  Fl_Tree, 904
contains
  Fl_Widget, 993
context
  Fl_Gl_Window, 512, 513
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
  Fl_Table, 780
context_valid
  Fl_Gl_Window, 513
COPELED_LABEL
  Fl_Widget, 984
COPELED_TOOLTIP
  Fl_Widget, 984
copy
  Fl_Bitmap, 380
  Fl_Image, 551
  Fl_Input_, 569
  Fl_Menu_, 603
  Fl_Pixmap, 653
  Fl_RGB_Image, 707
  Fl_Shared_IMAGE, 730
  Fl_SVG_IMAGE, 763
  Fl_Text_Buffer, 807
  Fl_Tiled_IMAGE, 874
  Selection & Clipboard functions, 257
copy_cuts
  Fl_Input_, 569
copy_label
  Fl_Widget, 993
copy_tooltip
  Fl_Widget, 993
CORE
  Fl_Preferences, 674
CORE_READ_OK
  Fl_Preferences, 689
CORE_SYSTEM
  Fl_Preferences, 674
CORE_SYSTEM_L
  Fl_Preferences, 674
CORE_USER
  Fl_Preferences, 674
CORE_USER_L
  Fl_Preferences, 674
  CORE_WRITE_OK
  Fl_Preferences, 689
count
  Fl_Image, 552
  Fl_Native_File_Chooser, 637
count_displayed_characters
  Fl_Text_Buffer, 808
count_lines
  Fl_Text_Buffer, 808
  Fl_Text_Display, 830
create_window_menu
  Fl_Sys_Menu_Bar, 769
current
  Fl_Group, 526
  Fl_Timeout, 876
  Fl_Tooltip, 886
  Fl_Window, 1026
current_
  Fl_Window, 1040
current_style_index
  Fl_Simple_Terminal, 739
current_timeout
  Fl_Timeout, 879
cursor
  Fl_Window, 1026, 1027
cursor2rowcol
  Fl_Table, 783
cursor_color
  Fl_Input_, 569, 570
  Fl_Price_Display, 831
  Fl_Value_Input, 969
cursor_style
  Fl_Price_Display, 831
custom_application_menu_items
  Fl_Mac_App_Menu, 596
cut
  Fl_Input_, 570
d
  Fl_Image, 552
damage
  Fl_Widget, 994
damage_zone
  Fl_Table, 783
data
  Fl_Browser, 391
  Fl_Image, 552
deactivate
  Fl_Menu_Item, 622
  Fl_Tree_Item, 939
  Fl_Widget, 995
decorated_h
  Fl_Window, 1027
decorated_w
  Fl_Window, 1027
default_atclose
  Windows handling functions, 241
default_callback
  Fl_Widget, 995
default_cursor
  Fl_Window, 1027
default_icon
  Fl_Window, 1028
default_icons
  Fl_Window, 1028
default_size_range
  Fl_Window, 1029
default_xclass
  Fl_Window, 1030
deimage
  Fl_Widget, 995, 996
delay
  Fl_Tooltip, 886
delete_child
  Fl_Group, 526
  Fl_Scroll, 717
delete_entry
  Fl_Preferences, 677
delete_group
  Fl_Preferences, 678
delete_widget
  Safe widget deletion support functions, 306
deleted
  Fl_Widget_Tracker, 1019
deleting
  Fl_Browser_, 410
deparent
  Fl_Tree_Item, 939
  Fl_Tree_Item_Array, 955
depth
  Fl_Tree_Item, 939
desaturate
  Fl_Image, 553
  Fl_Pixmap, 654
  Fl_RGB_Image, 708
  Fl_Shared_Image, 731
  Fl_SVG_Image, 764
  Fl_Tiled_Image, 874
deselect
  Fl_Browser_, 410
  Fl_Tree, 904, 905
deselect_all
  Fl_Tree, 905
  Fl_Tree_Item, 939
DIM_CURSOR
  Fl_Text_Display, 826
direction
  Fl_Timer, 881
directory
  Fl_Native_File_Chooser, 637
dirty
  Fl_Preferences, 678
disable
  Fl_Tooltip, 886
disable_im
  Events handling functions, 247
display
  Fl, 367
  Fl_Browser, 391
  Fl_Browser_, 411
  Fl_Tree, 906
display_insert
  Fl_Text_Display, 831
displayed
  Fl_Browser, 392
  Fl_Browser_, 411
  Fl_Tree, 906
dnd
  Selection & Clipboard functions, 258
dnd_text_ops
  Fl, 367
do_callback
  Fl_Menu_Item, 622, 623
  Fl_Table, 783
  Fl_Widget, 996, 997
do_widget_deletion
  Safe widget deletion support functions, 306
down_box
  Fl_Button, 424
  Fl_File_Input, 488
  Fl_Menu_, 603
draw
  Fl_Adjuster, 378
  Fl_Bitmap, 380
  Fl_Box, 384
  Fl_Button, 425
  Fl_Chart, 432
  Fl_Choice, 443
  Fl_Clock_Output, 448
  Fl_Counter, 461
  Fl_Dial, 465
  Fl_File_Icon, 483
  Fl_FormsPixmap, 503
  Fl_FormsText, 504
  Fl_Free, 506
  Fl_Gl_Window, 513
  Fl_Glut_Window, 520
  Fl_Group, 527
  Fl_Image, 553
  Fl_Input, 564
  Fl_Label, 592
  Fl_Light_Button, 594
  Fl_Menu_Bar, 612
  Fl_Menu_Button, 615
  Fl_Pack, 646
  Fl_Pixmap, 654
  Fl_Positioner, 661
  Fl_Return_Button, 704
  Fl_RGB_Image, 708
  Fl_Roller, 711
  Fl_Scroll, 717
  Fl_Scrollbar, 722
  Fl_Shared_Image, 731
  Fl_Simple_Terminal, 739
  Fl_Slider, 747
INDEX

Unicode and UTF-8 functions, 311

event
  Events handling functions, 247

event_button
  Events handling functions, 247

event_button1
  Events handling functions, 247

event_button2
  Events handling functions, 247

event_button3
  Events handling functions, 247

event_buttons
  Events handling functions, 248

event_clicks
  Events handling functions, 248

event_clipboard
  Events handling functions, 248

event_clipboard_type
  Events handling functions, 248

event_dispatch
  Events handling functions, 249

event_dx
  Events handling functions, 249

event_dy
  Events handling functions, 249

event_inside
  Events handling functions, 249

event_key
  Events handling functions, 250

event_length
  Events handling functions, 251

event_original_key
  Events handling functions, 251

event_state
  Events handling functions, 251

event_text
  Events handling functions, 252

event_x_root
  Events handling functions, 252

event_y_root
  Events handling functions, 252

executive
  Fl_Paged_Device, 648

exists
  Fl_Widget_Tracker, 1019

extend_range_for_styles
  Fl_Text_Display, 834

extend_selection
  Fl_Tree, 906

extend_selection_dir
  Fl_Tree, 907

fail
  Fl_Image, 554

fatal
  Common Dialog Classes and Functions, 342

File names and URI utility functions, 343

fl_decode_uri, 344

fl_filename_absolute, 344

fl_filename_clean, 345

fl_filename_free, 345

fl_filename_match, 346

fl_filename_name, 346

fl_filename_name, 348

fl_filename_relative, 348

fl_filename_setext, 349

fl_open_uri, 349

file_access
  Fl_Preferences, 679

file_encoding_warning_message
  Fl_Text_Buffer, 816

filename
visual, 375
wait, 375
Fl.cxx, 1072
  fl_disable_wayland, 1073
  fl_find, 1073
  fl_open_display, 1074
Fl.H, 1074
  fl_disable_wayland, 1075
FL_ABI_VERSION
  Enumerations.H, 1060
fl_access
  Unicode and UTF-8 functions, 312
FL_ACTIVATE
  Enumerations.H, 1067
fl_add_symbol
  Drawing functions, 279
Fl_Adjuster, 377
  draw, 378
  Fl_Adjuster, 378
  handle, 378
  soft, 378, 379
fl_alert
  Common Dialog Classes and Functions, 328
FL_ALIGN_LEFT
  Enumerations.H, 1071
FL_ALIGN_TOP
  Enumerations.H, 1071
fl_antialias
  Drawing functions, 279
FL_API_VERSION
  Enumerations.H, 1060
fl_arc
  Drawing functions, 279, 280
fl_arc.cxx, 1076
fl_ask
  Common Dialog Classes and Functions, 329
fl_ask.cxx, 1076
fl_ask.H, 1078
  FL_Beep, 1079
  FL_BEEP_DEFAULT, 1079
  FL_BEEP_ERROR, 1079
  FL_BEEP_MESSAGE, 1079
  FL_BEEP_NOTIFICATION, 1079
  FL_BEEP_PASSWORD, 1079
  FL_BEEP_QUESTION, 1079
  fl_message_position, 1079
Fl_Beep
  fl_ask.H, 1079
fl_beep
  Common Dialog Classes and Functions, 329
FL_BEEP_DEFAULT
  fl_ask.H, 1079
FL_BEEP_ERROR
  fl_ask.H, 1079
FL_BEEP_MESSAGE
  fl_ask.H, 1079
FL_BEEP_NOTIFICATION
  fl_ask.H, 1079
FL_BEEP_PASSWORD
  fl_ask.H, 1079
FL_BEEP_QUESTION
  fl_ask.H, 1079
  fl_begin_complex_polygon
    Drawing functions, 281
  fl_begin_offscreen
    Drawing functions, 281
  fl_begin_points
    Drawing functions, 281
Fl_Bitmap, 379
  copy, 380
  draw, 380
  label, 381
  uncache, 381
Fl_BMP_Image, 381
  Fl_BMP_Image, 382
FL_BORDER_BOX
  Enumerations.H, 1063
FL_BORDER_FRAME
  Enumerations.H, 1063
Fl_Box, 383
  draw, 384
  Fl_Box, 384
  handle, 384
fl_box
  Enumerations.H, 1070
Fl_Boxtype
  Enumerations.H, 1062
fl_boxtype.cxx, 1080
  fl_internal_boxtype, 1081
  fl_rectbound, 1081
Fl_Browser, 385
  _remove, 389
  add, 389
  bottomline, 389
  clear, 390
  column_char, 390
  column_widths, 390, 391
  data, 391
  display, 391
  displayed, 392
  find_line, 392
  Fl_Browser, 388
  format_char, 392, 393
  full_height, 393
  hide, 393
  icon, 394
  incr_height, 394
  insert, 395
  item_at, 395
  item_draw, 396
  item_first, 396
  item_height, 396
  item_last, 397
  item_next, 397
  item_prev, 397
  item_select, 398

Generated by Doxygen
item_selected, 398
item_swap, 398
item_text, 399
item_width, 399
lineno, 399
lineposition, 400
load, 400
make_visible, 400
middeline, 401
move, 401
remove, 401
remove_icon, 402
select, 402
selected, 402
show, 402
size, 403
swap, 403
text, 403, 404
textsize, 404
topleft, 404
value, 405
visible, 405
Fl_Browser_, 406
ALWAYS_ON, 409
bbox, 410
BOTH, 409
BOTH_ALWAYS, 409
deleting, 410
deselect, 410
display, 411
displayed, 411
find_item, 411
FI_Browser_, 409
full_height, 412
full_width, 412
handle, 412
has_scrollbar, 412
HORIZONTAL, 409
HORIZONTAL_ALWAYS, 409
hposition, 413
hscrollbar, 421
incr_height, 413
inserting, 413
item_at, 414
item_first, 414
item_height, 414
item_last, 414
item_next, 415
item_prev, 415
item_quick_height, 415
item_select, 415
item_selected, 416
item_swap, 416
item_text, 416
item_width, 416
leftedge, 417
new_list, 417
position, 417
redraw_line, 418
redraw_lines, 418
replacing, 418
resize, 418
scrollbar, 421
scrollbar_left, 419
scrollbar_right, 419
scrollbar_size, 419
scrollbar_width, 419, 420
select, 420
select_only, 420
selection, 420
sort, 421
swapping, 421
textfont, 421
VERTICAL, 409
VERTICAL_ALWAYS, 409
Fl_Button, 422
clear, 424
down_box, 424
draw, 425
Fl_Button, 423
handle, 425
set, 425
shortcut, 425, 426
value, 426
Fl_Cairo_State, 426
cc, 427
Fl_Cairo_Window, 427
set_draw_cb, 429
fl_can_do_alpha_blending
Drawing functions, 282
FL_CAP_FLAT
Drawing functions, 279
FL_CAP_ROUND
Drawing functions, 279
FL_CAP_SQUARE
Drawing functions, 279
fl_capture_window
Drawing functions, 282
fl_casenumericsort
numericsort.c, 1111
Fl_Chart, 429
add, 431
autosize, 431, 432
bounds, 432
draw, 432
Fl_Chart, 431
insert, 433
maxsize, 433
replace, 433
FL_CHART_ENTRY, 434
fl_chdir
Unicode and UTF-8 functions, 312
Fl_Check_Browser, 434
add, 436
handle, 436
item_at, 436
item_first, 437
item_height, 437
item_next, 437
item_prev, 437
item_select, 437
item_selected, 438
item_swap, 438
item_text, 438
item_width, 439
nitems, 439
remove, 439
Fl_Check_Button, 439
   Fl_Check_Button, 440
fl_chmod
   Unicode and UTF-8 functions, 313
Fl_Choice, 440
draw, 443
   Fl_Choice, 442
   handle, 443
   value, 443, 444
fl_choice
   Common Dialog Classes and Functions, 329
fl_choice_n
   Common Dialog Classes and Functions, 330
fl_circle
   Drawing functions, 282
fl_clip
   Drawing functions, 283
fl_clip_box
   Drawing functions, 283
fl_clip_region
   Drawing functions, 284
Fl_Clock, 444
   Fl_Clock, 445, 446
   handle, 446
Fl_Clock_Output, 447
draw, 448
   Fl_Clock_Output, 448
   hour, 449
   minute, 449
   second, 449
   shadow, 449, 450
   value, 450
FL_CLOSE
   Enumerations.H, 1067
fl_cmap
   fl_color.cxx, 1082
fl_color
   Color & Font functions, 266
fl_color.cxx, 1081
   fl_cmap, 1082
fl_color_average
   Color & Font functions, 267
Fl_Color_Chooser, 451
b, 453
   Fl_Color_Chooser, 452
   g, 453
   handle, 453
hsv, 454
   hsv2rgb, 454
hue, 454
mode, 454
r, 455
rgb, 455
rgb2hsv, 455
saturation, 455
value, 456
fl_color_chooser
   Common Dialog Classes and Functions, 331, 332
Fl_Color_Chooser.H, 1082
fl_color_cube
   Enumerations.H, 1070
Fl-compose.cxx, 1082
fl_contrast
   Color & Font functions, 267
fl_copy_offscreen
   Drawing functions, 284
Fl_Copy_Surface, 456
   Fl_Copy_Surface, 457
   origin, 457, 458
   printable_rect, 458
   set_current, 458
   translate, 459
Fl_Counter, 459
draw, 461
   Fl_Counter, 460
   handle, 461
   lstep, 461
   step, 462
fl_create_offscreen
   Drawing functions, 284
Fl_Cursor
   Enumerations.H, 1064
fl_cursor
   Drawing functions, 285
FL_CURSOR_ARROW
   Enumerations.H, 1064
FL_CURSOR_CROSS
   Enumerations.H, 1064
FL_CURSOR_DEFAULT
   Enumerations.H, 1064
FL_CURSOR_DEFAULT
   Enumerations.H, 1064
FL_CURSOR_E
   Enumerations.H, 1065
FL_CURSOR_E
   Enumerations.H, 1065
FL_CURSOR_HAND
   Enumerations.H, 1064
FL_CURSOR_HELP
   Enumerations.H, 1064
FL_CURSOR_INSERT
   Enumerations.H, 1064
FL_CURSOR_MOVE
   Enumerations.H, 1064
FL_CURSOR_N
   Enumerations.H, 1065
FL_CURSOR_NE
   Enumerations.H, 1065
FL_CURSOR_NESW
   Enumerations.H, 1065
   Enumerations.H, 1065
FL_CURSOR_NW
   Enumerations.H, 1065
FL_CURSOR_S
   Enumerations.H, 1065
FL_CURSOR_SE
   Enumerations.H, 1065
FL_CURSOR_SN
   Enumerations.H, 1065
FL_CURSOR_SOUT
   Enumerations.H, 1065
FL_CURSOR_SW
   Enumerations.H, 1065
FL_CURSOR_W
   Enumerations.H, 1065
FL_CURSOR_WEST
   Enumerations.H, 1065
FL_CURSOR_WEST
   Enumerations.H, 1065
Generated by Doxygen
spacing, 499, 500
VERTICAL, 492
Fl_Float_Input, 500
Fl_Float_Input, 500
FL_FOCUS
  Enumerations.H, 1066
fl_focus_rect
  Drawing functions, 291
fl_font
  Color & Font functions, 267
fl_fontnames
  Events handling functions, 255
Fl_Fontsize
  Enumerations.H, 1062
fl_open
  Unicode and UTF-8 functions, 313
Fl_FormsBitmap, 501
  set, 501
Fl_FormsPixmap, 502
draw, 503
  Fl_FormsPixmap, 502
Pixmap, 503
  set, 503
Fl_FormsText, 504
draw, 504
fl_frame
  Drawing functions, 291
  Enumerations.H, 1071
fl_frame2
  Drawing functions, 291
Fl_Free, 504
draw, 506
  Fl_Free, 505
handle, 506
FL_FREE_BOXTYPE
  Enumerations.H, 1064
FL_FREE_LABELTYPE
  Enumerations.H, 1069
FL_FULLSCREEN
  Enumerations.H, 1068
fl_gap
  Drawing functions, 292
fl_getcwd
  Unicode and UTF-8 functions, 313
fl_getenv
  Unicode and UTF-8 functions, 314
Fl_GIF_Image, 506
  Fl_GIF_Image, 507, 508
Fl_Gl_Choice, 509
Fl_Gl_Window, 511
  make_current, 514
  make_overlay_current, 514
  mode, 514, 515
  ortho, 515
  pixel_h, 515
  pixel_w, 516
  pixels_per_unit, 516
  redraw_overlay, 516
  resize, 516
  show, 517
  swap_buffers, 517
  valid, 517
Fl_Glut_Bitmap_Font, 518
Fl_Glut_Strip, 518
Fl_Glut_Window, 519
draw, 520
draw_overlay, 521
fl_gray_ramp
  Enumerations.H, 1071
Fl_Group, 521
  ~Fl_Group, 524
    array, 524
    as_group, 524
    begin, 525
    bounds, 525
    child, 525
    clear, 525
    clip_children, 526
    current, 526
    delete_child, 526
draw, 527
draw_child, 527
draw_children, 527
draw, 528
  find, 528
Fl_Group, 523
  focus, 528
  handle, 528
  init_sizes, 528
  insert, 529
  remove, 529
  resizable, 530
  resize, 531
  sizes, 531
  update_child, 532
fl_height
  Color & Font functions, 268
Fl_Help_Block, 532
Fl_Help_Dialog, 533
  load, 534
  show, 534
  textsize, 534
  value, 535
Fl_Help_Font_Stack, 535
Generated by Doxygen

1138 INDEX

fl_boxtype.cxx, 1081
fl Intelli_t
platform_types.h, 1113
FL JOIN BEVEL
Drawing functions, 279
FL JOIN MITER
Drawing functions, 279
FL JOIN ROUND
Drawing functions, 279
FI JPEG_image, 589
FI JPEG_image, 590
FI Kdialog_Native_File_Chooser_Driver, 591
FL KEYBOARD
Enumerations.h, 1067
FL KEYDOWN
Enumerations.h, 1067
FL Label, 591
draw, 592
measure, 592
type, 592
FI Labeltype
Enumerations.h, 1068
fi latin1_to local
Color & Font functions, 268
FL LEAVE
Enumerations.h, 1066
FL Light_Button, 593
draw, 594
FI Light_Button, 593
handle, 594
FI Line_Dial, 595
fl_line_style
Drawing functions, 292
fl local_to latin1
Color & Font functions, 268
fl local_to mac roman
Color & Font functions, 269
FI Mac_App_Menu, 595
custom_application_menu_items, 596
print, 596
fl mac_os_version
Mac OS X-specific symbols, 326
fl mac_roman_to_local
Color & Font functions, 269
fl mac set about
Mac OS X-specific symbols, 326
FL MAJOR_VERSION
Enumerations.h, 1061
fl make path
Unicode and UTF-8 functions, 314
fl make path_for_file
Unicode and UTF-8 functions, 315
fl measure
Drawing functions, 292
fl measure pixmap
Drawing functions, 293
FI Menu_, 596
add, 599
clear, 602
clear_submenu, 602
copy, 603
down_box, 603
find_index, 603, 604
find_item, 604, 605
FI Menu_, 599
global, 605
insert, 605
item_pathname, 606
menu, 607
menu_end, 607
mode, 608
mvalue, 608
picked, 608
remove, 608
replace, 608
size, 609
test Shortcut, 609
text, 609
textcolor, 609
textfont, 609, 610
textsize, 610
value, 610
FI Menu_Bar, 611
draw, 612
FI Menu_Bar, 612
handle, 612
update, 613
FI Menu Button, 613
draw, 615
FI Menu Button, 615
handle, 615
popup, 616
POPUP1, 615
POPUP12, 615
POPUP123, 615
POPUP13, 615
POPUP2, 615
POPUP23, 615
POPUP3, 615
popup_buttons, 614
FL MENU DIVIDER
FI Menu_Item.H, 1091
FL MENU HORIZONTAL
FI Menu_Item.H, 1091
FL MENU INACTIVE
FI Menu_Item.H, 1091
FL MENU INVISIBLE
FI Menu_Item.H, 1091
FI Menu_Item, 616
add, 620
argument, 621
callback, 621, 622
check, 622
checkbox, 622

1138 INDEX
INDEX

checked, 622
deactivate, 622
do_callback, 622, 623
find_shortcut, 623
image, 623
insert, 624
label, 624
labelcolor, 624
labelfont, 625
measure, 625
next, 625
popup, 625
pulldown, 626
radio, 626
set, 626
setonly, 627
shortcut, 627
size, 627
submenu, 627
test_shortcut, 627
uncheck, 627
value, 628
Fl_Menu_Item.H, 1090
    FL_MENU_DIVIDER, 1091
    FL_MENU_HORIZONTAL, 1091
    FL_MENU_INACTIVE, 1091
    FL_MENU_INVISIBLE, 1091
    FL_MENU_RADIO, 1091
    FL_MENU_RESERVED, 1091
    FL_MENU_TOGGLE, 1091
    FL_MENU_VALUE, 1091
    FL_SUBMENU, 1091
    FL_SUBMENU_POINTER, 1091
FL_MENU_RADIO
    Fl_Menu_Item.H, 1091
FL_MENU_RESERVED
    Fl_Menu_Item.H, 1091
FL_MENU_TOGGLE
    Fl_Menu_Item.H, 1091
FL_MENU_VALUE
    Fl_Menu_Item.H, 1091
Fl_Menu_Window, 628
fl_message
    Common Dialog Classes and Functions, 336
    fl_message_hotspot
        Common Dialog Classes and Functions, 336, 337
    fl_message_icon
        Common Dialog Classes and Functions, 337
    fl_message_icon_label
        Common Dialog Classes and Functions, 337
    fl_message_position
        Common Dialog Classes and Functions, 337
        fl_ask.H, 1079
    fl_message_title
        Common Dialog Classes and Functions, 340
    fl_message_title_default
        Common Dialog Classes and Functions, 341
    FL_MINOR_VERSION
        Enumerations.H, 1061
    fl_mkdir
        Unicode and UTF-8 functions, 315
    FL_MOUSEWHEEL
        Enumerations.H, 1068
    fl_mult_matrix
        Drawing functions, 293
    Fl_Multi_Browser, 629
        FL_Multi_Browser, 629
    FL_MULTI_LABEL
        Enumerations.H, 1061
    Fl_Multi_Label, 630
        labela, 631
        labelb, 631
        typea, 631
        typeb, 631
    Fl_Multiline_Input, 631
        FL_Multiline_Input, 632
    Fl_Multiline_Output, 632
        FL_Multiline_Output, 633
    Fl_Native_File_Chooser, 633
        ~Fl_Native_File_Chooser, 637
        BROWSE_DIRECTORY, 636
        BROWSE_FILE, 636
        BROWSE_MULTI_DIRECTORY, 636
        BROWSE_MULTI_FILE, 636
        BROWSE_SAVE_DIRECTORY, 637
        BROWSE_SAVE_FILE, 637
        count, 637
        directory, 637
        errmsg, 637
        filename, 637
        filter, 638
        filter_value, 638
        Fl_Native_File_Chooser, 637
        NEW_FOLDER, 636
        NO_OPTIONS, 636
        Option, 636
        options, 638
        preset_file, 639
        PREVIEW, 636
        SAVEAS_CONFIRM, 636
        show, 639
        title, 639
        Type, 636
        USE_FILTER_EXT, 636
    Fl_Native_File_Chooser.H, 1091
    Fl_Nice_Slider, 639
    FL_NO_BOX
        Enumerations.H, 1063
    FL_NO_EVENT
        Enumerations.H, 1066
    FL_NO_LABEL
        Enumerations.H, 1069
    fl_nonsspacing
INDEX

fl_pop_clip
Drawing functions, 296

Fl_Positioner, 660
draw, 661
Fl_Positioner, 661
handle, 662

Fl_PostScript.H, 1092
Fl_PostScript_Close_Command, 1092

Fl_PostScript_Close_Command
Fl_PostScript.H, 1092

Fl_PostScript_File_Device, 662
begin_job, 664, 665
begin_page, 665
draw_current, 665
draw_job, 666
draw_page, 666
margins, 666
origin, 666, 667
printable_rect, 667
rotate, 667
scale, 668
set_current, 668
start_job, 668
translate, 669

Fl_Preferences, 669
~Fl_Preferences, 677
C_LOCALE, 674
CORE, 674
CORE_READ_OK, 689
CORE_SYSTEM, 674
CORE_SYSTEM_L, 674
CORE_USER, 674
CORE_USER_L, 674
CORE_WRITE_OK, 689
delete_entry, 677
delete_group, 678
dirty, 678
entries, 678
entry, 678
entry_exists, 679
file_access, 679
filename, 679, 680
Fl_Preferences, 674–677
flush, 680
get, 680–683
get_userdata_path, 684
group, 685
group_exists, 685
groups, 685
ID, 673
MEMORY, 674
new_UUID, 685
NONE, 689
Root, 674
ROOT_MASK, 674
set, 686–688
size, 688
SYSTEM, 674

SYSTEM_L, 674
UNKNOWN_ROOT_TYPE, 674
USER, 674
USER_L, 674

Fl_Preferences::Entry, 351
Fl_Preferences::Name, 1044
Name, 1045
Fl_Preferences::Node, 1045
Fl_Preferences::RootNode, 1046

Fl_Printer, 689
begin_job, 692
begin_page, 692
draw_page, 693
margins, 693
origin, 693
printable_rect, 694
rotate, 694
scale, 694
set_current, 695
translate, 695

Fl_Printer.H, 1093
Fl_Progress, 695
Fl_Progress, 696
maximum, 696
minimum, 697
value, 697

FL_PUSH

Enumerations.H, 1066

fi_push_clip
Drawing functions, 296

fi_push_matrix
Drawing functions, 297

fi_putenv
Unicode and UTF-8 functions, 316

Fl_Radio_Button, 697
Fl_Radio_Button, 698
Fl_Radio_Light_Button, 698
Fl_Radio_Round_Button, 698
Fl_Radio_Round_Button, 699

FL_READ

Enumerations.H, 1062

fi_read_image
Drawing functions, 297

Fl_Rect, 699
b, 700
r, 700

fl_rect
Drawing functions, 297

fl_rect.cxx, 1093

fl_rectbound
fl_boxtype.cxx, 1081

fl_rectf
Drawing functions, 298

Fl_Region
platform_types.h, 1113

fl_register_images
Fi_Shared_Image.H, 1094

FL_RELEASE

Generated by Doxygen
INDEX 1143

FL_SHOW
Enumerations.H, 1068

fl_show_colormap
Color & Font functions, 269
fl_show_colormap.H, 1094

Fl_Simple_Counter, 734
Fl_Simple_Terminal, 735
ansi, 737, 738
append, 738
clear, 739
current_style_index, 739
draw, 739
enforce_history_lines, 740
enforce_stay_at_bottom, 740
history_lines, 740
normal_style_index, 740
printf, 741
remove_lines, 741
stay_at_bottom, 741, 742
style_table, 742
style_table_size, 743
text, 743
vprintf, 744

Fl_Single_Window, 744
show, 745

fl_size
Color & Font functions, 270

Fl_Slider, 745
bounds, 747
draw, 747
Fl_Slider, 747
handle, 747
scrollvalue, 748
slider_size, 748

FL_SOLID
Drawing functions, 279

Fl_Spinner, 748
Fl_Spinner, 750
handle, 750
resize, 751
step, 751
type, 752
value, 752
wrap, 752

Fl_Spinner::Fl_Spinner_Input, 753
handle, 753

fl_stat
Unicode and UTF-8 functions, 317

fl_strdup
Fl_string, 350
Fl_String, 754
Fl_string, 350
fl_strdup, 350
Fl_String.cxx, 1095
Fl_String.H, 1095
fl_string_functions.h, 1095

FL_SUBMENU
Fl_Menu_Item.H, 1091

FL_SUBMENU_POINTER
Fl_Menu_Item.H, 1091

Fl_Surface_Device, 755
draw_current, 756
input_current, 756
push_current, 756
set_current, 756
surface, 756

Fl_SVG_File_Surface, 758
~Fl_SVG_File_Surface, 759
close, 759
Fl_SVG_File_Surface, 759
origin, 759, 760
printable_rect, 760
translate, 760

Fl_SVG_Image, 760
as_svg_image, 763
color_average, 763
copy, 763
desaturate, 764
draw, 764
Fl_SVG_Image, 762
normalize, 764
proportional, 765
resize, 764

FL_SYMBOL_LABEL
Enumerations.H, 1061

Fl_Sys_Menu_Bar, 765
about, 768
add, 768, 769
clear, 769
clear_submenu, 769
create_window_menu, 769
draw, 770
Fl_Sys_Menu_Bar, 768
insert, 770
menu, 771
mode, 771
no_window_menu, 767
remove, 771
replace, 771
tabbing_modeAutomatic, 768
tabbing_modeNone, 768
tabbing_modePreferred, 768
update, 772
window_menu_style, 772
window_menu_style_enum, 767
Fl_Sys_Menu_Bar.H, 1095

fl_system
Unicode and UTF-8 functions, 318

Fl_Table, 772
~Fl_Table, 780
array, 780
callback, 780
callback_col, 781
callback_context, 781
callback_row, 781
child, 781

Generated by Doxygen
kf_default, 861
kf_delete, 861
kf_down, 861
kf_end, 861
kf_enter, 862
kf_home, 862
kf_ignore, 862
kf_insert, 862
kf_left, 862
kf_m_s_move, 862
kf_meta_move, 862
kf_move, 863
kf_page_down, 863
kf_page_up, 863
kf_paste, 863
kf_right, 863
kf_select_all, 863
kf_shift_move, 864
kf_undo, 864
kf_up, 864
remove_key_binding, 864
update, 869
flip_text::Key_Binding, 1044
fl_text_extents
Color & Font functions, 270, 271
Fl_Text_Selection, 865
end, 867
includes, 867
length, 867
position, 867
selected, 868
set, 868
start, 868
update, 869
FL_THIN_DOWN_BOX
Enumerations.H, 1063
FL_THIN_DOWN_FRAME
Enumerations.H, 1063
FL_THIN_UP_BOX
Enumerations.H, 1063
FL_THIN_UP_FRAME
Enumerations.H, 1063
Fl_Tile, 869
Fl_Tile, 870
handle, 871
position, 871
resize, 871
Fl_Tiled_Image, 872
color_average, 873
copy, 874
desaturate, 874
draw, 874
Fl_Tiled_Image, 873
Fl_Timeout, 875
current, 876
current_timeout, 879
elapse_timeouts, 876
first_timeout, 876
free_timeout, 879
generate, 877
has_timeout, 877
insert, 878
make_current, 878
release, 878
remove_timeout, 878
time_to_wait, 879
Fl_Timeout.cxx, 1096
Fl_Timeout.h, 1096
Fl_Timer, 880
direction, 881
draw, 881
Fl_Timer, 881
handle, 881
suspended, 883
Fl_Toggle_Button, 883
Fl_Toggle_Button, 884
Fl_Tooltip, 884
color, 886
current, 886
delay, 886
disable, 886
enable, 887
enabled, 887
enter_area, 887
font, 887
hidedelay, 887
hoverdelay, 888
margin_height, 888
margin_width, 888
size, 888
textcolor, 889
wrap_width, 889
fl_transform_dx
Drawing functions, 301
fl_transform_dy
Drawing functions, 302
fl_transform_x
Drawing functions, 302
fl_transform_y
Drawing functions, 302
fl_transformed_vertex
Drawing functions, 302
fl_translate
Drawing functions, 303
Fl_Tree, 889
add, 900, 901
calc_dimensions, 901
calc_tree, 901
callback_item, 902
callback_reason, 902
clear, 902
clear_children, 902
close, 903
closeicon, 904
current, 904
deselect, 904, 905
deselect_all, 905
display, 906
displayed, 906
extend_selection, 906
extend_selection_dir, 907
find_clicked, 907
find_item, 908
first, 908
first_selected_item, 908
first_visible, 909
first_visible_item, 909
get_selected_items, 909
handle, 910
hposition, 910
insert, 911
insert_above, 911
is_close, 912
is_hscroll_visible, 912
is_open, 913
is_scrollbar, 913
is_selected, 914
is_vscroll_visible, 914
item_clicked, 915
item_draw_mode, 915
item_labelbgcolor, 915, 916
item_labelbgcolor, 916
item_labelfont, 916
item_labelsize, 916
item_pathname, 916
item_reselect_mode, 917
last, 917
last_selected_item, 917
last_visible, 918
last_visible_item, 918
load, 918
next, 918
next_item, 919
next_selected_item, 920
next_visible_item, 920
open, 921
open_toggle, 922
openicon, 922, 923
prev, 923
recalc_tree, 923
remove, 923
resize, 924
root, 924
root_label, 924
scrollbar_size, 924
select, 925
select_all, 926
select_only, 926
select_toggle, 927
selectbox, 927
selectmode, 927, 928
set_item_focus, 928
show_item, 928
show_item_bottom, 929
show_item_middle, 929
show_item_top, 929
show_self, 929
showcollapse, 929, 930
showroot, 930
sortorder, 930
usericon, 930
vposition, 930, 931
Fl_Tree.H, 1096
Fl_TreeConnector
Fl_Tree_Prefs.H, 1099
FL_TREE_REASON_CLOSED, 1097
FL_TREE_REASON_DESELECTED, 1097
FL_TREE_REASON_DRAGGED, 1097
FL_TREE_REASON_NONE, 1097
FL_TREE_REASON_OPENED, 1097
FL_TREE_REASON_RESELECTED, 1097
FL_TREE_REASON_SELECTED, 1097
Fl_Tree_Item, 931
activate, 937
add, 937, 938
calc_item_height, 938
child, 938
decactivate, 939
deparent, 939
depth, 939
deselect_all, 939
draw, 939
draw_horizontal_connector, 940
draw_item_content, 940
draw_vertical_connector, 941
drawbgcolor, 941
drawfgcolor, 941
find_child, 942
find_child_item, 942
find_clicked, 943
find_item, 943
Fl_Tree_Item, 936, 937
hide_widgets, 943
insert, 943
insert_above, 944
label, 944
label_h, 944
label_w, 944
label_x, 944
label_y, 945
labelbgcolor, 945
move, 945, 946
move_above, 946
move_below, 946
move_into, 946
Enumerations.H, 1069
FL_WHEN_ENTER_KEY_CHANGED
Enumerations.H, 1069
FL_WHEN_NEVER
Enumerations.H, 1069
FL_WHEN_NOT_CHANGED
Enumerations.H, 1069
FL_WHEN_RELEASE
Enumerations.H, 1069
FL_WHEN_RELEASE_ALWAYS
Enumerations.H, 1069
Fl_Widget, 977
∼−Fl_Widget, 985
activate, 985
active, 985
active_r, 986
align, 986
argument, 987
as_gl_window, 987
as_group, 987
as_window, 988
box, 988
callback, 989, 990
CHANGED, 984
changed, 990
clear_active, 990
clear_changed, 991
clear_damage, 991
clear_output, 991
clear_visible, 991
clear_visible_focus, 991
CLIP_CHILDREN, 984
color, 992
color2, 992, 993
contains, 993
COPYED_LABEL, 984
COPYED_TOOLTIP, 984
copy_label, 993
copy_tooltip, 993
damage, 994
deactivate, 995
default_callback, 995
deimage, 995, 996
do_callback, 996, 997
draw, 997
draw_focus, 997, 998
draw_label, 999
Fl_Widget, 985
FORCE_POSITION, 984
FULLSCREEN, 984
GROUP_RELATIVE, 984
h, 999
handle, 999
hide, 1000
image, 1000, 1001
INACTIVE, 984
inside, 1001
INVISIBLE, 984
is_label_copied, 1001
label, 1001, 1002
label_shortcut, 1002
labelcolor, 1003
labelfont, 1003
labels, 1004
labeltype, 1004
MAC_USE_ACCENTS_MENU, 984
measure_label, 1005
MENU_WINDOW, 984
MODAL, 984
NEEDS_KEYBOARD, 984
NO_OVERLAY, 984
NOBORDER, 984
NON_MODAL, 984
OUTPUT, 984
output, 1005
OVERRIDE, 984
parent, 1005
position, 1005
redraw, 1006
redraw_label, 1006
resize, 1006
selection_color, 1006, 1007
set_active, 1007
set_changed, 1007
set_output, 1007
set_visible, 1007
set_visible_focus, 1008
SHORTCUT_LABEL, 984
shortcut_label, 1008
show, 1008
size, 1008
take_focus, 1009
takesevents, 1009
test_shortcut, 1009
tooltip, 1010
TOOLTIP_WINDOW, 984
top_window, 1011
top_window_offset, 1011
type, 1011
user_data, 1011, 1012
USERFLAG1, 985
USERFLAG2, 984
USERFLAG3, 984
visible, 1012
VISIBLE_FOCUS, 984
visible_focus, 1012
visible_r, 1013
w, 1013
when, 1013
window, 1014
x, 1014
y, 1015
Fl_Widget.H, 1103
FL_RESERVED_TYPE, 1104
Fl_Widget_Surface, 1015
draw, 1016
INDEX

Fl_Window, 1031
fullscreen_screens
   Fl_Window, 1032

g
   Fl_Color_Chooser, 453
gap
   Fl_Flex, 495
get
   Fl_Preferences, 680–683
   Fl_Shared_Image, 732
   Fl_Timeout, 877
get_absolute_top_line_number
   Fl_Text_Display, 835
get_color
   Color & Font functions, 271, 272
get_font
   Color & Font functions, 272
get_font_name
   Color & Font functions, 272
get_font_sizes
   Color & Font functions, 272
get_key
   Events handling functions, 253
get_mouse
   Events handling functions, 253
get_selected_items
   Fl_Tree, 909
get_selection
   Fl_Table, 785
get_system_colors
   Fl, 368
get_userdata_path
   Fl_Preferences, 684
gl.h, 1106
gl_color, 1108
gl_draw, 1108, 1109
gl_font, 1109
gl_rect, 1110
gl_rectf, 1110
gl_texture_pile_height, 1110
gl_color
   gl.h, 1108
gl_draw
   gl.h, 1108, 1109
gl_font
   gl.h, 1109
gl_rect
   gl.h, 1110
gl_rectf
   gl.h, 1110
gl_texture_pile_height
   gl.h, 1110
gl_visual
   Fl, 368
GLContext
   platform_types.h, 1114
global
   Fl_Menu_, 605
global_key_bindings
   Fl_Text_Editor, 865
grab
   Windows handling functions, 241
grammar_underline_color
   Fl_Text_Display, 835, 836
group
   Fl_Preferences, 685
group_exists
   Fl_Preferences, 685
GROUP_RELATIVE
   Fl_Widget, 984
groups
   Fl_Preferences, 685
h
   Fl_Image, 554, 555
   Fl_Widget, 999
handle
   Events handling functions, 253
   Fl_Adjuster, 378
   Fl_Box, 384
   Fl_Browser_, 412
   Fl_Button, 425
   Fl_Check_Browser, 436
   Fl_Choice, 443
   Fl_Clock, 446
   Fl_Color_Chooser, 453
   Fl_Counter, 461
   Fl_Dial, 465
   Fl_File_Input, 488
   Fl_Free, 506
   Fl_Group, 528
   Fl_Input, 564
   Fl_Light_Button, 594
   Fl_Menu_Bar, 612
   Fl_Menu_Button, 615
   Fl_Positioner, 662
   Fl_Repeat_Button, 702
   Fl_Return_Button, 704
   Fl_Roller, 711
   Fl_Scroll, 718
   Fl_Scrollbar, 722
   Fl_Secret_Input, 725
   Fl_Slider, 747
   Fl_Spinner, 750
   Fl_Spinner::Fl_Spinner_Input, 753
   Fl_Tabs, 798
   Fl_Tile, 871
   Fl_Timer, 881
   Fl_Tree, 910
   Fl_Value_Input, 969
   Fl_Value_Output, 973
   Fl_Value_Slider, 976
   Fl_Widget, 999
   Fl_Window, 1032
handle_
   Events handling functions, 254
handle_key
insert_
   Fl_Text_Buffer, 809
insert_above
   Fl_Tree, 911
   Fl_Tree_Item, 944
insert_mode
   Fl_Text_Editor, 860
insert_position
   Fl_Text_Display, 838
insertfile
   Fl_Text_Buffer, 809
inserting
   Fl_Browser_, 413
inside
   Fl_Widget, 1001
INVISIBLE
   Fl_Widget, 984
is_close
   Fl_Tree, 912
is_hscroll_visible
   Fl_Tree, 912
is_interactive_resize
   Fl_Table, 786
is_label_copied
   Fl_Widget, 1001
is_open
   Fl_Tree, 913
is_resizable
   Fl_Window, 1035
is_scheme
   Fl, 368
is_scrollbar
   Fl_Tree, 913
is_selected
   Fl_Table, 786
   Fl_Tree, 914
is_vscroll_visible
   Fl_Tree, 914
is_word_separator
   Fl_Text_Buffer, 810
isSetSize
   Fl_Flex, 496
item_at
   Fl_Browser, 395
   Fl_Browser_, 414
   Fl_Check_Browser, 436
item_clicked
   Fl_Tree, 915
item_draw
   Fl_Browser, 396
item_draw_mode
   Fl_Tree, 915
   Fl_Tree_Prefs, 961
item_first
   Fl_Browser, 396
   Fl_Browser_, 414
   Fl_Check_Browser, 437
item_height
   Fl_Browser, 396
   Fl_Browser_, 414
   Fl_Check_Browser, 437
item_labelbgcolor
   Fl_Tree, 915, 916
   Fl_Tree_Prefs, 961
item_labelbgcolor
   Fl_Tree, 916
item_labelfont
   Fl_Tree, 916
item_labelsize
   Fl_Tree, 916
item_last
   Fl_Browser, 397
   Fl_Browser_, 414
item_next
   Fl_Browser, 397
   Fl_Browser_, 415
   Fl_Check_Browser, 437
item_pathname
   Fl_Menu_, 606
   Fl_Tree, 916
item_prev
   Fl_Browser, 397
   Fl_Browser_, 415
   Fl_Check_Browser, 437
item_quick_height
   Fl_Browser_, 415
item_reselect_mode
   Fl_Tree, 917
item_select
   Fl_Browser, 398
   Fl_Browser_, 415
   Fl_Check_Browser, 437
item_selected
   Fl_Browser, 398
   Fl_Browser_, 416
   Fl_Check_Browser, 438
item_swap
   Fl_Browser, 398
   Fl_Browser_, 416
   Fl_Check_Browser, 438
item_text
   Fl_Browser, 399
   Fl_Browser_, 416
   Fl_Check_Browser, 438
item_width
   Fl_Browser, 399
   Fl_Browser_, 416
   Fl_Check_Browser, 439
keyboard_screen_scaling
   Screen functions, 260
kf_backspace
   Fl_Text_Editor, 860
kf_c_s_move
   Fl_Text_Editor, 860
kf_copy
   Fl_Text_Editor, 860
line_text
  Fl_Text_Buffer, 810
lineno
  Fl_Browser, 399
linenumber_align
  Fl_Text_Display, 839
linenumberbgcolor
  Fl_Text_Display, 839
linenumberbgcolor
  Fl_Text_Display, 840
linenumber_font
  Fl_Text_Display, 840
linenumberformat
  Fl_Text_Display, 840
linenumber_size
  Fl_Text_Display, 840
linenumber_width
  Fl_Text_Display, 840
lineposition
  Fl_Browser, 400
linesize
  Fl_Scrollbar, 722
link
  Fl_Help_View, 541
load
  Fl_Browser, 400
  Fl_File_Browser, 474
  Fl_File_Icon, 485
  Fl_Help_Dialog, 534
  Fl_Help_View, 541
  Fl_Plugin_Manager, 657
  Fl_Treeprefs, 961
  Fl_Tree, 918
load_fti
  Fl_File_Icon, 485
load_image
  Fl_File_Icon, 485
load_system_icons
  Fl_File_Icon, 485
loadfile
  Fl_Text_Buffer, 811
lock
  Multithreading support functions, 304
longest_vline
  Fl_Text_Display, 841
lstep
  Fl_Counter, 461
Mac OS X-specific symbols, 325
  fl_mac_os_version, 326
  fl_mac_set_about, 326
  fl_open_callback, 326
mac.H, 1110
MAC_USE_ACCENTS_MENU
  Fl_Widget, 984
maintain_absolute_top_line_number
  Fl_Text_Display, 841
maintaining_absolute_top_line_number
  Fl_Text_Display, 841
make_current
  Fl_Gl_Window, 514
  Fl_Timeout, 878
  Fl_Window, 1035
make_overlay_current
  Fl_Gl_Window, 514
make_visible
  Fl_Browser, 400
manage_item_destroy
  Fl_Tree_Item_Array, 955
margin
  Fl_Flex, 496, 497
  Fl_Tooltip, 888
marginwidth
  Fl_Tooltip, 888
marginbottom
  Fl_Treeprefs, 961
margins
  Fl_Flex, 497
  Fl_Paged_Device, 650
  Fl_PostScript_File_Device, 666
  Fl_Printer, 693
mark
  Fl_Input_, 573
maxsize
  Fl_RGB_Image, 709
maximum
  Fl_Progress, 696
  Fl_Valuator, 965
maximumsize
  Fl_Input_, 574
maxsize
  Fl_Chart, 433
measure
  Fl_Label, 592
  Fl_Menu_Item, 625
measure_deleted_lines
  Fl_Text_Display, 841
measure_label
  Fl_Widget, 1005
measure_proportional_character
  Fl_Text_Display, 842
measure_vline
  Fl_Text_Display, 842
MEMORY
  Fl_Preferences, 674
menu
  Fl_Menu_, 607
  Fl_Sys_Menu_Bar, 771
menu_end
  Fl_Menu_, 607
menu_linespacing
  Fl_Gl_Window, 514
menu_x
  Fl_Input_Choice, 587
Option
  Fl_Native_File_Chooser, 636
option
  Fl, 369, 370
OPTION_ARROW_FOCUS
  Fl, 360
OPTION_DND_TEXT
  Fl, 360
OPTION_FNFC_USES_GTK
  Fl, 361
OPTION_LAST
  Fl, 361
OPTION_PRINTER_USES_GTK
  Fl, 361
OPTION_SHOW_SCALING
  Fl, 361
OPTION_SHOW_TOOLTIPS
  Fl, 360
OPTION_VISIBLE_FOCUS
  Fl, 360
options
  Fl_Native_File_Chooser, 638
ORNAMENTATION
  Fl_Paged_Device, 649
origin
  Fl_Copy_Surface, 457, 458
  Fl_EPS_File_Surface, 470, 471
  Fl_Image_Surface, 560
  Fl_PostScript_File_Device, 666, 667
  Fl_Printer, 693
  Fl_SVG_File_Surface, 759, 760
  Fl_Widget_Surface, 1017
original
  Fl_Shared_Image, 733
ortho
  Fl_Gl_Window, 515
os_id
  Fl_Window, 1035
OUTPUT
  Fl_Widget, 984
output
  Fl_Widget, 1005
outputfile
  Fl_Text_Buffer, 811
OVERRIDE
  Fl_Widget, 984
overstrike
  Fl_Text_Display, 843
own_colormap
  Fl, 370
Page_Format
  Fl_Paged_Device, 648
Page_Layout
  Fl_Paged_Device, 648
parent
  Fl_Tree_Item, 948
  Fl_Widget, 1005
paste
  Selection & Clipboard functions, 258
picked
  Fl_Menu_, 608
pixel_h
  Fl_Gl_Window, 515
pixel_w
  Fl_Gl_Window, 516
pixels_per_unit
  Fl_Gl_Window, 516
Pixmap
  Fl_FormsPixmap, 503
platform_types.h, 1113
  fl_intptr_t, 1113
  Fl_Offscreen, 1113
  Fl_Region, 1113
  fl_uintptr_t, 1114
  GLContext, 1114
pop_current
  Fl_Surface_Device, 756
popup
  Fl_Menu_Button, 616
  Fl_Menu_Item, 625
POPUP1
  Fl_Menu_Button, 615
POPUP12
  Fl_Menu_Button, 615
POPUP123
  Fl_Menu_Button, 615
POPUP13
  Fl_Menu_Button, 615
POPUP2
  Fl_Menu_Button, 615
POPUP23
  Fl_Menu_Button, 615
POPUP3
  Fl_Menu_Button, 615
popup_buttons
  Fl_Menu_Button, 614
PORTRAIT
  Fl_Paged_Device, 649
position
  Fl_Browser_, 417
  Fl_Input_, 574
  Fl_Text_Selection, 867
  Fl_Tile, 871
  Fl_Widget, 1005
position_style
  Fl_Text_Display, 843
position_to_line
  Fl_Text_Display, 844
position_to_linecol
  Fl_Text_Display, 844
precision
  Fl_Valuator, 966
prefs
  Fl_Tree_Item, 948
preset_file
   Fl_Native_File_Chooser, 639
prev
   Fl_Tree, 923
   Fl_Tree_Item, 948
prev_char
   Fl_Text_Buffer, 811
prev_displayed
   Fl_Tree_Item, 948
prev_sibling
   Fl_Tree_Item, 948
prev_visible
   Fl_Tree_Item, 948
PREVIEW
   Fl_Native_File_Chooser, 636
preview
   Fl_File_Chooser, 480
print
   Fl_Mac_App_Menu, 596
print_window_part
   Fl_Widget_Surface, 1017
printable_rect
   Fl_Copy_Surface, 458
   Fl_EPS_File_Surface, 471
   Fl_Image_Surface, 561
   Fl_PostScript_File_Device, 667
   Fl_Printer, 694
   Fl_SVG_File_Surface, 760
   Fl_Widget_Surface, 1018
printf
   Fl_Simple_Terminal, 741
   Fl_Text_Buffer, 812
program_should_quit
   Fl, 370, 371
proportional
   Fl_SVG_Image, 765
pulldown
   Fl_Menu_Item, 626
push
   Fl_Tabs, 799
push_current
   Fl_Surface_Device, 757
pushed
   Events handling functions, 254
r
   Fl_Color_Chooser, 455
   Fl_Rect, 700
radio
   Fl_Menu_Item, 626
range
   Fl_Valuator, 966
readonly
   Fl_Input_, 575
readqueue
   Fl, 371
ready
   Fl, 371
recalc_dimensions
   Fl_Table, 786
recalc_scrollbars
   Fl_Scroll, 718
recalc_tree
   Fl_Tree, 923
   Fl_Tree_Item, 949
rectangle_capture
   Fl_Device_Plugin, 463
redisplay_range
   Fl_Text_Display, 845
redraw
   Fl_Widget, 1006
   Fl_Table, 786
redraw_label
   Fl_Widget, 1006
redraw_line
   Fl_Browser_, 418
redraw_lines
   Fl_Browser_, 418
redraw_overlay
   Fl_Gl_Window, 516
   Fl_Overlay_Window, 643
redraw_range
   Fl_Table, 786
refcount
   Fl_Shared_Image, 733
release
   Fl, 371
   Fl_Image, 555
   Fl_Shared_Image, 733
   Fl_Timeout, 878
release_widget_pointer
   Safe widget deletion support functions, 306
reload_scheme
   Fl, 372
remove
   Fl_Browser, 401
   Fl_Check_Browser, 439
   Fl_Group, 529
   Fl_Menu_, 608
   FlSys_Menu_Bar, 771
   Fl_Text_Buffer, 812
   Fl_Tree, 923
   Fl_Tree_Item_Array, 956
remove_
   Fl_Text_Buffer, 812
remove_check
   Fl, 372
remove_child
   Fl_Tree_Item, 949
remove_handler
   Events handling functions, 255
remove_icon
   Fl_Browser, 402
remove_key_binding
   Fl_Text_Editor, 864
remove_lines
   Fl_Simple_Terminal, 741
remove_system_handler
Events handling functions, 255
remove_timeout
  Fl, 372
  Fl_Timeout, 878
removePlugin
  Fl_Plugin_Manager, 657
reparent
  Fl_Tree_Item, 949
  Fl_Tree_Item_Array, 956
repeat_timeout
  Fl, 372
replace
  Fl_Chart, 433
  Fl_Input_, 575
  Fl_Menu_, 608
  Fl_Sys_Menu_Bar, 771
  Fl_Text_Buffer, 812
  Fl_Tree_Item, 950
  Fl_Tree_Item_Array, 956
replace_child
  Fl_Tree_Item, 950
replacing
  Fl_Browser_, 418
rescale
  Fl_Image_Surface, 561
reset_absolute_top_line_number
  Fl_Text_Display, 846
resizable
  Fl_Group, 530
resize
  Fl_Browser_, 418
  Fl_Double_Window, 467
  Fl_Flex, 498
  Fl_Gl_Window, 516
  Fl_Group, 531
  Fl_Input_, 576
  Fl_Overlay_Window, 643
  Fl_Scroll, 718
  Fl_Spinner, 751
  Fl_SVG_Image, 764
  Fl_Table, 787
  Fl_Text_Display, 846
  Fl_Tile, 871
  Fl_Tree, 924
  Fl_Value_Input, 969
  Fl_Widget, 1006
  Fl_Window, 1036
REVERSED
  Fl_Paged_Device, 649
rewind_lines
  Fl_Text_Buffer, 813
  Fl_Text_Display, 846
rgb
  Fl_Color_Chooser, 455
rgb2hsv
  Fl_Color_Chooser, 455
RGB_scaling
  Fl_Image, 556
Root
  Fl_Preferences, 674
root
  Fl_Tree, 924
root_label
  Fl_Tree, 924
ROOT_MASK
  Fl_Preferences, 674
rotate
  Fl_Paged_Device, 650
  Fl_PostScript_File_Device, 667
  Fl_Printer, 694
round
  Fl_Valuator, 966
ROW
  Fl_Flex, 493
row_col_clamp
  Fl_Table, 787
row_header
  Fl_Table, 787
row_height
  Fl_Table, 787
row_height_all
  Fl_Table, 787
row_resize
  Fl_Table, 787
row_resize_min
  Fl_Table, 788
row_selected
  Fl_Table_Row, 792
run
  Fl, 372
Safe widget deletion support functions, 305
clear_widget_pointer, 305
delete_widget, 306
do_widget_deletion, 306
release_widget_pointer, 306
watch_widget_pointer, 307
saturation
  Fl_Color_Chooser, 455
SAVEAS_CONFIRM
  Fl_Native_File_Chooser, 636
savefile
  Fl_Text_Buffer, 813
scale
  Fl_Image, 556
  Fl_Paged_Device, 650
  Fl_PostScript_File_Device, 668
  Fl_Printer, 694
scaling_algorithm
  Fl_Image, 557
scheme
  Fl, 373
Screen functions, 259
keyboard_screen_scaling, 260
screen_dpi, 261
screen_num, 261
screen_scale, 261
screen_scaling_supported, 262
screen_work_area, 262
screen_xywh, 263, 264
screen_dpi
  Screen functions, 261
screen_num
  Fl_Window, 1036
  Screen functions, 261
screen_scale
  Screen functions, 261
screen_scaling_supported
  Screen functions, 262
screen_work_area
  Screen functions, 262
screen_xywh
  Screen functions, 263, 264
scroll
  Fl_Text_Display, 846
scroll
  Fl_Text_Display, 847
scroll_timer_cb
  Fl_Text_Display, 847
scroll_to
  Fl_Scroll, 719
scrollbar
  Fl_Browser_, 421
scrollbar_align
  Fl_Text_Display, 847
scrollbar_left
  Fl_Browser_, 419
scrollbar_right
  Fl_Browser_, 419
scrollbar_size
  Fl, 373
  Fl_Browser_, 419
  Fl_Help_View, 542
  Fl_Scroll, 719
  Fl_Table, 788
  Fl_Text_Display, 847, 848
  Fl_Tree, 924
scrollbar_width
  Fl_Browser_, 419, 420
  Fl_Text_Display, 848
scrollvalue
  Fl_Slider, 748
search_backward
  Fl_Text_Buffer, 813
search_forward
  Fl_Text_Buffer, 814
second
  Fl_Clock_Output, 449
secondary_selection_color
  Fl_Text_Display, 848
secondary_selection_text
  Fl_Text_Buffer, 814
select
  Fl_Browser, 402
  Fl_Browser_, 420
  Fl_Tree, 925
  Fl_Tree_Item, 951
select_all
  Fl_Tree, 926
  Fl_Tree_Item, 951
select_all_rows
  Fl_Table_Row, 793
select_only
  Fl_Browser_, 420
  Fl_Tree, 926
select_row
  Fl_Table_Row, 793
select_toggle
  Fl_Tree, 927
selectbox
  Fl_Tree, 927
selected
  Fl_Browser, 402
  Fl_Text_Selection, 888
selection
  Fl_Browser_, 420
Selection & Clipboard functions, 259
Selection & Clipboard functions, 256
add_clipboard_notify, 257
clipboard_contains, 257
copy, 257
dnd, 258
paste, 258
selection, 259
selection_owner, 259
selection_color
  Fl_Widget, 1006, 1007
selection_owner
  Selection & Clipboard functions, 259
selection_text
  Fl_Text_Buffer, 814
selectmode
  Fl_Tree, 927, 928
  Fl_Tree_Prefs, 961
set
  Fl_Button, 425
  Fl_FormsBitmap, 501
  Fl_FormsPixmap, 503
  Fl_Menu_Item, 626
  Fl_Preferences, 686–688
  Fl_Text_Selection, 888
set_active
  Fl_Widget, 1007
set_atclose
  Windows handling functions, 242
set_box_color
  Fl, 374
set_changed
  Fl_Widget, 1007
set_color
  Color & Font functions, 272, 273
set_current
  Fl_Copy_Surface, 458
INDEX 1163

FL_Tree_Prefs, 962
spacing
FL_Flex, 499, 500
spelling_underline_color
FL_Text_Display, 850
start
FL_Text_Selection, 868
start_job
FL_Paged_Device, 651
FL_PostScript_File_Device, 668
start_page
FL_Paged_Device, 651
static_value
FL_Input_, 577, 578
stay_at_bottom
FL_Simple_Terminal, 741, 742
step
FL_Counter, 462
FL_Spinner, 751
FL_Valuator, 966
STRICT_RFC3629
Unicode and UTF-8 functions, 312
string_width
FL_Text_Display, 850
style_buffer
FL_Text_Display, 851
style_table
FL_Simple_Terminal, 742
style_table_size
FL_Simple_Terminal, 743
submenu
FL_Menu_Item, 627
surface
FL_Surface_Device, 757
suspended
FL_Timer, 883
swap
FL_Browser, 403
swapBuffers
FL_GL_Window, 517
swap_children
FL_Tree_Item, 951
swapping
FL_Browser_, 421
SYSTEM
FL_Preferences, 674
SYSTEM_L
FL_Preferences, 674
tab_align
FL_Tabs, 799
tab_cell_nav
FL_Table, 789
tab_distance
FL_Text_Buffer, 815
tab_nav
FL_Input_, 578
FL_Text_Editor, 864, 865
tabbing_mode_automatic
FL_Sys_Menu_Bar, 768
tabbing_mode_none
FL_Sys_Menu_Bar, 768
tabbing_mode_preferred
FL_Sys_Menu_Bar, 768
table_box
FL_Table, 789
table_resized
FL_Table, 789
table_scrolled
FL_Table, 790
TableContext
FL_Table, 779
TABLOID
FL_Paged_Device, 648
take_focus
FL_Widget, 1009
takeEvents
FL_Widget, 1009
test_shortcut
Events handling functions, 255
FL_Menu_, 609
FL_Menu_Item, 627
FL_Widget, 1009
text
FL_Browser, 403, 404
FL_Menu_, 609
FL_Simple_Terminal, 743
FL_Text_Buffer, 815
text_range
FL_Text_Buffer, 815
textcolor
FL_Input_, 580
FL_Menu_, 609
FL_Text_Display, 851
FL_Tooltip, 889
FL_Value_Input, 970
FL_Value_Output, 973, 974
FL_Value_Slider, 976
textFont
FL_Browser_, 421
FL_Input_, 580
FL_Menu_, 609, 610
FL_Text_Display, 851
FL_Value_Input, 970, 971
FL_Value_Output, 974
FL_Value_Slider, 976, 977
textsize
FL_Browser, 404
FL_Help_Dialog, 534
FL_Input_, 581
FL_Menu_, 610
FL_Text_Display, 852
FL_Value_Input, 971
FL_Value_Output, 974
FL_Value_Slider, 977
thread_message
Multithreading support functions, 304

Generated by Doxygen
time_to_wait
  Fl_Timeout, 879
title
  Fl_Native_File_Chooser, 639
tooltip
  Fl_Widget, 1010
  TOOLTIP_WINDOW
  Fl_Widget, 984
top_row
  Fl_Table, 790
top_window
  Fl_Widget, 1011
top_window_offset
  Fl_Widget, 1011	
topline
  Fl_Browser, 404
  Fl_Help_View, 542, 544
transcoding_warning_action
  Fl_Text_Buffer, 817
translate
  Fl_Copy_Surface, 459
  Fl_EPS_File_Surface, 471
  Fl_Image_Surface, 561
  Fl_PostScript_File_Device, 669
  Fl_Printer, 695
  Fl_SVG_File_Surface, 760
  Fl_Widget_Surface, 1018
tree
  Fl_Tree_Item, 952>Type
  Fl_Native_File_Chooser, 636
type
  Fl_File_Icon, 486
  Fl_Label, 592
  Fl_Spinner, 752
  Fl_Table_Row, 793
  Fl_Widget, 1011
typea
  Fl_Multi_Label, 631
typeb
  Fl_Multi_Label, 631
uncache
  Fl_Bitmap, 381
  Fl_Image, 557
  Fl_Pixmap, 655
  Fl_RGB_Image, 709
  Fl_Shared_Image, 734
uncHECK
  Fl_Menu_Item, 627
undo
  Fl_Input_, 581
Unicode and UTF-8 functions, 309
  ERRORS_TO_CP1252, 311
  ERRORS_TO_ISO8859_1, 311
  fl_access, 312
  fl_chdir, 312
  fl_chmod, 313
  fl_fopen, 313
  fl_getcwd, 313
  fl_getenv, 314
  fl_make_path, 314
  fl_make_path_for_file, 315
  fl_mkdir, 315
  fl_nonspaceR_Branching, 315
  fl_open, 315
  fl_open_ext, 316
  fl_putenv, 316
  fl_rename, 317
  fl_rmdir, 317
  fl_stat, 317
  fl_system, 318
  fl_ucs_to_Utf16, 318
  fl_unlink, 318
  fl_utf8back, 319
  fl_utf8bytes, 319
  fl_utf8decode, 319
  fl_utf8encode, 320
  fl_utf8from_mb, 320
  fl_utf8fromom, 320
  fl_utf8fromwc, 320
  fl_utf8in, 321
  fl_utf8len, 321
  fl_utf8len1, 321
  fl_utf8locale, 321
  fl_utf8test, 322
  fl_utf8to_mb, 322
  fl_utf8toa, 322
  fl_utf8toUtf16, 323
  fl_utf8towc, 323
  fl_utf_strcasecmp, 323
  fl_utf_strncasecmp, 323
  fl_utf_strcascmp, 324
  fl_utf_tolower, 324
  fl_utf_toupper, 324
  fl_wcwidth, 324
  fl_wcwidth_, 325
  STRICt_RFC3629, 312
UNKNOWN_ROOT_TYPE
  Fl_Preferences, 674
unlock
  Multithreading support functions, 304
up_down_position
  Fl_Input_, 581
update
  Fl_Menu_Bar, 613
  Fl_Sys_Menu_Bar, 772
  Fl_Text_Selection, 869
update_child
  Fl_Group, 532
update_h_scrollbar
  Fl_Text_Display, 852
update_line_starts
  Fl_Text_Display, 852
update_menubutton
  Fl_Input_Choice, 587
update_prev_next
  Fl_Tree_Item, 952

Generated by Doxygen
INDEX

update_v_scrollbar
    Fl_Text_Display, 853
USE_FILTER_EXT
    Fl_Native_File_Chooser, 636
use_high_res_GL
    Fl, 374
USER
    Fl_Preferences, 674
user_data
    Fl_Widget, 1011, 1012
USER_L
    Fl_Preferences, 674
userdeicon
    Fl_Tree_Item, 952
    Fl_Tree_Prefs, 962
USERFLAG1
    Fl_Widget, 985
USERFLAG2
    Fl_Widget, 984
USERFLAG3
    Fl_Widget, 984
usericon
    Fl_Tree, 930
    Fl_Tree_Item, 953
valid
    Fl_Gl_Window, 517
value
    Fl_Browser, 405
    Fl_Button, 426
    Fl_Choice, 443, 444
    Fl_Clock_Output, 450
    Fl_Color_Chooser, 456
    Fl_File_Chooser, 480
    Fl_File_Input, 488, 489
    Fl_Help_Dialog, 535
    Fl_Help_View, 544
    Fl_Input_, 581, 582
    Fl_Input_Choice, 588
    Fl_Menu_, 610
    Fl_Menu_Item, 628
    Fl_Progress, 697
    Fl_Scrollbar, 722, 723
    Fl_Spinner, 752
    Fl_Tabs, 800
    Fl_Valuator, 966
version
    Fl, 374
VERTICAL
    Fl_Browser_, 409
    Fl_Flex, 492
VERTICAL_ALWAYS
    Fl_Browser_, 409
visible
    Fl_Browser, 405
    Fl_Widget, 1012
visible_cells
    Fl_Table, 790
VISIBLE_FOCUSS
    Fl_Widget, 984
visible_focus
    Fl, 375
    Fl_Widget, 1012
visible_r
    Fl_Tree_Item, 953
    Fl_Widget, 1013
visual
    Fl, 375
vline_length
    Fl_Text_Display, 853
vposition
    Fl_Tree, 930, 931
vprintf
    Fl_Simple_Terminal, 744
    Fl_Text_Buffer, 815
vsnprintf.c, 1114
    fl_vsnprintf, 1114
w
    Fl_Image, 557
    Fl_Widget, 1013
wait
    Fl, 375
wait_for_expose
    Fl_Window, 1039
warning
    Common Dialog Classes and Functions, 343
watch_widget_pointer
    Safe widget deletion support functions, 307
wayland.H, 1115
when
    Fl_Table, 790
    Fl_Widget, 1013
which
    Fl_Tabs, 800
widget
    Fl_Widget_Tracker, 1019
win32.H, 1116
window
    Fl_Widget, 1014
window_menu_style
    Fl_Sys_Menu_Bar, 772
window_menu_style_enum
    Fl_Sys_Menu_Bar, 767
Windows handling functions, 240
atclose, 242
default_atclose, 241
first_window, 241
grab, 241
modal, 241
next_window, 242
set_atclose, 242
word_end
    Fl_Input_, 583
    Fl_Text_Buffer, 816
    Fl_Text_Display, 853
word_start
    Fl_Input_, 583