<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.7.13 replall_cb()</td>
<td>48</td>
</tr>
<tr>
<td>8.7.14 replcan_cb()</td>
<td>48</td>
</tr>
<tr>
<td>8.7.15 save_cb()</td>
<td>48</td>
</tr>
<tr>
<td>8.7.16 saveas_cb()</td>
<td>48</td>
</tr>
<tr>
<td>8.8 Other Functions</td>
<td>49</td>
</tr>
<tr>
<td>8.8.1 check_save()</td>
<td>49</td>
</tr>
<tr>
<td>8.8.2 load_file()</td>
<td>49</td>
</tr>
<tr>
<td>8.8.3 save_file()</td>
<td>49</td>
</tr>
<tr>
<td>8.8.4 set_title()</td>
<td>49</td>
</tr>
<tr>
<td>8.9 The main() Function</td>
<td>50</td>
</tr>
<tr>
<td>8.10 Compiling the Editor</td>
<td>50</td>
</tr>
<tr>
<td>8.11 The Final Product</td>
<td>50</td>
</tr>
<tr>
<td>8.12 Advanced Features</td>
<td>51</td>
</tr>
<tr>
<td>8.12.1 Syntax Highlighting</td>
<td>51</td>
</tr>
<tr>
<td>9 Drawing Things in FLTK</td>
<td>55</td>
</tr>
<tr>
<td>9.1 When Can You Draw Things in FLTK?</td>
<td>55</td>
</tr>
<tr>
<td>9.2 What Drawing Units Do FLTK Drawing Functions Use?</td>
<td>56</td>
</tr>
<tr>
<td>9.3 Drawing Functions</td>
<td>57</td>
</tr>
<tr>
<td>9.3.1 Boxes</td>
<td>57</td>
</tr>
<tr>
<td>9.3.2 Clipping</td>
<td>58</td>
</tr>
<tr>
<td>9.3.3 Colors</td>
<td>59</td>
</tr>
<tr>
<td>9.3.4 Line Dashes and Thickness</td>
<td>61</td>
</tr>
<tr>
<td>9.3.5 Drawing Fast Shapes</td>
<td>62</td>
</tr>
<tr>
<td>9.3.6 Drawing Complex Shapes</td>
<td>65</td>
</tr>
<tr>
<td>9.3.7 Drawing Text</td>
<td>67</td>
</tr>
<tr>
<td>9.3.8 Fonts</td>
<td>70</td>
</tr>
<tr>
<td>9.3.9 Character Encoding</td>
<td>70</td>
</tr>
<tr>
<td>9.3.10 Drawing Overlays</td>
<td>71</td>
</tr>
<tr>
<td>9.4 Drawing Images</td>
<td>71</td>
</tr>
<tr>
<td>9.4.1 Direct Image Drawing</td>
<td>71</td>
</tr>
<tr>
<td>9.4.2 Direct Image Reading</td>
<td>73</td>
</tr>
<tr>
<td>9.4.3 Image Classes</td>
<td>73</td>
</tr>
<tr>
<td>9.5 Offscreen Drawing</td>
<td>75</td>
</tr>
<tr>
<td>10 Handling Events</td>
<td>77</td>
</tr>
<tr>
<td>10.1 The FLTK Event Model</td>
<td>77</td>
</tr>
<tr>
<td>10.2 Mouse Events</td>
<td>77</td>
</tr>
<tr>
<td>10.2.1 FL_PUSH</td>
<td>77</td>
</tr>
<tr>
<td>10.2.2 FL_DRAG</td>
<td>77</td>
</tr>
<tr>
<td>10.2.3 FL_RELEASE</td>
<td>78</td>
</tr>
<tr>
<td>10.2.4 FL_MOVE</td>
<td>78</td>
</tr>
<tr>
<td>10.2.5 FL_MOUSEWHEEL</td>
<td>78</td>
</tr>
</tbody>
</table>
21.4 New FL_OVERRIDE Macro .......................... 194
21.5 Fl_Image::copy() 'const' ......................... 195

22 Software License ....................................... 197

23 Example Source Code ................................... 203

23.1 Example Applications: Overview .......................... 203
  23.1.1 adjuster ........................................ 203
  23.1.2 animated ........................................ 203
  23.1.3 arc ............................................. 203
  23.1.4 ask ............................................. 204
  23.1.5 bitmap ........................................ 204
  23.1.6 blocks ......................................... 204
  23.1.7 boxtype ........................................ 204
  23.1.8 browser ........................................ 204
  23.1.9 button ......................................... 204
  23.1.10 buttons ...................................... 204
  23.1.11 cairo_test .................................... 204
  23.1.12 checkers ...................................... 204
  23.1.13 clock ......................................... 205
  23.1.14 colbrowser .................................. 205
  23.1.15 color_chooser ................................ 205
  23.1.16 cube ......................................... 205
  23.1.17 CubeView .................................... 205
  23.1.18 cursor ........................................ 205
  23.1.19 curve ......................................... 205
  23.1.20 demo .......................................... 205
  23.1.21 device ........................................ 205
  23.1.22 doublebuffer ................................ 206
  23.1.23 editor ........................................ 206
  23.1.24 fast_slow ..................................... 206
  23.1.25 file_chooser ................................ 206
  23.1.26 fonts .......................................... 206
  23.1.27 forms ......................................... 206
  23.1.28 fractals ...................................... 206
  23.1.29 fullscreen .................................... 206
  23.1.30 gl_overlay ................................... 206
  23.1.31 glpuzzle ..................................... 206
  23.1.32 hello ......................................... 207
  23.1.33 help_dialog .................................. 207
  23.1.34 icon .......................................... 207
  23.1.35 iconize ....................................... 207
  23.1.36 image ......................................... 207
23.1.37 inactive ................................................................. 207
23.1.38 input ................................................................. 207
23.1.39 input_choice ...................................................... 207
23.1.40 keyboard ........................................................... 207
23.1.41 label ................................................................. 208
23.1.42 line_style ........................................................... 208
23.1.43 list_visuals ......................................................... 208
23.1.44 mandelbrot ......................................................... 208
23.1.45 menubar ............................................................ 208
23.1.46 message ............................................................. 208
23.1.47 minimum ........................................................... 208
23.1.48 native-filechooser ............................................... 208
23.1.49 navigation .......................................................... 208
23.1.50 offscreen ........................................................... 208
23.1.51 output ............................................................... 209
23.1.52 overlay .............................................................. 209
23.1.53 pack ................................................................. 209
23.1.54 pixmap .............................................................. 209
23.1.55 pixmap_browser ................................................... 209
23.1.56 preferences ........................................................ 209
23.1.57 radio ................................................................. 209
23.1.58 resizebox .......................................................... 209
23.1.59 rotated_text ....................................................... 209
23.1.60 resize .............................................................. 209
23.1.61 scroll ............................................................... 210
23.1.62 shape ............................................................... 210
23.1.63 subwindow ........................................................ 210
23.1.64 sudoku ............................................................. 210
23.1.65 symbols ............................................................ 210
23.1.66 table ............................................................... 210
23.1.67 tabs ................................................................. 210
23.1.68 threads ............................................................. 210
23.1.69 tile ................................................................. 210
23.1.70 tiled_image ......................................................... 211
23.1.71 tree ................................................................. 211
23.1.72 twowin ............................................................. 211
23.1.73 unittests .......................................................... 211
23.1.74 utf8 ................................................................. 211
23.1.75 valuators .......................................................... 211
23.1.76 windowfocus ...................................................... 211
23.1.77 fluid ............................................................... 211

23.2 Example Applications: Images ........................................... 211
32.3.2.3 belowmouse() [1/2] ................................. 278
32.3.2.4 belowmouse() [2/2] ................................. 278
32.3.2.5 callback_reason() ........................................ 278
32.3.2.6 compose() ........................................... 278
32.3.2.7 compose_reset() ........................................ 279
32.3.2.8 disable_im() .......................................... 279
32.3.2.9 enable_im() ........................................... 279
32.3.2.10 event() ............................................... 279
32.3.2.11 event_button() ........................................ 279
32.3.2.12 event_button1() ....................................... 280
32.3.2.13 event_button2() ....................................... 280
32.3.2.14 event_button3() ....................................... 280
32.3.2.15 event_buttons() ....................................... 280
32.3.2.16 event_clicks() [1/2] ................................. 280
32.3.2.17 event_clicks() [2/2] ................................. 280
32.3.2.18 event_clipboard() ..................................... 281
32.3.2.19 event_clipboard_type() .............................. 281
32.3.2.20 event_dispatch() ..................................... 281
32.3.2.21 event_dx() .......................................... 282
32.3.2.22 event_dy() .......................................... 282
32.3.2.23 event_inside() [1/2] ............................... 282
32.3.2.24 event_inside() [2/2] ............................... 282
32.3.2.25 event_is_click() [1/2] .............................. 283
32.3.2.26 event_is_click() [2/2] .............................. 283
32.3.2.27 event_key() [1/2] .................................... 283
32.3.2.28 event_key() [2/2] .................................... 283
32.3.2.29 event_length() ....................................... 284
32.3.2.30 event_original_key() ................................. 284
32.3.2.31 event_state() [1/2] ................................. 284
32.3.2.32 event_state() [2/2] ................................. 284
32.3.2.33 event_text() .......................................... 285
32.3.2.34 event_x_root() ....................................... 285
32.3.2.35 event_y_root() ....................................... 285
32.3.2.36 focus() [1/2] ......................................... 285
32.3.2.37 focus() [2/2] ......................................... 285
32.3.2.38 get_key() ........................................... 286
32.3.2.39 get_mouse() .......................................... 286
32.3.2.40 handle() ........................................... 286
32.3.2.41 handle_() ........................................... 286
32.3.2.42 pushed() [1/2] ....................................... 287
32.3.2.43 pushed() [2/2] ....................................... 287
32.3.2.44 remove_handler() .................................... 287
32.7.3.3 fl_arc() [1/2]  .................................................. 320
32.7.3.4 fl_arc() [2/2]  .................................................. 321
32.7.3.5 fl_begin_complex_polygon()  .................................. 322
32.7.3.6 fl_begin_offscreen() ........................................... 322
32.7.3.7 fl_begin_points() ............................................... 322
32.7.3.8 fl_can_do_alpha_blending() .................................... 322
32.7.3.9 fl_capture_window() ............................................ 323
32.7.3.10 fl_circle() ..................................................... 323
32.7.3.11 fl_clip() ....................................................... 323
32.7.3.12 fl_clip_box() .................................................. 324
32.7.3.13 fl_clip_region() [1/2] ........................................ 324
32.7.3.14 fl_clip_region() [2/2] ........................................ 325
32.7.3.15 fl_copy_offscreen() .......................................... 325
32.7.3.16 fl_create_offscreen() ........................................ 325
32.7.3.17 fl_cursor() ..................................................... 326
32.7.3.18 fl_curve() ...................................................... 326
32.7.3.19 fl_delete_offscreen() ........................................ 326
32.7.3.20 fl_draw() [1/4] ................................................ 327
32.7.3.21 fl_draw() [2/4] ................................................ 327
32.7.3.22 fl_draw() [3/4] ................................................ 327
32.7.3.23 fl_draw() [4/4] ................................................ 327
32.7.3.24 fl_draw_arrow() .............................................. 328
32.7.3.25 fl_draw_box() .................................................. 328
32.7.3.26 fl_draw_check() .............................................. 328
32.7.3.27 fl_draw_circle() ................................................ 329
32.7.3.28 fl_draw_image() [1/2] ........................................ 329
32.7.3.29 fl_draw_image() [2/2] ........................................ 330
32.7.3.30 fl_draw_image_mono() [1/2] .................................. 330
32.7.3.31 fl_draw_image_mono() [2/2] .................................. 331
32.7.3.32 fl_draw_pixmap() [1/2] ....................................... 331
32.7.3.33 fl_draw_pixmap() [2/2] ....................................... 331
32.7.3.34 fl_draw_symbol() ............................................ 332
32.7.3.35 fl_expand_text() ............................................. 332
32.7.3.36 fl_focus_rect() .............................................. 332
32.7.3.37 fl_frame() ...................................................... 332
32.7.3.38 fl_frame2() .................................................... 333
32.7.3.39 fl_gap() ......................................................... 333
32.7.3.40 fl_line_style() ............................................... 333
32.7.3.41 fl_load_matrix() ............................................. 334
32.7.3.42 fl_measure() .................................................. 334
32.7.3.43 fl_measure_pixmap() [1/2] .................................. 335
32.7.3.44 fl_measure_pixmap() [2/2] .................................. 335
32.7.3.45 fl_mult_matrix() ........................................... 335
32.7.3.46 fl_not_clipped() ........................................... 336
32.7.3.47 fl_oldShortcut() ............................................ 336
32.7.3.48 fl_overlay_clear() ........................................... 337
32.7.3.49 fl_overlay_rect() ........................................... 337
32.7.3.50 fl_override_scale() ......................................... 338
32.7.3.51 fl_pie() ................................................... 338
32.7.3.52 fl_polygon() [1/2] ......................................... 339
32.7.3.53 fl_polygon() [2/2] ......................................... 339
32.7.3.54 fl_pop_clip() ................................................ 339
32.7.3.55 fl_push_clip() ............................................... 339
32.7.3.56 fl_push_matrix() ........................................... 340
32.7.3.57 fl_read_image() ............................................ 340
32.7.3.58 fl_rect() [1/2] .............................................. 340
32.7.3.59 fl_rect() [2/2] .............................................. 340
32.7.3.60 fl_rectf() [1/2] ............................................ 341
32.7.3.61 fl_rectf() [2/2] ............................................ 341
32.7.3.62 fl_rescale_offscreen() .................................... 341
32.7.3.63 fl_reset_spot() ............................................. 341
32.7.3.64 fl_restore_scale() ......................................... 341
32.7.3.65 fl_rotate() .................................................. 342
32.7.3.66 fl_rounded_rect() .......................................... 342
32.7.3.67 fl_rounded_rectf() ......................................... 342
32.7.3.68 fl_scale() [1/2] ............................................ 342
32.7.3.69 fl_scale() [2/2] ............................................ 342
32.7.3.70 fl_scroll() .................................................. 343
32.7.3.71 fl_set_spot() ............................................... 343
32.7.3.72 fl_set_status() ............................................. 343
32.7.3.73 fl_shortcut_label() [1/2] ................................. 344
32.7.3.74 fl_shortcut_label() [2/2] ................................. 344
32.7.3.75 fl_transform_dx() ......................................... 345
32.7.3.76 fl_transform_dy() ......................................... 345
32.7.3.77 fl_transform_x() .......................................... 345
32.7.3.78 fl_transform_y() .......................................... 346
32.7.3.79 fl_transformed_vertex() .................................. 346
32.7.3.80 fl_translate() ............................................. 346
32.7.3.81 fl_vertex() .................................................. 346

32.8 Multithreading support functions ........................................ 346
32.8.1 Detailed Description ........................................... 347
32.8.2 Function Documentation ......................................... 347
32.8.2.1 awake() [1/2] .............................................. 347
32.8.2.2 awake() [2/2] .............................................. 347
32.11.3.17 fl_stat() ........................................... 363
32.11.3.18 fl_system() ....................................... 363
32.11.3.19 fl_uca_to_Utf16() ................................. 363
32.11.3.20 fl_unlink() ....................................... 364
32.11.3.21 fl_utf8back() .................................... 364
32.11.3.22 fl_utf8bytes() ..................................... 364
32.11.3.23 fl_utf8decode() .................................... 364
32.11.3.24 fl_utf8encode() .................................... 365
32.11.3.25 fl_utf8from_mb() .................................. 365
32.11.3.26 fl_utf8fromma() ................................... 365
32.11.3.27 fl_utf8fromwc() ................................... 366
32.11.3.28 fl_utf8fwd() ....................................... 366
32.11.3.29 fl_utf8len() ....................................... 366
32.11.3.30 fl_utf8len1() ...................................... 366
32.11.3.31 fl_utf8locale() ..................................... 367
32.11.3.32 fl_utf8strlen() ...................................... 367
32.11.3.33 fl_utf8test() ....................................... 367
32.11.3.34 fl_utf8to_mb() ...................................... 368
32.11.3.35 fl_utf8toa() ....................................... 368
32.11.3.36 fl_utf8toUtf16() ................................... 368
32.11.3.37 fl_utf8towc() ..................................... 368
32.11.3.38 fl_utf_nb_char() ................................... 369
32.11.3.39 fl_utf_strcasecmp() ................................ 369
32.11.3.40 fl_utf_strncasecmp() .............................. 369
32.11.3.41 fl_utf_tolower() ................................... 370
32.11.3.42 fl_utf_toupper() ................................... 370
32.11.3.43 fl_wcwidth() ...................................... 370
32.11.3.44 fl_wcwidth_() ..................................... 370

32.12 Mac OS X-specific symbols .................................. 371
32.12.1 Detailed Description .................................. 371
32.12.2 Function Documentation ................................ 371
  32.12.2.1 fl_mac_set_about() .................................. 372
  32.12.2.2 fl_open_callback() .................................. 372
32.12.3 Variable Documentation ................................ 372
  32.12.3.1 fl_mac_os_version .................................. 372

32.13 Common Dialog Classes and Functions ...................... 372
32.13.1 Detailed Description .................................. 374
32.13.2 Function Documentation ................................ 374
  32.13.2.1 fl_alert() ........................................... 374
  32.13.2.2 fl_ask() ........................................... 374
  32.13.2.3 fl_beep() ........................................... 375
  32.13.2.4 fl_choice() ......................................... 375
32.13.2.5 fl_choice_n() ........................................... 377
32.13.2.6 fl_color_chooser() [1/2] .................................. 377
32.13.2.7 fl_color_chooser() [2/2] .................................. 378
32.13.2.8 fl_dir_chooser() ........................................... 379
32.13.2.9 fl_file_chooser() ......................................... 380
32.13.2.10 fl_file_chooser_callback() .............................. 381
32.13.2.11 fl_file_chooser_ok_label() .............................. 381
32.13.2.12 fl_input() .............................................. 381
32.13.2.13 fl_input_str() [1/2] ...................................... 381
32.13.2.14 fl_input_str() [2/2] ...................................... 382
32.13.2.15 fl_message() ........................................... 382
32.13.2.16 fl_message_hotspot() [1/2] .............................. 383
32.13.2.17 fl_message_hotspot() [2/2] .............................. 383
32.13.2.18 fl_message_icon() ....................................... 383
32.13.2.19 fl_message_icon_label() ................................. 384
32.13.2.20 fl_message_position() [1/3] ............................. 384
32.13.2.21 fl_message_position() [2/3] ............................. 385
32.13.2.22 fl_message_position() [3/3] ............................. 385
32.13.2.23 fl_message_title() ....................................... 386
32.13.2.24 fl_message_title_default() .............................. 386
32.13.2.25 fl_password() ........................................... 386
32.13.2.26 fl_password_str() [1/2] .................................. 387
32.13.2.27 fl_password_str() [2/2] .................................. 387

32.13.3 Variable Documentation .................................... 388
32.13.3.1 error ..................................................... 388
32.13.3.2 fatal ..................................................... 388
32.13.3.3 warning .................................................. 388

32.14 File names and URI utility functions .......................... 388
32.14.1 Detailed Description ........................................ 389
32.14.2 Typedef Documentation ..................................... 389
32.14.2.1 Fl_File_Sort_F ........................................ 389
32.14.3 Function Documentation .................................... 389
32.14.3.1 fl_decode_uri() ......................................... 390
32.14.3.2 fl_filename_absolute() .................................. 390
32.14.3.3 fl_filename_expand() .................................... 390
32.14.3.4 fl_filename_ext() ....................................... 391
32.14.3.5 fl_filename_free_list() .................................. 391
32.14.3.6 fl_filename_isdir() ...................................... 391
32.14.3.7 fl_filename_list() ....................................... 392
32.14.3.8 fl_filename_match() ...................................... 392
32.14.3.9 fl_filename_name() ...................................... 393
32.14.3.10 fl_filename_relative() ................................. 393
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.3.28</td>
<td>draw_GL_text_with_textures() [1/2]</td>
</tr>
<tr>
<td>33.3.29</td>
<td>draw_GL_text_with_textures() [2/2]</td>
</tr>
<tr>
<td>33.3.30</td>
<td>flush()</td>
</tr>
<tr>
<td>33.3.31</td>
<td>get_system_colors()</td>
</tr>
<tr>
<td>33.3.32</td>
<td>gl_visual()</td>
</tr>
<tr>
<td>33.3.33</td>
<td>has_timeout()</td>
</tr>
<tr>
<td>33.3.34</td>
<td>hide_all_windows()</td>
</tr>
<tr>
<td>33.3.35</td>
<td>is_scheme()</td>
</tr>
<tr>
<td>33.3.36</td>
<td>menu_linespacing() [1/2]</td>
</tr>
<tr>
<td>33.3.37</td>
<td>menu_linespacing() [2/2]</td>
</tr>
<tr>
<td>33.3.38</td>
<td>now()</td>
</tr>
<tr>
<td>33.3.39</td>
<td>option() [1/2]</td>
</tr>
<tr>
<td>33.3.40</td>
<td>option() [2/2]</td>
</tr>
<tr>
<td>33.3.41</td>
<td>own_colormap()</td>
</tr>
<tr>
<td>33.3.42</td>
<td>program.Should.Quit() [1/2]</td>
</tr>
<tr>
<td>33.3.43</td>
<td>program.Should.Quit() [2/2]</td>
</tr>
<tr>
<td>33.3.44</td>
<td>readqueue()</td>
</tr>
<tr>
<td>33.3.45</td>
<td>ready()</td>
</tr>
<tr>
<td>33.3.46</td>
<td>release()</td>
</tr>
<tr>
<td>33.3.47</td>
<td>reload_scheme()</td>
</tr>
<tr>
<td>33.3.48</td>
<td>remove_check()</td>
</tr>
<tr>
<td>33.3.49</td>
<td>remove_timeout()</td>
</tr>
<tr>
<td>33.3.50</td>
<td>repeat_timeout()</td>
</tr>
<tr>
<td>33.3.51</td>
<td>run()</td>
</tr>
<tr>
<td>33.3.52</td>
<td>scheme()</td>
</tr>
<tr>
<td>33.3.53</td>
<td>scrollbar_size() [1/2]</td>
</tr>
<tr>
<td>33.3.54</td>
<td>scrollbar_size() [2/2]</td>
</tr>
<tr>
<td>33.3.55</td>
<td>seconds_between()</td>
</tr>
<tr>
<td>33.3.56</td>
<td>seconds_since()</td>
</tr>
<tr>
<td>33.3.57</td>
<td>set_box_color()</td>
</tr>
<tr>
<td>33.3.58</td>
<td>set_idle()</td>
</tr>
<tr>
<td>33.3.59</td>
<td>ticks_between()</td>
</tr>
<tr>
<td>33.3.60</td>
<td>ticks_since()</td>
</tr>
<tr>
<td>33.3.61</td>
<td>use_high_res_GL() [1/2]</td>
</tr>
<tr>
<td>33.3.62</td>
<td>use_high_res_GL() [2/2]</td>
</tr>
<tr>
<td>33.3.63</td>
<td>version()</td>
</tr>
<tr>
<td>33.3.64</td>
<td>visible_focus() [1/2]</td>
</tr>
<tr>
<td>33.3.65</td>
<td>visible_focus() [2/2]</td>
</tr>
<tr>
<td>33.3.66</td>
<td>visual()</td>
</tr>
<tr>
<td>33.3.67</td>
<td>wait() [1/2]</td>
</tr>
<tr>
<td>33.3.68</td>
<td>wait() [2/2]</td>
</tr>
</tbody>
</table>

33.3.4 Member Data Documentation
33.3.4.1 help ................................................................. 426
33.3.4.2 idle ............................................................... 426

33.4 Fl_Adjuster Class Reference ................................................. 427
33.4.1 Detailed Description ....................................................... 427
33.4.2 Constructor & Destructor Documentation ................................ 428
  33.4.2.1 Fl_Adjuster() .................................................... 428
33.4.3 Member Function Documentation ....................................... 428
  33.4.3.1 draw() ......................................................... 428
  33.4.3.2 handle() ...................................................... 428
  33.4.3.3 soft() [1/2] ................................................... 429
  33.4.3.4 soft() [2/2] ................................................... 429
  33.4.3.5 value_damage() ............................................... 429

33.5 Fl_Anim_GIF_Image Class Reference ...................................... 429
33.5.1 Detailed Description ..................................................... 432
33.5.2 Member Enumeration Documentation .................................... 432
  33.5.2.1 Flags .......................................................... 432
33.5.3 Constructor & Destructor Documentation ................................ 432
  33.5.3.1 Fl_Anim_GIF_Image() [1/2] .................................... 432
  33.5.3.2 Fl_Anim_GIF_Image() [2/2] .................................... 433
  33.5.3.3 ~Fl_Anim_GIF_Image() .......................................... 433
33.5.4 Member Function Documentation ....................................... 433
  33.5.4.1 canvas() [1/2] ................................................ 433
  33.5.4.2 canvas() [2/2] ................................................ 434
  33.5.4.3 canvas_h() .................................................... 434
  33.5.4.4 canvas_w() .................................................... 434
  33.5.4.5 color_average() ............................................... 434
  33.5.4.6 copy() ........................................................ 435
  33.5.4.7 delay() [1/2] .................................................. 435
  33.5.4.8 delay() [2/2] .................................................. 435
  33.5.4.9 desaturate() ................................................... 435
  33.5.4.10 draw() ........................................................ 435
  33.5.4.11 frame() [1/2] ................................................ 436
  33.5.4.12 frame() [2/2] ................................................ 436
  33.5.4.13 frame_count() ............................................... 436
  33.5.4.14 frame_h() .................................................... 437
  33.5.4.15 frame_uncache() [1/2] ..................................... 437
  33.5.4.16 frame_uncache() [2/2] ..................................... 437
  33.5.4.17 frame_w() .................................................... 437
  33.5.4.18 frame_x() .................................................... 438
  33.5.4.19 frame_y() .................................................... 438
  33.5.4.20 frames() ...................................................... 438
  33.5.4.21 image() [1/2] ............................................... 438
33.5.4.22 image() [2/2] .................................................. 439
33.5.4.23 is_animated() ................................................. 439
33.5.4.24 load() ............................................................. 439
33.5.4.25 name() ............................................................ 439
33.5.4.26 next() ............................................................. 440
33.5.4.27 on_extension_data() ........................................... 440
33.5.4.28 on_frame_data() .............................................. 440
33.5.4.29 playing() ....................................................... 440
33.5.4.30 resize() [1/2] ................................................... 440
33.5.4.31 resize() [2/2] ................................................... 440
33.5.4.32 speed() [1/2] ................................................... 441
33.5.4.33 speed() [2/2] ................................................... 441
33.5.4.34 start() ........................................................... 441
33.5.4.35 stop() ............................................................ 441
33.5.4.36 uncache() ....................................................... 441
33.5.4.37 valid() ........................................................... 441

33.5.5 Member Data Documentation ....................................... 442
33.5.5.1 loop ............................................................ 442
33.5.5.2 min_delay ......................................................... 442

33.6 Fl_Bitmap Class Reference ........................................... 442
33.6.1 Detailed Description .............................................. 443
33.6.2 Constructor & Destructor Documentation ......................... 443
33.6.2.1 Fl_Bitmap() [1/4] .............................................. 443
33.6.2.2 Fl_Bitmap() [2/4] .............................................. 443
33.6.2.3 Fl_Bitmap() [3/4] .............................................. 444
33.6.2.4 Fl_Bitmap() [4/4] .............................................. 444
33.6.3 Member Function Documentation .................................. 444
33.6.3.1 copy() ........................................................... 444
33.6.3.2 draw() ........................................................... 445
33.6.3.3 label() [1/2] ..................................................... 445
33.6.3.4 label() [2/2] ..................................................... 445
33.6.3.5 uncache() ......................................................... 446

33.7 Fl_BMP_Image Class Reference ...................................... 446
33.7.1 Detailed Description .............................................. 446
33.7.2 Constructor & Destructor Documentation ......................... 446
33.7.2.1 Fl_BMP_Image() [1/2] ........................................... 447
33.7.2.2 Fl_BMP_Image() [2/2] ........................................... 447

33.8 Fl_Box Class Reference ............................................... 448
33.8.1 Detailed Description .............................................. 448
33.8.2 Constructor & Destructor Documentation ......................... 448
33.8.2.1 Fl_Box() ........................................................ 448
33.8.3 Member Function Documentation .................................. 449
33.9.3.36 lineno() ................................................................. 465
33.9.3.37 lineposition() ....................................................... 466
33.9.3.38 load() .................................................................. 466
33.9.3.39 make_visible() ........................................................ 466
33.9.3.40 middline() ............................................................... 467
33.9.3.41 move() .................................................................. 467
33.9.3.42 remove() .................................................................. 467
33.9.3.43 remove_icon() .......................................................... 468
33.9.3.44 select() .................................................................. 468
33.9.3.45 selected() ................................................................. 468
33.9.3.46 show() [1/2] ............................................................... 469
33.9.3.47 show() [2/2] ............................................................... 469
33.9.3.48 size() ..................................................................... 469
33.9.3.49 swap() [1/2] ............................................................... 469
33.9.3.50 swap() [2/2] ............................................................... 469
33.9.3.51 text() [1/2] ................................................................. 470
33.9.3.52 text() [2/2] ................................................................. 470
33.9.3.53 textsize() ................................................................. 470
33.9.3.54 topline() [1/2] ......................................................... 470
33.9.3.55 topline() [2/2] ......................................................... 471
33.9.3.56 value() [1/2] .............................................................. 471
33.9.3.57 value() [2/2] .............................................................. 471
33.9.3.58 visible() ................................................................. 471

33.10 Fl_Browser_ Class Reference ........................................ 472
    33.10.1 Detailed Description .................................................. 475
    33.10.1.1 Keyboard navigation of browser items ......................... 475
    33.10.2 Member Enumeration Documentation ............................... 475
    33.10.2.1 anonymous enum ............................................... 475
    33.10.3 Constructor & Destructor Documentation .......................... 476
    33.10.3.1 Fl_Browser_() ....................................................... 476
    33.10.4 Member Function Documentation .................................. 476
    33.10.4.1 bbox() ................................................................. 476
    33.10.4.2 deleting() ............................................................. 477
    33.10.4.3 deselect() ............................................................. 477
    33.10.4.4 display() ............................................................. 477
    33.10.4.5 displayed() .......................................................... 477
    33.10.4.6 draw() ................................................................. 478
    33.10.4.7 find_item() ............................................................ 478
    33.10.4.8 full_height() ......................................................... 478
    33.10.4.9 full_width() .......................................................... 478
    33.10.4.10 handle() ............................................................. 479
    33.10.4.11 has_scrollbar() .................................................... 479
33.11 Fl_Button Class Reference .................................................. 489
  33.11.1 Detailed Description ..................................................... 490
  33.11.2 Constructor & Destructor Documentation ........................... 490
    33.11.2.1 Fl_Button() .......................................................... 490
  33.11.3 Member Function Documentation ..................................... 491
    33.11.3.1 clear() ................................................................. 491
    33.11.3.2 down_box() ......................................................... 491
      [1/2]
    33.11.3.3 down_box() ......................................................... 491
      [2/2]
    33.11.3.4 draw() .................................................................. 492
    33.11.3.5 handle() .................................................................. 492
    33.11.3.6 set() ...................................................................... 492
    33.11.3.7 shortcut() ............................................................... 492
      [1/2]
    33.11.3.8 shortcut() ............................................................... 493
      [2/2]
    33.11.3.9 value() .................................................................. 493
  33.12 Fl_Cairo_State Class Reference ........................................ 493
    33.12.1 Detailed Description .................................................... 494
    33.12.2 Member Function Documentation ..................................... 494
      33.12.2.1 cc() .................................................................. 494
  33.13 Fl_Cairo_Window Class Reference ....................................... 494
    33.13.1 Detailed Description .................................................... 495
    33.13.2 Member Function Documentation ..................................... 496
      33.13.2.1 draw() ................................................................. 496
      33.13.2.2 set_draw_cb() ....................................................... 496
  33.14 Fl_Chart Class Reference ................................................ 496
    33.14.1 Detailed Description .................................................... 497
    33.14.2 Constructor & Destructor Documentation ........................... 498
      33.14.2.1 Fl_Chart() ............................................................. 498
    33.14.3 Member Function Documentation ..................................... 498
      33.14.3.1 add() ................................................................. 498
      33.14.3.2 autosize() ......................................................... 499
        [1/2]
      33.14.3.3 autosize() ......................................................... 499
        [2/2]
      33.14.3.4 bounds() ........................................................... 499
        [1/2]
      33.14.3.5 bounds() ........................................................... 499
        [2/2]
      33.14.3.6 draw() ............................................................... 500
      33.14.3.7 draw_barchart() .................................................. 500
      33.14.3.8 draw_horbarchart() .............................................. 500
      33.14.3.9 draw_linechart() ................................................ 501
      33.14.3.10 draw_piechart() ................................................ 501
      33.14.3.11 insert() ............................................................ 502
      33.14.3.12 maxsize() ........................................................ 502
      33.14.3.13 replace() ........................................................ 502
      33.14.3.14 size() ............................................................. 503
33.15 FL_CHART_ENTRY Struct Reference ............................................. 503
   33.15.1 Detailed Description ...................................................... 503
33.16 Fl_Check_Browser Class Reference .......................................... 503
   33.16.1 Detailed Description ..................................................... 505
   33.16.2 Member Function Documentation
      33.16.2.1 add() [1/2] ..................................................... 505
      33.16.2.2 add() [2/2] ..................................................... 505
      33.16.2.3 handle() ....................................................... 506
      33.16.2.4 item_at() ..................................................... 506
      33.16.2.5 item_draw() .................................................. 506
      33.16.2.6 item_first() .................................................. 506
      33.16.2.7 item_height() ................................................. 507
      33.16.2.8 item_next() .................................................. 507
      33.16.2.9 item_prev() .................................................. 507
      33.16.2.10 item_select() ............................................... 508
      33.16.2.11 item_selected() ............................................. 508
      33.16.2.12 item_swap() ................................................ 508
      33.16.2.13 item_text() ................................................ 508
      33.16.2.14 item_width() .............................................. 508
      33.16.2.15 nitems() ..................................................... 509
      33.16.2.16 remove() ..................................................... 509
33.17 Fl_Check_Button Class Reference ............................................ 509
   33.17.1 Detailed Description .................................................... 509
   33.17.2 Constructor & Destructor Documentation
      33.17.2.1 Fl_Check_Button() .......................................... 510
33.18 Fl_Choice Class Reference .................................................... 510
   33.18.1 Detailed Description .................................................. 511
   33.18.2 Constructor & Destructor Documentation
      33.18.2.1 Fl_Choice() .................................................. 512
   33.18.3 Member Function Documentation
      33.18.3.1 draw() ....................................................... 512
      33.18.3.2 handle() ..................................................... 512
      33.18.3.3 value() [1/3] ............................................... 513
      33.18.3.4 value() [2/3] ............................................... 513
      33.18.3.5 value() [3/3] ............................................... 513
33.19 Fl_Clock Class Reference ..................................................... 514
   33.19.1 Detailed Description .................................................. 514
   33.19.2 Constructor & Destructor Documentation
      33.19.2.1 Fl_Clock() [1/2] ............................................. 515
      33.19.2.2 Fl_Clock() [2/2] ............................................. 515
   33.19.3 Member Function Documentation
      33.19.3.1 handle() ..................................................... 516
33.29 Fl_EPS_File_Surface Class Reference ........................................... 540
33.29.1 Detailed Description ............................................................... 540
33.29.2 Constructor & Destructor Documentation ................................... 541
   33.29.2.1 Fl_EPS_File_Surface() ...................................................... 541
   33.29.2.2 Fl_EPS_File_Surface() ...................................................... 541
33.29.3 Member Function Documentation ............................................. 541
   33.29.3.1 close() ............................................................................ 541
   33.29.3.2 origin() [1/2] .................................................................... 542
   33.29.3.3 origin() [2/2] .................................................................... 542
   33.29.3.4 printable_rect() ................................................................... 542
   33.29.3.5 translate() ......................................................................... 542
   33.29.3.6 untranslate() ....................................................................... 543
33.30 Fl_File_Browser Class Reference ................................................... 543
33.30.1 Detailed Description .................................................................... 544
33.30.2 Constructor & Destructor Documentation ................................... 544
   33.30.2.1 Fl_File_Browser() ............................................................... 544
33.30.3 Member Function Documentation ............................................. 544
   33.30.3.1 errmsg() [1/2] .................................................................... 544
   33.30.3.2 errmsg() [2/2] .................................................................... 544
   33.30.3.3 filetype() [1/2] .................................................................... 544
   33.30.3.4 filetype() [2/2] .................................................................... 544
   33.30.3.5 filter() [1/2] ...................................................................... 545
   33.30.3.6 filter() [2/2] ...................................................................... 545
   33.30.3.7 iconsize() [1/2] ................................................................. 545
   33.30.3.8 iconsize() [2/2] ................................................................. 545
   33.30.3.9 load() ................................................................................ 545
33.31 Fl_File_Chooser Class Reference ................................................... 545
33.31.1 Detailed Description .................................................................... 548
33.31.2 Constructor & Destructor Documentation ................................... 550
   33.31.2.1 Fl_File_Chooser() ............................................................... 550
33.31.3 Member Function Documentation ............................................. 551
   33.31.3.1 add_extra() ....................................................................... 551
   33.31.3.2 filter() .............................................................................. 551
   33.31.3.3 iconsize() [1/2] ................................................................. 551
   33.31.3.4 iconsize() [2/2] ................................................................. 552
   33.31.3.5 preview() .......................................................................... 552
   33.31.3.6 shown() ........................................................................... 552
   33.31.3.7 value() ............................................................................ 552
33.31.4 Member Data Documentation ..................................................... 552
   33.31.4.1 showHiddenButton ............................................................. 552
33.32 Fl_File_Icon Class Reference ............................................................ 552
33.32.1 Detailed Description .................................................................... 554
33.41.3 Member Function Documentation ............................................. 577
   33.41.3.1 draw() .......................................................... 577
   33.41.3.2 handle() ....................................................... 577

33.42 Fl_GIF_Image Class Reference ............................................... 578
   33.42.1 Detailed Description ................................................... 579
   33.42.2 Constructor & Destrctor Documentation ................................. 579
      33.42.2.1 Fl_GIF_Image() [1/3] ........................................ 579
      33.42.2.2 Fl_GIF_Image() [2/3] ........................................ 579
      33.42.2.3 Fl_GIF_Image() [3/3] ........................................ 580
   33.42.3 Member Data Documentation ............................................. 580
      33.42.3.1 animate ..................................................... 581

33.43 Fl_Gl_Choice Class Reference .............................................. 581

33.44 Fl_Gl_Window Class Reference .............................................. 581
   33.44.1 Detailed Description ................................................... 583
   33.44.2 Constructor & Destructor Documentation ................................. 583
      33.44.2.1 Fl_Gl_Window() [1/2] ........................................ 584
      33.44.2.2 Fl_Gl_Window() [2/2] ........................................ 584
   33.44.3 Member Function Documentation ....................................... 584
      33.44.3.1 as_gl_window() [1/2] .......................................... 584
      33.44.3.2 as_gl_window() [2/2] .......................................... 584
      33.44.3.3 can_do() ................................................... 584
      33.44.3.4 can_do_overlay() ........................................... 585
      33.44.3.5 context() [1/2] .............................................. 585
      33.44.3.6 context() [2/2] .............................................. 585
      33.44.3.7 context_valid() ............................................ 585
      33.44.3.8 draw() .................................................... 585
      33.44.3.9 draw_begin() ............................................... 586
      33.44.3.10 draw_end() ............................................... 586
      33.44.3.11 flush() .................................................. 586
      33.44.3.12 handle() ................................................ 586
      33.44.3.13 hide() .................................................. 586
      33.44.3.14 make_current() ........................................... 587
      33.44.3.15 make_overlay_current() .................................... 587
      33.44.3.16 mode() [1/3] .............................................. 587
      33.44.3.17 mode() [2/3] .............................................. 587
      33.44.3.18 mode() [3/3] .............................................. 587
      33.44.3.19 ortho() ................................................ 588
      33.44.3.20 pixel_h() ................................................. 588
      33.44.3.21 pixel_w() ................................................. 588
      33.44.3.22 pixels_per_unit() ....................................... 589
      33.44.3.23 redraw_overlay() ........................................ 589
      33.44.3.24 resize() ................................................ 589
33.65.3.20 ld() [1/2] ................................................. 630
33.65.3.21 ld() [2/2] ................................................. 631
33.65.3.22 release() ............................................. 631
33.65.3.23 RGB_scaling() ....................................... 631
33.65.3.24 scale() .................................................. 631
33.65.3.25 scaling_algorithm() ................................... 632
33.65.3.26 uncache() ............................................. 632
33.65.3.27 w() [1/2] .................................................. 632
33.65.3.28 w() [2/2] .................................................. 632

33.66 Fl_Image_Reader Class Reference ................................................. 632
33.67 Fl_image_Surface Class Reference ................................................. 633

33.67.1 Detailed Description ............................................. 634
33.67.2 Constructor & Destructor Documentation ................................. 634
33.67.2.1 Fl_image_Surface() ....................................... 634

33.67.3 Member Function Documentation ......................................... 635
33.67.3.1 highres_image() ......................................... 635
33.67.3.2 image() .................................................. 635
33.67.3.3 is_current() ............................................. 635
33.67.3.4 offscreen() ............................................. 635
33.67.3.5 origin() [1/2] ............................................. 635
33.67.3.6 origin() [2/2] ............................................. 635
33.67.3.7 printable_rect() ......................................... 636
33.67.3.8 rescale() .................................................. 636
33.67.3.9 set_current() ............................................ 636
33.67.3.10 translate() ............................................. 637
33.67.3.11 untranslate() ........................................... 637

33.68 Fl_input Class Reference ................................................. 637
33.68.1 Detailed Description ............................................. 638
33.68.2 Constructor & Destructor Documentation .................................. 639
33.68.2.1 Fl_input() ................................................. 639

33.68.3 Member Function Documentation ......................................... 639
33.68.3.1 draw() .................................................. 639
33.68.3.2 handle() .................................................. 640
33.68.3.3 handle_key() ............................................. 640

33.69 Fl_input_ Class Reference ................................................. 640
33.69.1 Detailed Description ............................................. 643
33.69.2 Constructor & Destructor Documentation ................................. 644
33.69.2.1 Fl_input_() ................................................. 644
33.69.2.2 ~Fl_input_() .............................................. 644

33.69.3 Member Function Documentation ......................................... 644
33.69.3.1 append() .................................................. 644
33.69.3.2 apply_undo() ............................................. 645
33.69.3 can_redo() ........................................... 645
33.69.3.4 can_undo() ..................................... 645
33.69.3.5 copy() ........................................... 645
33.69.3.6 copy_cuts() ................................... 645
33.69.3.7 cursor_color() [1/2] ......................... 646
33.69.3.8 cursor_color() [2/2] ......................... 646
33.69.3.9 cut() [1/3] .................................. 646
33.69.3.10 cut() [2/3] .................................. 646
33.69.3.11 cut() [3/3] .................................. 647
33.69.3.12 drawtext() [1/2] .............................. 647
33.69.3.13 drawtext() [2/2] .............................. 647
33.69.3.14 handle_mouse() .................................. 648
33.69.3.15 handletext() ................................ 648
33.69.3.16 index() ........................................... 648
33.69.3.17 input_type() [1/2] ........................... 648
33.69.3.18 input_type() [2/2] ........................... 649
33.69.3.19 insert() [1/3] ................................ 649
33.69.3.20 insert_position() [1/3] ....................... 649
33.69.3.21 insert_position() [2/3] ....................... 649
33.69.3.22 insert_position() [3/3] ....................... 650
33.69.3.23 line_end() ..................................... 650
33.69.3.24 line_start() ................................... 650
33.69.3.25 mark() [1/2] .................................. 651
33.69.3.26 mark() [2/2] .................................. 651
33.69.3.27 maximum_size() [1/2] ....................... 651
33.69.3.28 maximum_size() [2/2] ....................... 651
33.69.3.29 position() [1/3] ............................... 652
33.69.3.30 position() [2/3] ............................... 652
33.69.3.31 position() [3/3] ............................... 652
33.69.3.32 readonly() [1/2] ............................. 652
33.69.3.33 readonly() [2/2] ............................. 652
33.69.3.34 redo() ........................................... 652
33.69.3.35 replace() ...................................... 653
33.69.3.36 resize() ....................................... 653
33.69.3.37 shortcut() [1/2] .............................. 654
33.69.3.38 shortcut() [2/2] .............................. 654
33.69.3.39 size() [1/2] ................................... 654
33.69.3.40 size() [2/2] ................................... 654
33.69.3.41 static_value() [1/2] ......................... 655
33.69.3.42 static_value() [2/2] ......................... 655
33.69.3.43 tab_nav() [1/2] ............................... 655
33.69.3.44 tab_nav() [2/2] ............................... 656
33.69.3.45 textcolor() [1/2] .............................................. 656
33.69.3.46 textcolor() [2/2] .............................................. 656
33.69.3.47 textfont() [1/2] .............................................. 657
33.69.3.48 textfont() [2/2] .............................................. 657
33.69.3.49 textsize() [1/2] .............................................. 657
33.69.3.50 textsize() [2/2] .............................................. 657
33.69.3.51 undo() ......................................................... 657
33.69.3.52 up_down_position() ........................................ 657
33.69.3.53 value() [1/3] .................................................. 658
33.69.3.54 value() [2/3] .................................................. 658
33.69.3.55 value() [3/3] .................................................. 658
33.69.3.56 word_end() ................................................... 659
33.69.3.57 word_start() ................................................ 659
33.69.3.58 wrap() [1/2] .................................................. 660
33.69.3.59 wrap() [2/2] .................................................. 660

33.70 Fl_Input_Choice Class Reference .............................................. 660
33.70.1 Detailed Description ................................................. 662
33.70.2 Constructor & Destructor Documentation ................................ 663
33.70.2.1 Fl_Input_Choice() ............................................ 663

33.70.3 Member Function Documentation ...................................... 663
33.70.3.1 add() ......................................................... 663
33.70.3.2 inp_x() ....................................................... 663
33.70.3.3 input() ....................................................... 663
33.70.3.4 menu_x() ..................................................... 664
33.70.3.5 menubutton() ............................................... 664
33.70.3.6 resize() ....................................................... 664
33.70.3.7 update_menubutton() ....................................... 664
33.70.3.8 value() [1/2] .................................................. 665
33.70.3.9 value() [2/2] .................................................. 665

33.71 Fl_Int_Input Class Reference .............................................. 665
33.71.1 Detailed Description ................................................. 665
33.71.2 Constructor & Destructor Documentation ................................ 666
33.71.2.1 Fl_Int_Input() ............................................... 666

33.72 Fl_Int_Vector Class Reference ............................................ 666
33.72.1 Detailed Description ................................................. 667
33.72.2 Member Function Documentation ...................................... 667
33.72.2.1 back() ....................................................... 667
33.72.2.2 empty() ...................................................... 667
33.72.2.3 operator=() ................................................ 667
33.72.2.4 operator[](1/2) .......................................... 667
33.72.2.5 operator[](2/2) .......................................... 668
33.72.2.6 pop_back() .................................................. 668
33.82.2.38 testShortcut() .......................................................... 709
33.82.2.39 uncheck() ............................................................... 709
33.82.2.40 value() ..................................................................... 709

33.83 Fl_Menu_Window Class Reference ........................................ 709
  33.83.1 Detailed Description ......................................................... 710

33.84 Fl_Multi_Browser Class Reference ...................................... 710
  33.84.1 Detailed Description ......................................................... 710
  33.84.2 Constructor & Destructor Documentation .......................... 711
    33.84.2.1 Fl_Multi_Browser() .......................................................... 711

33.85 Fl_Multi_Label Struct Reference ........................................ 711
  33.85.1 Detailed Description ........................................................ 711
  33.85.2 Member Data Documentation ......................................... 712
    33.85.2.1 labela ................................................................. 712
    33.85.2.2 labelb ................................................................. 712
    33.85.2.3 typea ................................................................. 712
    33.85.2.4 typeb ................................................................. 712

33.86 Fl_Multiline_Input Class Reference .................................... 713
  33.86.1 Detailed Description ......................................................... 713
  33.86.2 Constructor & Destructor Documentation .......................... 713
    33.86.2.1 Fl_Multiline_Input() ....................................................... 713

33.87 Fl_Multiline_Output Class Reference .................................. 714
  33.87.1 Detailed Description ......................................................... 714
  33.87.2 Constructor & Destructor Documentation .......................... 714
    33.87.2.1 Fl_Multiline_Output() ...................................................... 714

33.88 Fl_Native_File_Chooser Class Reference ............................. 715
  33.88.1 Detailed Description ......................................................... 715
  33.88.2 Member Enumeration Documentation ................................ 717
    33.88.2.1 Option ...................................................................... 717
    33.88.2.2 Type ....................................................................... 718
  33.88.3 Constructor & Destructor Documentation .......................... 718
    33.88.3.1 Fl_Native_File_Chooser() ................................................. 718
    33.88.3.2 ~Fl_Native_File_Chooser() ............................................ 718
  33.88.4 Member Function Documentation ..................................... 718
    33.88.4.1 count() ................................................................... 718
    33.88.4.2 directory() .............................................................. 718
    33.88.4.3 errmsg() .................................................................. 718
    33.88.4.4 filename() [1/2] .......................................................... 719
    33.88.4.5 filename() [2/2] .......................................................... 719
    33.88.4.6 filter() [1/2] .............................................................. 719
    33.88.4.7 filter() [2/2] .............................................................. 719
    33.88.4.8 filter_value() [1/2] ....................................................... 719
    33.88.4.9 filter_value() [2/2] ....................................................... 720
33.88.4.10 options() ........................................ 720
33.88.4.11 preset_file() ................................... 720
33.88.4.12 show() ........................................... 720
33.88.4.13 title() [1/2] ..................................... 720
33.88.4.14 title() [2/2] ..................................... 720

33.89 Fl_Nice_Slider Class Reference ........................................ 721
33.90 Fl_Output Class Reference ........................................ 721
  33.90.1 Detailed Description ....................................... 721
  33.90.2 Constructor & Destructor Documentation ..................... 722
    33.90.2.1 Fl_Output() ...................................... 722
33.91 Fl_Overlay_Window Class Reference .................................. 722
  33.91.1 Detailed Description ...................................... 723
  33.91.2 Constructor & Destructor Documentation ..................... 724
    33.91.2.1 Fl_Overlay_Window() ................................ 724
  33.91.3 Member Function Documentation ................................ 724
    33.91.3.1 as_overlay_window() ................................ 724
    33.91.3.2 draw_overlay() .................................... 724
    33.91.3.3 flush() ........................................... 724
    33.91.3.4 hide() ............................................ 724
    33.91.3.5 redraw_overlay() .................................. 724
    33.91.3.6 resize() .......................................... 725
    33.91.3.7 show() ............................................ 725
33.92 Fl_Pack Class Reference ........................................ 725
  33.92.1 Detailed Description ....................................... 726
  33.92.2 Constructor & Destructor Documentation ..................... 726
    33.92.2.1 Fl_Pack() .......................................... 726
  33.92.3 Member Function Documentation ................................ 727
    33.92.3.1 draw() ............................................ 727
    33.92.3.2 horizontal() ...................................... 727
    33.92.3.3 resize() .......................................... 727
33.93 Fl_Paged_Device Class Reference ..................................... 728
  33.93.1 Detailed Description ....................................... 729
  33.93.2 Member Enumeration Documentation ............................ 729
    33.93.2.1 Page_Format ....................................... 729
    33.93.2.2 Page_Layout ....................................... 730
  33.93.3 Member Function Documentation ................................ 730
    33.93.3.1 begin_job() ....................................... 730
    33.93.3.2 begin_page() ...................................... 731
    33.93.3.3 end_job() .......................................... 731
    33.93.3.4 end_page() ........................................ 731
    33.93.3.5 margins() ......................................... 732
    33.93.3.6 rotate() .......................................... 732
33.119.1 Detailed Description ............................................. 811
33.119.2 Constructor & Destructor Documentation ......................... 811
   33.119.2.1 Fl_Scrollbar() ........................................ 811
33.119.3 Member Function Documentation ................................... 812
   33.119.3.1 draw() ............................................. 812
   33.119.3.2 handle() ............................................ 812
   33.119.3.3 linesize() ........................................... 812
   33.119.3.4 value() [1/3] ........................................... 812
   33.119.3.5 value() [2/3] ........................................... 813
   33.119.3.6 value() [3/3] ........................................... 813
33.120 Fl_Scroll::Fl_Scrollbar_Data Struct Reference ..................... 813
   33.120.1 Detailed Description ....................................... 814
33.121 Fl_Secret_Input Class Reference .................................... 814
   33.121.1 Detailed Description ....................................... 814
   33.121.2 Constructor & Destructor Documentation ......................... 814
      33.121.2.1 Fl_Secret_Input() .................................... 815
33.121.3 Member Function Documentation ................................... 815
   33.121.3.1 handle() ............................................. 815
33.122 Fl_Select_Browser Class Reference .................................. 815
   33.122.1 Detailed Description ....................................... 816
   33.122.2 Constructor & Destructor Documentation ......................... 816
      33.122.2.1 Fl_Select_Browser() .................................... 816
33.123 Fl_Shared_Image Class Reference .................................. 816
   33.123.1 Detailed Description ....................................... 818
   33.123.2 Constructor & Destructor Documentation ......................... 819
      33.123.2.1 Fl_Shared_Image() [1/2] .................................. 819
      33.123.2.2 Fl_Shared_Image() [2/2] .................................. 819
      33.123.2.3 ~Fl_Shared_Image() ..................................... 819
   33.123.3 Member Function Documentation ................................ 819
      33.123.3.1 add() ............................................. 819
      33.123.3.2 add_handler() ........................................ 819
      33.123.3.3 as_shared_image() ..................................... 819
      33.123.3.4 color_average() ....................................... 820
      33.123.3.5 compare() ............................................. 820
      33.123.3.6 copy() .............................................. 821
      33.123.3.7 desaturate() .......................................... 821
      33.123.3.8 draw() .............................................. 821
      33.123.3.9 find() ............................................... 822
      33.123.3.10 get() [1/2] .......................................... 822
      33.123.3.11 get() [2/2] .......................................... 822
      33.123.3.12 image() ............................................. 823
      33.123.3.13 original() .......................................... 823
33.131.2.2 Fl_String() [2/3] .......................................................... 850
33.131.2.3 Fl_String() [3/3] .......................................................... 851
33.131.3 Member Function Documentation ........................................ 851
  33.131.3.1 append() [1/3] .......................................................... 851
  33.131.3.2 append() [2/3] .......................................................... 851
  33.131.3.3 append() [3/3] .......................................................... 852
  33.131.3.4 assign() [1/3] .......................................................... 852
  33.131.3.5 assign() [2/3] .......................................................... 852
  33.131.3.6 assign() [3/3] .......................................................... 852
  33.131.3.7 at() ................................................................. 853
  33.131.3.8 c_str() ............................................................... 853
  33.131.3.9 capacity() ......................................................... 853
  33.131.3.10 data() .................................................................. 853
  33.131.3.11 data() .................................................................. 854
  33.131.3.12 debug() ............................................................... 854
  33.131.3.13 empty() .............................................................. 854
  33.131.3.14 erase() ............................................................... 854
  33.131.3.15 hexdump() ........................................................... 854
  33.131.3.16 insert() [1/2] ......................................................... 855
  33.131.3.17 insert() [2/2] ......................................................... 855
  33.131.3.18 operator+=() [1/3] .................................................. 855
  33.131.3.19 operator+=() [2/3] .................................................. 856
  33.131.3.20 operator+=() [3/3] .................................................. 856
  33.131.3.21 operator=() [1/2] ..................................................... 856
  33.131.3.22 operator=() [2/2] ..................................................... 856
  33.131.3.23 operator[]() [1/2] ................................................. 857
  33.131.3.24 operator[]() [2/2] ................................................. 857
  33.131.3.25 push_back() .......................................................... 857
  33.131.3.26 replace() [1/2] ....................................................... 857
  33.131.3.27 replace() [2/2] ....................................................... 858
  33.131.3.28 reserve() ............................................................ 858
  33.131.3.29 resize() ............................................................. 858
  33.131.3.30 size() ............................................................... 859
  33.131.3.31 strlcn() ............................................................. 859
  33.131.3.32 substr() ............................................................. 859
33.131.4 Member Data Documentation ............................................... 859
  33.131.4.1 npos ............................................................... 859
33.132 Fl_Surface_Device Class Reference ....................................... 860
  33.132.1 Detailed Description ................................................... 860
  33.132.2 Member Function Documentation ................................... 861
    33.132.2.1 end_current() ..................................................... 861
    33.132.2.2 is_current() ....................................................... 861

Generated by Doxygen
33.132.2.3 pop_current() ................................. 861
33.132.2.4 push_current() ............................. 862
33.132.2.5 set_current() .............................. 862
33.132.2.6 surface() ................................... 862

33.133 Fl_SVG_File_Surface Class Reference ............... 862
33.133.1 Detailed Description ............................ 863
33.133.2 Constructor & Destructor Documentation ......... 863
33.133.2.1 Fl_SVG_File_Surface() ...................... 864
33.133.2.2 ~Fl_SVG_File_Surface() ...................... 864
33.133.3 Member Function Documentation ................. 864
33.133.3.1 close() .................................... 864
33.133.3.2 origin()[1/2] ................................ 864
33.133.3.3 origin()[2/2] ................................ 865
33.133.3.4 printable_rect() ............................ 865
33.133.3.5 translate() ................................ 865
33.133.3.6 untranslate() ............................... 865

33.134 Fl_SVG_Image Class Reference ...................... 865
33.134.1 Detailed Description ............................ 866
33.134.2 Constructor & Destructor Documentation ......... 867
33.134.2.1 Fl_SVG_Image()[1/3] ......................... 868
33.134.2.2 Fl_SVG_Image()[2/3] ......................... 868
33.134.2.3 Fl_SVG_Image()[3/3] ......................... 868
33.134.3 Member Function Documentation ................. 868
33.134.3.1 as_svg_image() ............................. 869
33.134.3.2 color_average() ............................ 869
33.134.3.3 copy() .................................... 869
33.134.3.4 desaturate() ............................... 870
33.134.3.5 draw() .................................... 870
33.134.3.6 normalize() ............................... 870
33.134.3.7 resize() .................................... 870
33.134.4 Member Data Documentation ..................... 870
33.134.4.1 proportional ................................ 871

33.135 Fl_Sys_Menu_Bar Class Reference .................... 871
33.135.1 Detailed Description ............................ 872
33.135.2 Member Enumeration Documentation ............... 873
33.135.2.1 window_menu_style_enum ..................... 873
33.135.3 Constructor & Destructor Documentation ......... 873
33.135.3.1 Fl_Sys_Menu_Bar() .......................... 874
33.135.4 Member Function Documentation .................. 874
33.135.4.1 about() .................................... 874
33.135.4.2 add()[1/3] .................................. 874
33.135.4.3 add()[2/3] .................................. 874
33.136.4.22 init_sizes() ........................................... 892
33.136.4.23 insert() .................................................. 892
33.136.4.24 is_interactive_resize() ................................ 892
33.136.4.25 is_selected() ............................................. 892
33.136.4.26 move_cursor() ........................................... 892
33.136.4.27 recalc_dimensions() .................................... 893
33.136.4.28 redraw_range() .......................................... 893
33.136.4.29 resize() .................................................. 893
33.136.4.30 row_col_clamp() ......................................... 893
33.136.4.31 row_header() ............................................ 893
33.136.4.32 row_height() ............................................. 893
33.136.4.33 row_height_all() ........................................ 894
33.136.4.34 row_resize() ............................................. 894
33.136.4.35 row_resize_min() ........................................ 894
33.136.4.36 rows() ................................................... 894
33.136.4.37 scrollbar_size() [1/2] .................................. 894
33.136.4.38 scrollbar_size() [2/2] .................................. 894
33.136.4.39 set_selection() ........................................... 895
33.136.4.40 tab_cell_nav() [1/2] ..................................... 895
33.136.4.41 tab_cell_nav() [2/2] ..................................... 895
33.136.4.42 table_box() ............................................... 896
33.136.4.43 table_resized() ......................................... 896
33.136.4.44 table_scrolled() ......................................... 896
33.136.4.45 top_row() [1/2] .......................................... 896
33.136.4.46 top_row() [2/2] .......................................... 896
33.136.4.47 visible_cells() .......................................... 896
33.136.4.48 when() ................................................... 897

33.137 Fl_Table_Row Class Reference .............................................. 897
33.137.1 Detailed Description .......................................... 898
33.137.2 Constructor & Destructor Documentation .......................... 898
33.137.2.1 Fl_Table_Row() ............................................ 898
33.137.2.2 ~Fl_Table_Row() ........................................... 898
33.137.3 Member Function Documentation .................................. 898
33.137.3.1 clear() ................................................... 899
33.137.3.2 handle() ................................................... 899
33.137.3.3 row_selected() ............................................ 899
33.137.3.4 rows() ................................................... 899
33.137.3.5 select_all_rows() ......................................... 899
33.137.3.6 select_row() .............................................. 899
33.137.3.7 type() .................................................... 899

33.138 Fl_Tabs Class Reference ............................................. 900
33.138.1 Detailed Description .......................................... 901
33.138.2 Constructor & Destructor Documentation ........................................... 904
33.138.2.1 Fl_Tabs() ................................................................. 904
33.138.3 Member Function Documentation .................................................... 905
33.138.3.1 client_area() .............................................................. 905
33.138.3.2 draw() ........................................................................ 905
33.138.3.3 handle() ..................................................................... 906
33.138.3.4 handle_overflow() ................................................................... 906
33.138.3.5 hit_close() ..................................................................... 906
33.138.3.6 on_insert() ...................................................................... 907
33.138.3.7 on_move() ...................................................................... 907
33.138.3.8 on_remove() .................................................................... 907
33.138.3.9 push() [1/2] ..................................................................... 907
33.138.3.10 push() [2/2] ................................................................... 907
33.138.3.11 resize() ........................................................................ 908
33.138.3.12 tab_align() [1/2] ............................................................... 908
33.138.3.13 tab_align() [2/2] ............................................................... 908
33.138.3.14 value() [1/2] .................................................................. 908
33.138.3.15 value() [2/2] .................................................................. 908
33.138.3.16 which() .......................................................................... 909

33.139 Fl_Text_Buffer Class Reference .......................................................... 909
33.139.1 Detailed Description ....................................................................... 914
33.139.2 Constructor & Destructor Documentation ........................................ 914
33.139.2.1 Fl_Text_Buffer() ................................................................ 914
33.139.3 Member Function Documentation .................................................. 915
33.139.3.1 add_modify_callback() ......................................................... 915
33.139.3.2 address() [1/2] .................................................................. 915
33.139.3.3 address() [2/2] .................................................................. 915
33.139.3.4 append() ........................................................................... 915
33.139.3.5 appendfile() ...................................................................... 916
33.139.3.6 byte_at() ........................................................................... 916
33.139.3.7 can_redo() ........................................................................ 916
33.139.3.8 can_undo() ........................................................................ 916
33.139.3.9 canUndo() ....................................................................... 916
33.139.3.10 char_at() ........................................................................ 917
33.139.3.11 copy() ............................................................................ 917
33.139.3.12 count_displayed_characters() ........................................... 917
33.139.3.13 count_lines() .................................................................. 917
33.139.3.14 findchar_backward() ........................................................ 917
33.139.3.15 findchar_forward() ........................................................... 918
33.139.3.16 highlight_text() ................................................................. 918
33.139.3.17 insert() ............................................................................ 918
33.139.3.18 insert_() ........................................................................ 919
33.140.4.42 linenumber_fgcolor() ......................................................... 949
33.140.4.43 linenumber_font() ............................................................... 949
33.140.4.44 linenumber_format() ............................................................. 950
33.140.4.45 linenumber_size() ............................................................... 950
33.140.4.46 linenumber_width() ............................................................. 950
33.140.4.47 longest_vline() ................................................................. 950
33.140.4.48 maintain_absolute_top_line_number() .................................... 950
33.140.4.49 maintaining_absolute_top_line_number() .................................. 951
33.140.4.50 measure_deleted_lines() ...................................................... 951
33.140.4.51 measure_proportional_character() ......................................... 951
33.140.4.52 measure_vline() .................................................................. 952
33.140.4.53 move_down() ...................................................................... 952
33.140.4.54 move_left() ........................................................................ 952
33.140.4.55 move_right() ....................................................................... 952
33.140.4.56 move_up() ........................................................................... 952
33.140.4.57 offset_line_starts() ............................................................... 953
33.140.4.58 overstrike() .......................................................................... 953
33.140.4.59 position_style() ................................................................. 953
33.140.4.60 position_to_line() ............................................................... 954
33.140.4.61 position_to_linecol() ............................................................. 954
33.140.4.62 position_to_xy() ................................................................. 954
33.140.4.63 redisplay_range() ................................................................. 955
33.140.4.64 reset_absolute_top_line_number() ......................................... 955
33.140.4.65 resize() ................................................................................ 955
33.140.4.66 rewind_lines() ..................................................................... 956
33.140.4.67 scroll() .............................................................................. 956
33.140.4.68 scroll_() ............................................................................ 956
33.140.4.69 scroll_timer_cb() ................................................................ 956
33.140.4.70 scrollbar_align() [1/2] ............................................................. 957
33.140.4.71 scrollbar_align() [2/2] ............................................................. 957
33.140.4.72 scrollbar_size() [1/2] ............................................................. 957
33.140.4.73 scrollbar_size() [2/2] ............................................................. 957
33.140.4.74 scrollbar_width() [1/2] ........................................................... 958
33.140.4.75 scrollbar_width() [2/2] ........................................................... 958
33.140.4.76 secondary_selection_color() [1/2] ....................................... 958
33.140.4.77 secondary_selection_color() [2/2] ....................................... 958
33.140.4.78 shortcut() [1/2] ................................................................. 958
33.140.4.79 shortcut() [2/2] ................................................................. 959
33.140.4.80 show_cursor() ..................................................................... 959
33.140.4.81 show_insert_position() ........................................................ 959
33.140.4.82 skip_lines() ....................................................................... 959
33.140.4.83 spelling_underline_color() [1/2] ........................................... 959
33.140.4.84 spelling_underline_color() [2/2] ................................. 960
33.140.4.85 string_width() .................................................. 960
33.140.4.86 style_buffer() ................................................... 960
33.140.4.87 textcolor() [1/2] .................................................. 960
33.140.4.88 textcolor() [2/2] .................................................. 961
33.140.4.89 textfont() [1/2] .................................................. 961
33.140.4.90 textfont() [2/2] .................................................. 961
33.140.4.91 textsize() [1/2] .................................................. 961
33.140.4.92 textsize() [2/2] .................................................. 961
33.140.4.93 update_h_scrollbar() ............................................. 962
33.140.4.94 update_line_starts() ............................................ 962
33.140.4.95 update_v_scrollbar() ............................................. 962
33.140.4.96 vline_length() ................................................... 962
33.140.4.97 word_end() ....................................................... 963
33.140.4.98 word_start() ..................................................... 963
33.140.4.99 wrap_mode() ..................................................... 963
33.140.4.100 wrap_uses_character() ........................................ 964
33.140.4.101 wrapped_column() .............................................. 964
33.140.4.102 wrapped_line_counter() ...................................... 965
33.140.4.103 wrapped_row() ................................................ 965
33.140.4.104 x_to_col() ..................................................... 966
33.140.4.105 xy_to_position() .............................................. 966
33.140.4.106 xy_to_rowcol() ................................................ 966

33.141 Fl_Text_Editor Class Reference ....................................... 967
  33.141.1 Detailed Description ............................................... 969
    33.141.2 Member Function Documentation ................................ 969
      33.141.2.1 add_key_binding() ....................................... 969
      33.141.2.2 handle() .................................................. 970
      33.141.2.3 insert_mode() [1/2] .................................... 970
      33.141.2.4 insert_mode() [2/2] .................................... 970
      33.141.2.5 kf_backspace() ......................................... 970
      33.141.2.6 kf_c_s_move() ........................................... 970
      33.141.2.7 kf_copy() ................................................ 970
      33.141.2.8 kf_ctrl_move() ......................................... 970
      33.141.2.9 kf_cut() .................................................. 971
      33.141.2.10 kf_default() ............................................ 971
      33.141.2.11 kf_delete() .............................................. 971
      33.141.2.12 kf_down() ................................................ 971
      33.141.2.13 kf_end() .................................................. 971
      33.141.2.14 kf_enter() ............................................... 971
      33.141.2.15 kf_home() ............................................... 971
      33.141.2.16 kf_ignore() .............................................. 972
33.144.1 Detailed Description ................................................................. 983
33.144.2 Constructor & Destructor Documentation ................................. 983
   33.144.2.1 Fl_Tiled_Image() .......................................................... 983
33.144.3 Member Function Documentation ............................................. 984
   33.144.3.1 color_average() ............................................................ 984
   33.144.3.2 copy() .................................................................. 984
   33.144.3.3 desaturate() ................................................................ 985
   33.144.3.4 draw() .................................................................. 985
33.145 Fl_Timeout Class Reference ....................................................... 985
   33.145.1 Detailed Description ............................................................. 987
   33.145.2 Member Function Documentation .......................................... 987
      33.145.2.1 add_timeout() .............................................................. 987
      33.145.2.2 current() ................................................................. 987
      33.145.2.3 elapse_timeouts() ...................................................... 988
      33.145.2.4 get() .................................................................. 988
      33.145.2.5 has_timeout() ............................................................ 988
      33.145.2.6 insert() .................................................................. 989
      33.145.2.7 make_current() ............................................................ 989
      33.145.2.8 release() .................................................................. 989
      33.145.2.9 remove_timeout() .......................................................... 989
      33.145.2.10 repeat_timeout() .......................................................... 991
      33.145.2.11 time_to_wait() ............................................................. 991
   33.145.3 Member Data Documentation ............................................... 992
      33.145.3.1 current_timeout ............................................................ 992
      33.145.3.2 first_timeout .............................................................. 992
      33.145.3.3 free_timeout .............................................................. 992
33.146 Fl_Timer Class Reference ........................................................ 992
   33.146.1 Detailed Description ............................................................. 993
   33.146.2 Constructor & Destructor Documentation ................................ 993
      33.146.2.1 Fl_Timer() .................................................................. 993
   33.146.3 Member Function Documentation ....................................... 994
      33.146.3.1 direction() [1/2] .......................................................... 994
      33.146.3.2 direction() [2/2] .......................................................... 994
      33.146.3.3 draw() .................................................................. 994
      33.146.3.4 handle() .................................................................. 994
      33.146.3.5 suspended() .................................................................. 994
33.147 Fl_Toggle_Button Class Reference ............................................ 995
   33.147.1 Detailed Description ............................................................. 995
   33.147.2 Constructor & Destructor Documentation ............................... 995
      33.147.2.1 Fl_Toggle_Button() ....................................................... 995
33.148 Fl_Tooltip Class Reference ......................................................... 996
   33.148.1 Detailed Description ............................................................. 997
33.148.2 Member Function Documentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>color()</td>
<td>997</td>
</tr>
<tr>
<td>color()</td>
<td>998</td>
</tr>
<tr>
<td>current()</td>
<td>998</td>
</tr>
<tr>
<td>delay()</td>
<td>998</td>
</tr>
<tr>
<td>delay()</td>
<td>998</td>
</tr>
<tr>
<td>disable()</td>
<td>998</td>
</tr>
<tr>
<td>enable()</td>
<td>998</td>
</tr>
<tr>
<td>enabled()</td>
<td>998</td>
</tr>
<tr>
<td>enter_area()</td>
<td>998</td>
</tr>
<tr>
<td>font()</td>
<td>999</td>
</tr>
<tr>
<td>font()</td>
<td>999</td>
</tr>
<tr>
<td>hidedelay()</td>
<td>999</td>
</tr>
<tr>
<td>hidedelay()</td>
<td>999</td>
</tr>
<tr>
<td>hoverdelay()</td>
<td>999</td>
</tr>
<tr>
<td>hoverdelay()</td>
<td>999</td>
</tr>
<tr>
<td>margin_height()</td>
<td>999</td>
</tr>
<tr>
<td>margin_height()</td>
<td>1000</td>
</tr>
<tr>
<td>margin_width()</td>
<td>1000</td>
</tr>
<tr>
<td>margin_width()</td>
<td>1000</td>
</tr>
<tr>
<td>size()</td>
<td>1000</td>
</tr>
<tr>
<td>size()</td>
<td>1000</td>
</tr>
<tr>
<td>textcolor()</td>
<td>1000</td>
</tr>
<tr>
<td>textcolor()</td>
<td>1000</td>
</tr>
<tr>
<td>wrap_width()</td>
<td>1000</td>
</tr>
<tr>
<td>wrap_width()</td>
<td>1000</td>
</tr>
</tbody>
</table>

33.149 Fl_Tree Class Reference

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add()</td>
<td>1011</td>
</tr>
<tr>
<td>add()</td>
<td>1012</td>
</tr>
<tr>
<td>calc_dimensions()</td>
<td>1012</td>
</tr>
<tr>
<td>calc_tree()</td>
<td>1013</td>
</tr>
<tr>
<td>callback_item()</td>
<td>1013</td>
</tr>
<tr>
<td>callback_item()</td>
<td>1013</td>
</tr>
<tr>
<td>callback_reason()</td>
<td>1013</td>
</tr>
<tr>
<td>callback_reason()</td>
<td>1014</td>
</tr>
<tr>
<td>clear()</td>
<td>1014</td>
</tr>
<tr>
<td>clear_children()</td>
<td>1014</td>
</tr>
<tr>
<td>close()</td>
<td>1014</td>
</tr>
<tr>
<td>close()</td>
<td>1015</td>
</tr>
<tr>
<td>closeicon()</td>
<td>1015</td>
</tr>
</tbody>
</table>
33.149.2.56 item_reselect_mode() [1/2] 
33.149.2.57 item_reselect_mode() [2/2] 
33.149.2.58 last() 
33.149.2.59 last_selected_item() 
33.149.2.60 last_visible() 
33.149.2.61 last_visible_item() 
33.149.2.62 load() 
33.149.2.63 next() 
33.149.2.64 next_item() . 
33.149.2.65 next_selected_item() 
33.149.2.66 next_visible_item() 
33.149.2.67 open() [1/2] 
33.149.2.68 open() [2/2] 
33.149.2.69 open_toggle() 
33.149.2.70 openicon() [1/2] 
33.149.2.71 openicon() [2/2] 
33.149.2.72 prev() 
33.149.2.73 recalc_tree() 
33.149.2.74 remove() 
33.149.2.75 resize() 
33.149.2.76 root() 
33.149.2.77 root_label() 
33.149.2.78 scrollbar_size() [1/2] 
33.149.2.79 scrollbar_size() [2/2] 
33.149.2.80 select() [1/2] 
33.149.2.81 select() [2/2] 
33.149.2.82 select_all() . 
33.149.2.83 select_only() 
33.149.2.84 select_toggle() 
33.149.2.85 selectbox() [1/2] 
33.149.2.86 selectbox() [2/2] 
33.149.2.87 selectmode() [1/2] 
33.149.2.88 selectmode() [2/2] 
33.149.2.89 set_item_focus() . 
33.149.2.90 show_item() [1/2] 
33.149.2.91 show_item() [2/2] 
33.149.2.92 show_item_bottom() 
33.149.2.93 show_item_middle() 
33.149.2.94 show_item_top() . 
33.149.2.95 show_self() 
33.149.2.96 showcollapse() [1/2] 
33.149.2.97 showcollapse() [2/2] 

Generated by Doxygen
33.149.2.98 showroot() ........................................... 1042
33.149.2.99 sortorder() ........................................... 1042
33.149.2.100 usericon() [1/2] ..................................... 1042
33.149.2.101 usericon() [2/2] ..................................... 1042
33.149.2.102 vposition() [1/2] .................................... 1042
33.149.2.103 vposition() [2/2] .................................... 1043

33.150 Fl_Tree_Item Class Reference .................................. 1043
33.150.1 Detailed Description ......................................... 1048
33.150.2 Constructor & Destructor Documentation ......................... 1048
  33.150.2.1 Fl_Tree_Item() [1/2] .................................... 1048
  33.150.2.2 Fl_Tree_Item() [2/2] .................................... 1049
33.150.3 Member Function Documentation ............................... 1049
  33.150.3.1 activate() ............................................. 1049
  33.150.3.2 add() [1/4] ............................................ 1049
  33.150.3.3 add() [2/4] ............................................ 1049
  33.150.3.4 add() [3/4] ............................................ 1050
  33.150.3.5 add() [4/4] ............................................ 1050
  33.150.3.6 calc_item_height() ..................................... 1050
  33.150.3.7 child() .............................................. 1050
  33.150.3.8 deactivate() .......................................... 1051
  33.150.3.9 deparent() ........................................... 1051
  33.150.3.10 depth() ............................................. 1051
  33.150.3.11 deselect_all() ....................................... 1051
  33.150.3.12 draw() .............................................. 1051
  33.150.3.13 draw_horizontal_connector() ................................ 1052
  33.150.3.14 draw_item_content() .................................. 1052
  33.150.3.15 draw_vertical_connector() ................................ 1053
  33.150.3.16 drawbgcolor() ........................................ 1053
  33.150.3.17 drawfgcolor() ........................................ 1053
  33.150.3.18 find_child() [1/2] .................................... 1054
  33.150.3.19 find_child() [2/2] .................................... 1054
  33.150.3.20 find_child_item() [1/2] ................................ 1054
  33.150.3.21 find_child_item() [2/2] ................................ 1054
  33.150.3.22 find_clicked() ........................................ 1054
  33.150.3.23 find_item() .......................................... 1055
  33.150.3.24 hide_widgets() ....................................... 1055
  33.150.3.25 insert() ............................................. 1055
  33.150.3.26 insert_above() ....................................... 1056
  33.150.3.27 label() ............................................... 1056
  33.150.3.28 label_h() ............................................ 1056
  33.150.3.29 label_w() ............................................ 1056
  33.150.3.30 label_x() ............................................ 1056
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>label_y()</td>
<td>1056</td>
</tr>
<tr>
<td>labelbgcolor()</td>
<td>1057</td>
</tr>
<tr>
<td>labelbgcolor()</td>
<td>1057</td>
</tr>
<tr>
<td>move()</td>
<td>1057</td>
</tr>
<tr>
<td>move()</td>
<td>1057</td>
</tr>
<tr>
<td>move_above()</td>
<td>1058</td>
</tr>
<tr>
<td>move_below()</td>
<td>1058</td>
</tr>
<tr>
<td>move_into()</td>
<td>1058</td>
</tr>
<tr>
<td>next()</td>
<td>1059</td>
</tr>
<tr>
<td>next_displayed()</td>
<td>1059</td>
</tr>
<tr>
<td>next_sibling()</td>
<td>1059</td>
</tr>
<tr>
<td>next_visible()</td>
<td>1059</td>
</tr>
<tr>
<td>parent()</td>
<td>1059</td>
</tr>
<tr>
<td>prefs()</td>
<td>1060</td>
</tr>
<tr>
<td>prev()</td>
<td>1060</td>
</tr>
<tr>
<td>prevDisplayed()</td>
<td>1060</td>
</tr>
<tr>
<td>prev_sibling()</td>
<td>1060</td>
</tr>
<tr>
<td>prev_visible()</td>
<td>1060</td>
</tr>
<tr>
<td>recalc_tree()</td>
<td>1061</td>
</tr>
<tr>
<td>remove_child()</td>
<td>1061</td>
</tr>
<tr>
<td>remove_child()</td>
<td>1061</td>
</tr>
<tr>
<td>reparent()</td>
<td>1061</td>
</tr>
<tr>
<td>replace()</td>
<td>1062</td>
</tr>
<tr>
<td>replace_child()</td>
<td>1062</td>
</tr>
<tr>
<td>select()</td>
<td>1062</td>
</tr>
<tr>
<td>select_all()</td>
<td>1063</td>
</tr>
<tr>
<td>show_self()</td>
<td>1063</td>
</tr>
<tr>
<td>show_widgets()</td>
<td>1063</td>
</tr>
<tr>
<td>swap_children()</td>
<td>1063</td>
</tr>
<tr>
<td>swap_children()</td>
<td>1063</td>
</tr>
<tr>
<td>tree()</td>
<td>1063</td>
</tr>
<tr>
<td>tree()</td>
<td>1064</td>
</tr>
<tr>
<td>update_prev_next()</td>
<td>1064</td>
</tr>
<tr>
<td>userdeicon()</td>
<td>1064</td>
</tr>
<tr>
<td>userdeicon()</td>
<td>1064</td>
</tr>
<tr>
<td>usericon()</td>
<td>1065</td>
</tr>
<tr>
<td>visible_r()</td>
<td>1065</td>
</tr>
</tbody>
</table>

33.151 Fl_Tree_Item_Array Class Reference

33.151.1 Detailed Description

33.151.2 Constructor & Destructor Documentation

33.151.3 Member Function Documentation

Generated by Doxygen
33.151.3.1 add() ................................................................. 1066
33.151.3.2 clear() ............................................................. 1066
33.151.3.3 deparent() .......................................................... 1067
33.151.3.4 insert() ............................................................... 1067
33.151.3.5 manage_item_destroy() ........................................ 1067
33.151.3.6 move() ................................................................. 1067
33.151.3.7 remove() ............................................................... 1067
33.151.3.8 remove() ............................................................... 1067
33.151.3.9 reparent() ............................................................ 1068
33.151.3.10 replace() .............................................................. 1068

33.152 Fl_Tree_Prefs Class Reference ........................................... 1068
  33.152.1 Detailed Description .................................................. 1071
  33.152.2 Member Function Documentation .................................... 1071
      33.152.2.1 closedeicon() .................................................. 1071
      33.152.2.2 closeicon() ..................................................... 1071
      33.152.2.3 item_draw_mode() ............................................ 1071
      33.152.2.4 item_labelbgcolor() [1/2] .................................. 1071
      33.152.2.5 item_labelbgcolor() [2/2] .................................. 1072
      33.152.2.6 marginbottom() ............................................... 1072
      33.152.2.7 opendeicon() .................................................. 1072
      33.152.2.8 openicon() [1/2] .............................................. 1072
      33.152.2.9 openicon() [2/2] .............................................. 1072
      33.152.2.10 selectmode() ............................................... 1072
      33.152.2.11 showcollapse() .............................................. 1072
      33.152.2.12 showroot() ................................................... 1073
      33.152.2.13 sortorder() .................................................. 1073
      33.152.2.14 userdeicon() ................................................ 1073

33.153 Fl_Valuator Class Reference ............................................. 1073
  33.153.1 Detailed Description ................................................ 1075
  33.153.2 Constructor & Destructor Documentation .......................... 1076
      33.153.2.1 Fl_Valuator() ............................................... 1076
  33.153.3 Member Function Documentation ................................... 1076
      33.153.3.1 format() ...................................................... 1076
      33.153.3.2 increment() .................................................. 1076
      33.153.3.3 maximum() [1/2] ............................................. 1076
      33.153.3.4 maximum() [2/2] ............................................. 1076
      33.153.3.5 minimum() [1/2] ............................................. 1076
      33.153.3.6 minimum() [2/2] ............................................. 1077
      33.153.3.7 precision() .................................................. 1077
      33.153.3.8 range() ....................................................... 1077
      33.153.3.9 round() ....................................................... 1077
      33.153.3.10 step() ....................................................... 1077

Generated by Doxygen
<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.153.11 value() [1/2]</td>
</tr>
<tr>
<td>33.153.12 value() [2/2]</td>
</tr>
<tr>
<td>33.153.13 value_damage()</td>
</tr>
<tr>
<td>33.154 Fl_Value_Input Class Reference</td>
</tr>
<tr>
<td>33.154.1 Detailed Description</td>
</tr>
<tr>
<td>33.154.2 Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>33.154.2.1 Fl_Value_Input()</td>
</tr>
<tr>
<td>33.154.3 Member Function Documentation</td>
</tr>
<tr>
<td>33.154.3.1 cursor_color() [1/2]</td>
</tr>
<tr>
<td>33.154.3.2 cursor_color() [2/2]</td>
</tr>
<tr>
<td>33.154.3.3 draw()</td>
</tr>
<tr>
<td>33.154.3.4 handle()</td>
</tr>
<tr>
<td>33.154.3.5 resize()</td>
</tr>
<tr>
<td>33.154.3.6 shortcut() [1/2]</td>
</tr>
<tr>
<td>33.154.3.7 shortcut() [2/2]</td>
</tr>
<tr>
<td>33.154.3.8 soft()</td>
</tr>
<tr>
<td>33.154.3.9 textcolor()</td>
</tr>
<tr>
<td>33.154.3.10 textfont() [1/2]</td>
</tr>
<tr>
<td>33.154.3.11 textfont() [2/2]</td>
</tr>
<tr>
<td>33.154.3.12 textsize() [1/2]</td>
</tr>
<tr>
<td>33.154.3.13 textsize() [2/2]</td>
</tr>
<tr>
<td>33.155 Fl_Value_Output Class Reference</td>
</tr>
<tr>
<td>33.155.1 Detailed Description</td>
</tr>
<tr>
<td>33.155.2 Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>33.155.2.1 Fl_Value_Output()</td>
</tr>
<tr>
<td>33.155.3 Member Function Documentation</td>
</tr>
<tr>
<td>33.155.3.1 draw()</td>
</tr>
<tr>
<td>33.155.3.2 handle()</td>
</tr>
<tr>
<td>33.155.3.3 soft() [1/2]</td>
</tr>
<tr>
<td>33.155.3.4 soft() [2/2]</td>
</tr>
<tr>
<td>33.155.3.5 textcolor() [1/2]</td>
</tr>
<tr>
<td>33.155.3.6 textcolor() [2/2]</td>
</tr>
<tr>
<td>33.155.3.7 textfont() [1/2]</td>
</tr>
<tr>
<td>33.155.3.8 textfont() [2/2]</td>
</tr>
<tr>
<td>33.155.3.9 textsize()</td>
</tr>
<tr>
<td>33.156 Fl_Value_Slider Class Reference</td>
</tr>
<tr>
<td>33.156.1 Detailed Description</td>
</tr>
<tr>
<td>33.156.2 Constructor &amp; Destructor Documentation</td>
</tr>
<tr>
<td>33.156.2.1 Fl_Value_Slider()</td>
</tr>
<tr>
<td>33.156.3 Member Function Documentation</td>
</tr>
<tr>
<td>33.156.3.1 draw()</td>
</tr>
<tr>
<td>33.156.3.2 handle()</td>
</tr>
</tbody>
</table>
33.156.3.3 value_height() [1/2] ........................................................................ 1088
33.156.3.4 value_height() [2/2] ........................................................................ 1088
33.156.3.5 value_width() [1/2] ......................................................................... 1088
33.156.3.6 value_width() [2/2] ......................................................................... 1088
33.157 Fl_Widget Class Reference ........................................................................ 1090
33.157.1 Detailed Description .............................................................................. 1096
33.157.2 Member Enumeration Documentation .................................................. 1096
33.157.2.1 anonymous enum .............................................................................. 1097
33.157.3 Constructor & Destructor Documentation ............................................... 1097
33.157.3.1 Fl_Widget() ....................................................................................... 1097
33.157.3.2 ~Fl_Widget() ..................................................................................... 1098
33.157.4 Member Function Documentation ......................................................... 1098
33.157.4.1 activate() ......................................................................................... 1098
33.157.4.2 active() .......................................................................................... 1098
33.157.4.3 active_r() ......................................................................................... 1099
33.157.4.4 align() [1/2] ................................................................................... 1099
33.157.4.5 align() [2/2] ................................................................................... 1099
33.157.4.6 argument() ....................................................................................... 1099
33.157.4.7 argument() [2/2] ............................................................................ 1100
33.157.4.8 as_gl_window() ................................................................................ 1100
33.157.4.9 as_group() ....................................................................................... 1100
33.157.4.10 as_window() .................................................................................. 1101
33.157.4.11 bind_deimage() [1/2] .................................................................... 1101
33.157.4.12 bind_deimage() [2/2] .................................................................... 1101
33.157.4.13 bind_image() [1/2] ....................................................................... 1102
33.157.4.14 bind_image() [2/2] ....................................................................... 1102
33.157.4.15 box() [1/2] ................................................................................... 1102
33.157.4.16 box() [2/2] ................................................................................... 1103
33.157.4.17 callback() [1/5] ............................................................................ 1103
33.157.4.18 callback() [2/5] ............................................................................ 1103
33.157.4.19 callback() [3/5] ............................................................................ 1103
33.157.4.20 callback() [4/5] ............................................................................ 1104
33.157.4.21 callback() [5/5] ............................................................................ 1104
33.157.4.22 changed() ..................................................................................... 1104
33.157.4.23 clear_active() ............................................................................... 1104
33.157.4.24 clear_changed() ........................................................................... 1105
33.157.4.25 clear_damage() .............................................................................. 1105
33.157.4.26 clear_output() ................................................................................ 1105
33.157.4.27 clear_visible() ............................................................................... 1105
33.157.4.28 clear_visible_focus() .................................................................... 1105
33.157.4.29 color() [1/3] ................................................................................ 1106
33.157.4.30 color() [2/3] ................................................................................ 1106

Generated by Doxygen
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>color()</td>
<td>1106</td>
</tr>
<tr>
<td>color2()</td>
<td>1106</td>
</tr>
<tr>
<td>contains()</td>
<td>1107</td>
</tr>
<tr>
<td>copy_label()</td>
<td>1107</td>
</tr>
<tr>
<td>copy_tooltip()</td>
<td>1107</td>
</tr>
<tr>
<td>damage()</td>
<td>1107</td>
</tr>
<tr>
<td>damage()</td>
<td>1107</td>
</tr>
<tr>
<td>damage()</td>
<td>1107</td>
</tr>
<tr>
<td>deactivate()</td>
<td>1108</td>
</tr>
<tr>
<td>deimage()</td>
<td>1108</td>
</tr>
<tr>
<td>deimage()</td>
<td>1108</td>
</tr>
<tr>
<td>deimage()</td>
<td>1108</td>
</tr>
<tr>
<td>deimage()</td>
<td>1108</td>
</tr>
<tr>
<td>deimage_bound()</td>
<td>1108</td>
</tr>
<tr>
<td>do_callback()</td>
<td>1110</td>
</tr>
<tr>
<td>do_callback()</td>
<td>1110</td>
</tr>
<tr>
<td>do_callback()</td>
<td>1110</td>
</tr>
<tr>
<td>draw()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_focus()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_focus()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_focus()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_label()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_label()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_label()</td>
<td>1111</td>
</tr>
<tr>
<td>draw_label()</td>
<td>1111</td>
</tr>
<tr>
<td>h()</td>
<td>1114</td>
</tr>
<tr>
<td>h()</td>
<td>1114</td>
</tr>
<tr>
<td>handle()</td>
<td>1114</td>
</tr>
<tr>
<td>hide()</td>
<td>1114</td>
</tr>
<tr>
<td>image()</td>
<td>1115</td>
</tr>
<tr>
<td>image()</td>
<td>1115</td>
</tr>
<tr>
<td>image()</td>
<td>1115</td>
</tr>
<tr>
<td>image()</td>
<td>1115</td>
</tr>
<tr>
<td>image_bound()</td>
<td>1115</td>
</tr>
<tr>
<td>inside()</td>
<td>1115</td>
</tr>
<tr>
<td>is_label_copied()</td>
<td>1115</td>
</tr>
<tr>
<td>label()</td>
<td>1115</td>
</tr>
<tr>
<td>label()</td>
<td>1115</td>
</tr>
<tr>
<td>label()</td>
<td>1115</td>
</tr>
<tr>
<td>label_shortcut()</td>
<td>1115</td>
</tr>
<tr>
<td>labelcolor()</td>
<td>1115</td>
</tr>
<tr>
<td>labelcolor()</td>
<td>1115</td>
</tr>
<tr>
<td>labelcolor()</td>
<td>1115</td>
</tr>
</tbody>
</table>
33.157.4.73 labelcolor() [2/2] ........................................... 1118
33.157.4.74 labelfont() [1/2] ........................................ 1118
33.157.4.75 labelfont() [2/2] ........................................ 1118
33.157.4.76 labelsize() [1/2] ........................................ 1119
33.157.4.77 labelsize() [2/2] ........................................ 1119
33.157.4.78 labeltype() [1/2] ........................................ 1119
33.157.4.79 labeltype() [2/2] ........................................ 1119
33.157.4.80 measure_label() ........................................ 1120
33.157.4.81 needs_keyboard() [1/2] ................................. 1120
33.157.4.82 needs_keyboard() [2/2] ................................. 1120
33.157.4.83 output() .................................................. 1120
33.157.4.84 parent() [1/2] ............................................ 1121
33.157.4.85 parent() [2/2] ............................................ 1121
33.157.4.86 position() ............................................... 1121
33.157.4.87 redraw() ................................................ 1121
33.157.4.88 redraw_label() ........................................ 1122
33.157.4.89 resize() ................................................ 1122
33.157.4.90 selection_color() [1/2] ................................. 1122
33.157.4.91 selection_color() [2/2] ................................. 1122
33.157.4.92 set_active() ............................................. 1123
33.157.4.93 set_changed() .......................................... 1123
33.157.4.94 set_output() ............................................ 1123
33.157.4.95 set_visible() .......................................... 1123
33.157.4.96 set_visible_focus() .................................. 1123
33.157.4.97 shortcut_label() [1/2] ................................. 1124
33.157.4.98 shortcut_label() [2/2] ................................. 1124
33.157.4.99 show() .................................................. 1124
33.157.4.100 size() ................................................ 1124
33.157.4.101 take_focus() ......................................... 1125
33.157.4.102 takesevents() ....................................... 1125
33.157.4.103 test_shortcut() [1/2] ................................. 1125
33.157.4.104 test_shortcut() [2/2] ................................. 1125
33.157.4.105 tooltip() [1/2] ...................................... 1126
33.157.4.106 tooltip() [2/2] ...................................... 1126
33.157.4.107 top_window() ........................................ 1126
33.157.4.108 top_window_offset() ................................ 1127
33.157.4.109 type() [1/2] ........................................... 1127
33.157.4.110 type() [2/2] ........................................... 1127
33.157.4.111 user_data() [1/2] ................................... 1127
33.157.4.112 user_data() [2/2] ................................... 1128
33.157.4.113 visible() ............................................. 1129
33.157.4.114 visible_focus() [1/2] ................................. 1129
33.160.3.6 clear_border() ........................................ 1143
33.160.3.7 clear_modal_states() ................................ 1143
33.160.3.8 current() ............................................. 1144
33.160.3.9 cursor() [1/3] ........................................ 1144
33.160.3.10 cursor() [2/3] ........................................ 1144
33.160.3.11 cursor() [3/3] ........................................ 1144
33.160.3.12 decorated_h() ........................................ 1144
33.160.3.13 decorated_w() ........................................ 1144
33.160.3.14 default_cursor() [1/2] .............................. 1145
33.160.3.15 default_cursor() [2/2] .............................. 1145
33.160.3.16 default_icon() ....................................... 1145
33.160.3.17 default_icons() [1/2] ............................... 1146
33.160.3.18 default_icons() [2/2] ............................... 1146
33.160.3.19 default_size_range() ................................ 1147
33.160.3.20 default_xclass() [1/2] ............................. 1148
33.160.3.21 default_xclass() [2/2] ............................. 1148
33.160.3.22 draw() ................................................ 1148
33.160.3.23 flush() .............................................. 1148
33.160.3.24 force_position() [1/2] ............................ 1148
33.160.3.25 force_position() [2/2] ............................ 1149
33.160.3.26 free_icons() .......................................... 1149
33.160.3.27 free_position() ...................................... 1149
33.160.3.28 fullscreen() ......................................... 1149
33.160.3.29 fullscreen_screens() ............................... 1150
33.160.3.30 handle() ............................................. 1150
33.160.3.31 hide() ................................................ 1150
33.160.3.32 hotspot() ............................................ 1150
33.160.3.33 icon() [1/3] .......................................... 1151
33.160.3.34 icon() [2/3] .......................................... 1151
33.160.3.35 icon() [3/3] .......................................... 1151
33.160.3.36 iconize() ............................................ 1152
33.160.3.37 icons() [1/2] ......................................... 1152
33.160.3.38 icons() [2/2] ......................................... 1152
33.160.3.39 is_resizable() ....................................... 1152
33.160.3.40 make_current() ...................................... 1153
33.160.3.41 modal() ............................................... 1153
33.160.3.42 os_id() ............................................... 1153
33.160.3.43 resize() ............................................. 1153
33.160.3.44 screen_num() ...................................... 1154
33.160.3.45 set_menu_window() ................................ 1154
33.160.3.46 set_modal() .......................................... 1154
33.160.3.47 set_non_modal() .................................... 1154
33.160.3.48 set_tooltip_window() ................................................................. 1154
33.160.3.49 shape() [1/2] ............................................................................. 1155
33.160.3.50 shape() [2/2] ............................................................................. 1155
33.160.3.51 show() [1/2] ............................................................................. 1156
33.160.3.52 show() [2/2] ............................................................................. 1156
33.160.3.53 shown() .................................................................................... 1156
33.160.3.54 size_range() ............................................................................ 1157
33.160.3.55 wait_for_expose() ..................................................................... 1157
33.160.3.56 xclass() [1/2] .......................................................................... 1158
33.160.3.57 xclass() [2/2] .......................................................................... 1158

33.160.4 Member Data Documentation .......................................................... 1159
33.160.4.1 current_ ....................................................................................... 1159

33.161 Fl_Wizard Class Reference ................................................................. 1159
33.161.1 Detailed Description ....................................................................... 1160
33.161.2 Constructor & Destructor Documentation ....................................... 1160
33.161.2.1 Fl_Wizard() .............................................................................. 1160
33.161.3 Member Function Documentation .................................................. 1160
33.161.3.1 draw() ....................................................................................... 1160
33.161.3.2 next() ....................................................................................... 1160

33.162 Fl_XBM_Image Class Reference ......................................................... 1160
33.162.1 Detailed Description ....................................................................... 1161
33.162.2 Constructor & Destructor Documentation ....................................... 1161
33.162.2.1 Fl_XBM_Image() ...................................................................... 1161

33.163 Fl_XColor Struct Reference ............................................................... 1161

33.164 Fl_XPM_Image Class Reference .......................................................... 1161
33.164.1 Detailed Description ....................................................................... 1162
33.164.2 Constructor & Destructor Documentation ....................................... 1162
33.164.2.1 Fl_XPM_Image() ...................................................................... 1162

33.165 Fl_Zenity_Native_File_Chooser_Driver Class Reference ....................... 1162
33.166 Fl_GIF_Image::GIF_FRAME Struct Reference ....................................... 1163
33.167 Fl_ICO_Image::IconDirEntry Struct Reference ..................................... 1163
33.167.1 Detailed Description ....................................................................... 1164

33.168 Fl_Text_Editor::Key_Binding Struct Reference .................................... 1164
33.168.1 Detailed Description ....................................................................... 1164

33.169 Fl_Preferences::Name Class Reference .............................................. 1164
33.169.1 Detailed Description ....................................................................... 1164
33.169.2 Constructor & Destructor Documentation ....................................... 1165
33.169.2.1 Name() [1/2] ............................................................................ 1165
33.169.2.2 Name() [2/2] ............................................................................ 1165

33.170 Fl_Preferences::Node Class Reference ............................................... 1165
33.171 Fl_Paged_Device::page_format Struct Reference .................................. 1166
33.171.1 Detailed Description ....................................................................... 1166
33.172 Fl_Preferences::RootNode Class Reference .......................... 1166
33.173 Fl_Scroll::ScrollInfo Struct Reference ......................... 1166
  33.173.1 Detailed Description ........................................ 1167
33.174 Fl_Text_Display::Style_Table_Entry Struct Reference ...... 1167
  33.174.1 Detailed Description ........................................ 1168

34 File Documentation .................................................................. 1169
  34.1 Enumerations.H File Reference ................................... 1169
    34.1.1 Detailed Description ........................................ 1180
  34.1.2 Macro Definition Documentation .................................. 1180
    34.1.2.1 FL_ABI_VERSION ......................................... 1180
    34.1.2.2 FL_API_VERSION ......................................... 1181
    34.1.2.3 FL_IMAGE_LABEL ....................................... 1181
    34.1.2.4 FL_MAJOR_VERSION ..................................... 1181
    34.1.2.5 FL_MINOR_VERSION ....................................... 1181
    34.1.2.6 FL_MULTI_LABEL ......................................... 1181
    34.1.2.7 FL_PATCH_VERSION ..................................... 1181
    34.1.2.8 FL_SYMBOL_LABEL ....................................... 1182
    34.1.2.9 FL_VERSION ............................................... 1182
    34.1.3 Typedef Documentation ........................................ 1182
      34.1.3.1 Fl_Contrast_Function ................................... 1182
      34.1.3.2 Fl_Fontsize ............................................. 1182
    34.1.4 Enumeration Type Documentation ................................ 1183
      34.1.4.1 anonymous enum ........................................ 1183
      34.1.4.2 Fl_Arrow_Type .......................................... 1183
      34.1.4.3 Fl_Boxtype ............................................. 1183
      34.1.4.4 Fl_Callback_Reason .................................... 1185
      34.1.4.5 Fl_Contrast_Mode ...................................... 1186
      34.1.4.6 Fl_Cursor ................................................. 1186
      34.1.4.7 Fl_Damage ................................................ 1187
      34.1.4.8 Fl_Event .................................................. 1187
      34.1.4.9 Fl_Labeltype ............................................ 1190
      34.1.4.10 Fl_Orientation ......................................... 1191
      34.1.4.11 Fl_When ................................................ 1192
    34.1.5 Function Documentation ...................................... 1192
      34.1.5.1 fl_box() ................................................. 1192
      34.1.5.2 fl_color_cube() ......................................... 1192
      34.1.5.3 fl_define_FL_EMBOSSED_LABEL() ...................... 1193
      34.1.5.4 fl_define_FL_ENGRAVED_LABEL() ...................... 1193
      34.1.5.5 fl_define_FL_ICON_LABEL() ............................ 1193
      34.1.5.6 fl_define_FL_IMAGE_LABEL() .......................... 1193
      34.1.5.7 fl_define_FL_MULTI_LABEL() .......................... 1193
34.1.5.8 fl_define_FL_SHADOW_LABEL() .............................. 1193
34.1.5.9 fl_down() .................................................. 1193
34.1.5.10 fl_frame() .................................................. 1193
34.1.5.11 fl_gray_ramp() ........................................... 1194
34.1.6 Variable Documentation ...................................... 1194
  34.1.6.1 FL_ALIGN_LEFT ........................................... 1194
  34.1.6.2 FL_ALIGN_TOP ........................................... 1194
  34.1.6.3 FL_NORMAL_SIZE ......................................... 1194
34.2 Enumerations.H .................................................. 1194
34.3 filename.H File Reference ...................................... 1202
  34.3.1 Detailed Description ...................................... 1203
34.4 filename.H ..................................................... 1203
34.5 Fl.H File Reference ............................................. 1204
  34.5.1 Detailed Description ...................................... 1206
34.6 Fl.H ............................................................. 1206
34.7 Fl_Adjuster.H ................................................... 1212
34.8 Fl_Anim_GIF_Image.H ........................................... 1213
34.9 fl_ask.H File Reference ........................................ 1214
  34.9.1 Detailed Description ...................................... 1216
  34.9.2 Enumeration Type Documentation ......................... 1216
    34.9.2.1 fl_Beep ................................................ 1216
  34.9.3 Function Documentation ................................... 1216
    34.9.3.1 fl_message_position() ................................ 1216
34.10 fl_ask.H ...................................................... 1217
34.11 fl_attr.h File Reference ...................................... 1218
  34.11.1 Detailed Description .................................... 1218
  34.11.2 Macro Definition Documentation ....................... 1218
    34.11.2.1 __fl_attr ........................................... 1218
    34.11.2.2 FL_DEPRECATED .................................... 1218
34.12 fl_attr.h ...................................................... 1219
34.13 Fl_Bitmap.H ................................................... 1221
34.14 Fl_BMP_Image.H ................................................ 1221
34.15 Fl_Box.H File Reference ...................................... 1222
  34.15.1 Detailed Description .................................... 1222
34.16 Fl_Box.H ...................................................... 1222
34.17 Fl_Browser.H .................................................. 1223
34.18 Fl_Browser_.H .................................................. 1224
34.19 Fl_Button.H .................................................... 1226
34.20 Fl_Cairo.H File Reference .................................... 1227
  34.20.1 Detailed Description .................................... 1227
34.21 Fl_Cairo.H ..................................................... 1228
34.22 Fl_Cairo_Window.H File Reference ............................ 1228
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.22.1</td>
<td>Detailed Description</td>
<td>1229</td>
</tr>
<tr>
<td>34.23</td>
<td>Fl_Cairo_Window.H</td>
<td>1229</td>
</tr>
<tr>
<td>34.24</td>
<td>fl_casts.H</td>
<td>1230</td>
</tr>
<tr>
<td>34.25.1</td>
<td>Detailed Description</td>
<td>1231</td>
</tr>
<tr>
<td>34.25</td>
<td>Fl_Chart.H File Reference</td>
<td>1230</td>
</tr>
<tr>
<td>34.26</td>
<td>Fl_Chart.H</td>
<td>1231</td>
</tr>
<tr>
<td>34.27</td>
<td>Fl_Check_Browser.H</td>
<td>1232</td>
</tr>
<tr>
<td>34.28</td>
<td>Fl_Check_Button.H</td>
<td>1233</td>
</tr>
<tr>
<td>34.29</td>
<td>Fl_Choice.H</td>
<td>1234</td>
</tr>
<tr>
<td>34.30</td>
<td>Fl_Clock.H</td>
<td>1234</td>
</tr>
<tr>
<td>34.31</td>
<td>Fl_ColorChooser.H File Reference</td>
<td>1235</td>
</tr>
<tr>
<td>34.31.1</td>
<td>Detailed Description</td>
<td>1236</td>
</tr>
<tr>
<td>34.32</td>
<td>Fl_ColorChooser.H</td>
<td>1236</td>
</tr>
<tr>
<td>34.33</td>
<td>fl_config.h</td>
<td>1237</td>
</tr>
<tr>
<td>34.34</td>
<td>Fl_Copy_Surface.H</td>
<td>1238</td>
</tr>
<tr>
<td>34.35</td>
<td>Fl_Counter.H</td>
<td>1239</td>
</tr>
<tr>
<td>34.36</td>
<td>Fl_Device.H File Reference</td>
<td>1240</td>
</tr>
<tr>
<td>34.36.1</td>
<td>Detailed Description</td>
<td>1240</td>
</tr>
<tr>
<td>34.37</td>
<td>Fl_Device.H</td>
<td>1240</td>
</tr>
<tr>
<td>34.38</td>
<td>Fl_Dial.H</td>
<td>1241</td>
</tr>
<tr>
<td>34.39</td>
<td>Fl_Double_Window.H</td>
<td>1241</td>
</tr>
<tr>
<td>34.40.1</td>
<td>Detailed Description</td>
<td>1248</td>
</tr>
<tr>
<td>34.40</td>
<td>fl_draw.H File Reference</td>
<td>1242</td>
</tr>
<tr>
<td>34.41.1</td>
<td>Detailed Description</td>
<td>1248</td>
</tr>
<tr>
<td>34.41</td>
<td>fl_draw.H</td>
<td>1248</td>
</tr>
<tr>
<td>34.42</td>
<td>Fl_Export.H</td>
<td>1253</td>
</tr>
<tr>
<td>34.43</td>
<td>Fl_File_Browser.H</td>
<td>1254</td>
</tr>
<tr>
<td>34.44</td>
<td>Fl_File_Chooser.H</td>
<td>1255</td>
</tr>
<tr>
<td>34.45</td>
<td>Fl_File_Icon.H</td>
<td>1257</td>
</tr>
<tr>
<td>34.46</td>
<td>Fl_File_Input.H</td>
<td>1258</td>
</tr>
<tr>
<td>34.47</td>
<td>Fl_Fill_Dial.H</td>
<td>1259</td>
</tr>
<tr>
<td>34.48</td>
<td>Fl_Fill_Slider.H</td>
<td>1259</td>
</tr>
<tr>
<td>34.49</td>
<td>Fl_Flex.H</td>
<td>1260</td>
</tr>
<tr>
<td>34.50</td>
<td>Fl_Float_Input.H</td>
<td>1261</td>
</tr>
<tr>
<td>34.51</td>
<td>Fl_FormsBitmap.H</td>
<td>1262</td>
</tr>
<tr>
<td>34.52</td>
<td>Fl_FormsPixmap.H</td>
<td>1262</td>
</tr>
<tr>
<td>34.53</td>
<td>Fl_Free.H</td>
<td>1263</td>
</tr>
<tr>
<td>34.54</td>
<td>Fl_GIF_Image.H</td>
<td>1263</td>
</tr>
<tr>
<td>34.55</td>
<td>Fl_Gl_Window.H</td>
<td>1264</td>
</tr>
<tr>
<td>34.56</td>
<td>Fl_Graphics_Driver.H</td>
<td>1266</td>
</tr>
<tr>
<td>34.57.1</td>
<td>Detailed Description</td>
<td>1271</td>
</tr>
<tr>
<td>34.57</td>
<td>Fl_Group.H File Reference</td>
<td>1271</td>
</tr>
<tr>
<td>34.58</td>
<td>Fl_Group.H</td>
<td>1271</td>
</tr>
<tr>
<td>Number</td>
<td>File</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>34.59</td>
<td>Fl_Help_Dialog.H</td>
<td>1272</td>
</tr>
<tr>
<td>34.60</td>
<td>Fl_Help_View.H</td>
<td>1273</td>
</tr>
<tr>
<td>34.61</td>
<td>Fl_Hold_Browser.H</td>
<td>1276</td>
</tr>
<tr>
<td>34.62</td>
<td>Fl_Hor_Fill_Slider.H</td>
<td>1277</td>
</tr>
<tr>
<td>34.63</td>
<td>Fl_Hor_Nice_Slider.H</td>
<td>1277</td>
</tr>
<tr>
<td>34.64</td>
<td>Fl_Hor_Slider.H</td>
<td>1278</td>
</tr>
<tr>
<td>34.65</td>
<td>Fl_Hor_Value_Slider.H</td>
<td>1278</td>
</tr>
<tr>
<td>34.66</td>
<td>Fl_ICO_Image.H</td>
<td>1278</td>
</tr>
<tr>
<td>34.67</td>
<td>Fl_Image.H File Reference</td>
<td>1279</td>
</tr>
<tr>
<td></td>
<td>34.67.1 Detailed Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>34.67.2 Enumeration Type Documentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>34.67.2.1 Fl_RGB_Scaling</td>
<td>1280</td>
</tr>
<tr>
<td>34.68</td>
<td>Fl_Image.H</td>
<td>1280</td>
</tr>
<tr>
<td>34.69</td>
<td>Fl_Image_Surface.H</td>
<td>1282</td>
</tr>
<tr>
<td>34.70</td>
<td>Fl_Input.H</td>
<td>1282</td>
</tr>
<tr>
<td>34.71</td>
<td>Fl_Input_.H</td>
<td>1283</td>
</tr>
<tr>
<td>34.72</td>
<td>Fl_Input_Choice.H</td>
<td>1286</td>
</tr>
<tr>
<td>34.73</td>
<td>Fl_Int_Input.H</td>
<td>1288</td>
</tr>
<tr>
<td>34.74</td>
<td>Fl_Int_Vector.H File Reference</td>
<td>1288</td>
</tr>
<tr>
<td></td>
<td>34.74.1 Detailed Description</td>
<td></td>
</tr>
<tr>
<td>34.75</td>
<td>Fl_Int_Vector.H</td>
<td>1289</td>
</tr>
<tr>
<td>34.76</td>
<td>Fl_JPEG_Image.H</td>
<td>1290</td>
</tr>
<tr>
<td>34.77</td>
<td>Fl_Light_Button.H</td>
<td>1290</td>
</tr>
<tr>
<td>34.78</td>
<td>Fl_Line_Dial.H</td>
<td>1291</td>
</tr>
<tr>
<td>34.79</td>
<td>Fl_Menu.H</td>
<td>1291</td>
</tr>
<tr>
<td>34.80</td>
<td>Fl_Menu_.H</td>
<td>1291</td>
</tr>
<tr>
<td>34.81</td>
<td>Fl_Menu_Bar.H</td>
<td>1293</td>
</tr>
<tr>
<td>34.82</td>
<td>Fl_Menu_Buton.H</td>
<td>1293</td>
</tr>
<tr>
<td>34.83</td>
<td>Fl_Menu_Item.H File Reference</td>
<td>1294</td>
</tr>
<tr>
<td></td>
<td>34.83.1 Enumeration Type Documentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>34.83.1.1 anonymous enum</td>
<td>1294</td>
</tr>
<tr>
<td>34.84</td>
<td>Fl_Menu_Item.H</td>
<td>1294</td>
</tr>
<tr>
<td>34.85</td>
<td>Fl_Menu_Window.H</td>
<td>1295</td>
</tr>
<tr>
<td>34.86</td>
<td>fl_message.H</td>
<td>1297</td>
</tr>
<tr>
<td>34.87</td>
<td>Fl_Multi_Browser.H</td>
<td>1298</td>
</tr>
<tr>
<td>34.88</td>
<td>Fl_Multi_Label.H</td>
<td>1298</td>
</tr>
<tr>
<td>34.89</td>
<td>Fl_Multiline_Input.H</td>
<td>1299</td>
</tr>
<tr>
<td>34.90</td>
<td>Fl_Multiline_Output.H</td>
<td>1299</td>
</tr>
<tr>
<td>34.91</td>
<td>Fl_Native_File_Chooser.H File Reference</td>
<td>1299</td>
</tr>
<tr>
<td></td>
<td>34.91.1 Detailed Description</td>
<td></td>
</tr>
<tr>
<td>34.92</td>
<td>Fl_Native_File_Chooser.H</td>
<td>1300</td>
</tr>
<tr>
<td>34.93</td>
<td>Fl_Nice_Slider.H</td>
<td>1300</td>
</tr>
</tbody>
</table>
34.127.3 Function Documentation ......................................................... 1326
  34.127.3.1 fl_register_images() .................................................. 1326
34.128 Fl_Shared_Image.H ................................................................. 1326
34.129 Fl_Shortcut_Button.H .............................................................. 1327
34.130 fl_show_colormap.H File Reference ............................................. 1328
  34.130.1 Detailed Description .......................................................... 1328
34.131 fl_show_colormap.H ................................................................. 1328
34.132 fl_show_input.H ........................................................................ 1328
34.133 Fl_Simple_Counter.H .................................................................. 1329
34.134 Fl_Simple_Terminal.H .................................................................. 1329
34.135 Fl_Single_Window.H ..................................................................... 1331
34.136 Fl_Slider.H .................................................................................. 1331
34.137 Fl_Spinner.H ............................................................................... 1332
34.138 Fl_String.H File Reference ......................................................... 1334
  34.138.1 Detailed Description .............................................................. 1334
    34.138.2 Function Documentation .................................................... 1334
      34.138.2.1 operator+() [1/2] ......................................................... 1334
      34.138.2.2 operator+() [2/2] ......................................................... 1334
      34.138.2.3 operator==() ................................................................ 1335
34.139 Fl_String.H .................................................................................. 1335
34.140 fl_string_functions.h File Reference .............................................. 1336
    34.140.1 Detailed Description .......................................................... 1336
34.141 fl_string_functions.h .................................................................... 1337
34.142 Fl_SVG_File_Surface.H ............................................................... 1337
34.143 Fl_SVG_Image.H .......................................................................... 1338
34.144 Fl_Sys_Menu_Bar.H File Reference ................................................ 1338
    34.144.1 Detailed Description .......................................................... 1339
34.145 Fl_Sys_Menu_Bar.H ...................................................................... 1339
34.146 Fl_Table.H ................................................................................... 1340
34.147 Fl_Table_Row.H ........................................................................... 1345
34.148 Fl_Tabs.H ...................................................................................... 1347
34.149 Fl_Text_Buffer.H ........................................................................... 1348
34.150 Fl_Text_Display.H ...................................................................... 1348
34.151 Fl_Text_Editor.H ......................................................................... 1352
34.152 Fl_Tile.H ....................................................................................... 1357
34.153 Fl_Tiled_Image.H ........................................................................... 1358
34.154 Fl_Timer.H .................................................................................... 1359
34.155 Fl_Toggle_Button.H ...................................................................... 1359
34.156 Fl_Toggle_Light_Button.H ........................................................... 1360
34.157 Fl_Toggle_Round_Button.H .......................................................... 1361
34.158 Fl_Tooltip.H .................................................................................. 1361
34.159 Fl_Tree.H File Reference ............................................................. 1362
34.182 FI_XPM_Image.H .......................................................... 1395
34.183 forms.H ................................................................. 1396
34.184 gl.h File Reference ..................................................... 1405
   34.184.1 Detailed Description ............................................ 1407
   34.184.2 Function Documentation ....................................... 1407
      34.184.2.1 gl_color() ................................................ 1407
      34.184.2.2 gl_draw() [1/7] ......................................... 1407
      34.184.2.3 gl_draw() [2/7] ......................................... 1407
      34.184.2.4 gl_draw() [3/7] ......................................... 1407
      34.184.2.5 gl_draw() [4/7] ......................................... 1408
      34.184.2.6 gl_draw() [5/7] ......................................... 1408
      34.184.2.7 gl_draw() [6/7] ......................................... 1408
      34.184.2.8 gl_draw() [7/7] ......................................... 1408
      34.184.2.9 gl_font() .............................................. 1408
      34.184.2.10 gl_rect() ............................................ 1409
      34.184.2.11 gl_rectf() .......................................... 1409
      34.184.2.12 gl_texture_pile_height() [1/2] ...................... 1409
      34.184.2.13 gl_texture_pile_height() [2/2] ...................... 1409
34.185 gl.h ................................................................. 1409
34.186 gl2opengl.h .......................................................... 1410
34.187 gl_draw.H ............................................................ 1411
34.188 glu.h ............................................................... 1411
34.189 glut.H ............................................................... 1412
34.190 mac.H File Reference ............................................... 1417
   34.190.1 Detailed Description ......................................... 1418
34.191 mac.h .............................................................. 1418
34.192 math.h .............................................................. 1420
34.193 names.h File Reference ............................................. 1420
   34.193.1 Detailed Description ....................................... 1421
34.194 names.h ............................................................ 1421
34.195 platform.H .......................................................... 1422
34.196 platform_types.h File Reference ................................... 1423
   34.196.1 Detailed Description ....................................... 1423
   34.196.2 Typedef Documentation ..................................... 1424
      34.196.2.1 fl_inptr_t ........................................... 1424
      34.196.2.2 FI_Offscreen ....................................... 1424
      34.196.2.3 FI_Region .......................................... 1424
      34.196.2.4 FI_Timestamp ...................................... 1424
      34.196.2.5 fl_uintptr_t ....................................... 1424
      34.196.2.6 GLContext .......................................... 1425
34.197 platform_types.h .................................................. 1425
34.198 wayland.H File Reference ......................................... 1426
34.198.1 Detailed Description .................................................. 1427
34.198.2 Function Documentation ............................................. 1427
  34.198.2.1 fl_wl_compositor() ............................................. 1427
34.199 wayland.H ............................................................... 1427
34.200 win32.H File Reference ................................................ 1427
  34.200.1 Detailed Description ............................................. 1428
34.201 win32.H ................................................................. 1428
34.202 x.H ........................................................................... 1429
34.203 x11.H File Reference ..................................................... 1429
  34.203.1 Detailed Description ............................................. 1430
  34.203.2 Function Documentation ........................................... 1430
    34.203.2.1 fl_x11_find() .................................................... 1430
    34.203.2.2 fl_x11_gc() ...................................................... 1430
    34.203.2.3 fl_x11_xid() ..................................................... 1430
34.204 x11.H ....................................................................... 1430
34.205 cgdebug.h .................................................................. 1431
34.206 fastarrow.h .................................................................. 1434
34.207 Fl.cxx File Reference ..................................................... 1434
  34.207.1 Detailed Description ............................................. 1435
  34.207.2 Function Documentation ........................................... 1435
    34.207.2.1 fl_find() .......................................................... 1435
    34.207.2.2 fl_open_display() .............................................. 1435
  34.207.3 Variable Documentation ........................................... 1435
    34.207.3.1 fl_disable_wayland ........................................... 1435
34.208 fl_arc.cxx File Reference ................................................ 1436
34.209 fl_ask.cxx File Reference ................................................ 1436
34.210 fl_boxtype.cxx File Reference .......................................... 1438
  34.210.1 Detailed Description ............................................. 1439
  34.210.2 Function Documentation ........................................... 1439
    34.210.2.1 fl_internal_boxtype() ....................................... 1439
    34.210.2.2 fl_rectbound() .................................................. 1439
34.211 fl_cmap.h ..................................................................... 1439
34.212 fl_color.cxx File Reference ............................................. 1443
  34.212.1 Detailed Description ............................................. 1443
  34.212.2 Variable Documentation ........................................... 1443
    34.212.2.1 fl_cmap ............................................................ 1443
34.213 Fl_compose.cxx File Reference ......................................... 1443
34.214 fl_contrast.cxx File Reference ......................................... 1443
  34.214.1 Detailed Description ............................................. 1444

Generated by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.240.1</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>34.240.1.1</td>
<td>fl_casenumericsort()</td>
</tr>
<tr>
<td>34.240.1.2</td>
<td>fl_numericsort()</td>
</tr>
<tr>
<td>34.241</td>
<td>print_button.h</td>
</tr>
<tr>
<td>34.242</td>
<td>print_panel.h</td>
</tr>
<tr>
<td>34.243</td>
<td>slowarrow.h</td>
</tr>
<tr>
<td>34.244</td>
<td>utf8_internal.h</td>
</tr>
<tr>
<td>34.245</td>
<td>vsnprintf.c File Reference</td>
</tr>
<tr>
<td>34.245.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>34.245.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>34.245.2.1</td>
<td>fl_vsnprintf()</td>
</tr>
<tr>
<td>34.246</td>
<td>Xutf8.h</td>
</tr>
<tr>
<td>34.247</td>
<td>case.h</td>
</tr>
<tr>
<td>34.248</td>
<td>dingbats_.h</td>
</tr>
<tr>
<td>34.249</td>
<td>spacing.h</td>
</tr>
<tr>
<td>34.250</td>
<td>symbol_.h</td>
</tr>
<tr>
<td>34.251</td>
<td>armscii_8.h</td>
</tr>
<tr>
<td>34.252</td>
<td>ascii.h</td>
</tr>
<tr>
<td>34.253</td>
<td>big5.h</td>
</tr>
<tr>
<td>34.254</td>
<td>big5_emacs.h</td>
</tr>
<tr>
<td>34.255</td>
<td>cp1133.h</td>
</tr>
<tr>
<td>34.256</td>
<td>cp1251.h</td>
</tr>
<tr>
<td>34.257</td>
<td>cp1255.h</td>
</tr>
<tr>
<td>34.258</td>
<td>cp1256.h</td>
</tr>
<tr>
<td>34.259</td>
<td>cp936ext.h</td>
</tr>
<tr>
<td>34.260</td>
<td>gb2312.h</td>
</tr>
<tr>
<td>34.261</td>
<td>georgian_academy.h</td>
</tr>
<tr>
<td>34.262</td>
<td>georgian_ps.h</td>
</tr>
<tr>
<td>34.263</td>
<td>iso8859_1.h</td>
</tr>
<tr>
<td>34.264</td>
<td>iso8859_10.h</td>
</tr>
<tr>
<td>34.265</td>
<td>iso8859_11.h</td>
</tr>
<tr>
<td>34.266</td>
<td>iso8859_13.h</td>
</tr>
<tr>
<td>34.267</td>
<td>iso8859_14.h</td>
</tr>
<tr>
<td>34.268</td>
<td>iso8859_15.h</td>
</tr>
<tr>
<td>34.269</td>
<td>iso8859_16.h</td>
</tr>
<tr>
<td>34.270</td>
<td>iso8859_2.h</td>
</tr>
<tr>
<td>34.271</td>
<td>iso8859_3.h</td>
</tr>
<tr>
<td>34.272</td>
<td>iso8859_4.h</td>
</tr>
<tr>
<td>34.273</td>
<td>iso8859_5.h</td>
</tr>
<tr>
<td>34.274</td>
<td>iso8859_6.h</td>
</tr>
<tr>
<td>34.275</td>
<td>iso8859_7.h</td>
</tr>
<tr>
<td>34.276</td>
<td>iso8859_8.h</td>
</tr>
<tr>
<td>34.277</td>
<td>iso8859_9.h</td>
</tr>
<tr>
<td>34.278</td>
<td>iso8859_9e.h</td>
</tr>
<tr>
<td>34.279</td>
<td>jisx0201.h</td>
</tr>
<tr>
<td>34.280</td>
<td>jisx0208.h</td>
</tr>
<tr>
<td>34.281</td>
<td>jisx0212.h</td>
</tr>
<tr>
<td>34.282</td>
<td>koi8_c.h</td>
</tr>
<tr>
<td>34.283</td>
<td>koi8_r.h</td>
</tr>
<tr>
<td>34.284</td>
<td>koi8_u.h</td>
</tr>
<tr>
<td>34.285</td>
<td>ksc5601.h</td>
</tr>
<tr>
<td>34.286</td>
<td>mulelao.h</td>
</tr>
<tr>
<td>34.287</td>
<td>tatar_cyr.h</td>
</tr>
<tr>
<td>34.288</td>
<td>tcvn.h</td>
</tr>
<tr>
<td>34.289</td>
<td>tis620.h</td>
</tr>
<tr>
<td>34.290</td>
<td>ucs2be.h</td>
</tr>
<tr>
<td>34.291</td>
<td>utf8.h</td>
</tr>
<tr>
<td>34.292</td>
<td>viscii.h</td>
</tr>
<tr>
<td>34.293</td>
<td>Ximint.h</td>
</tr>
<tr>
<td>34.294</td>
<td>Xlibint.h</td>
</tr>
</tbody>
</table>

**Index** 1817
<table>
<thead>
<tr>
<th>Preface</th>
<th>Adding and Extending Widgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to FLTK</td>
<td>Using OpenGL</td>
</tr>
<tr>
<td>FLTK Basics</td>
<td>Programming with FLUID</td>
</tr>
<tr>
<td>Common Widgets and Attributes</td>
<td></td>
</tr>
<tr>
<td>- Colors</td>
<td>- GUI Attributes</td>
</tr>
<tr>
<td>- Box Types</td>
<td>- Selecting and Moving Widgets</td>
</tr>
<tr>
<td>- Labels and Label Types</td>
<td>- Image Labels</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>- Forms Compatibility</td>
</tr>
<tr>
<td>- Offscreen Drawing</td>
<td>- Operating System Issues</td>
</tr>
<tr>
<td>Handling Events</td>
<td>- Migrating Code from FLTK 1.3 to 1.4</td>
</tr>
<tr>
<td>- Fl::event_+() methods</td>
<td>- Software License</td>
</tr>
<tr>
<td>- Event Propagation</td>
<td>- Example Source Code</td>
</tr>
<tr>
<td></td>
<td>- FAQ (Frequently Asked Questions)</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/Unix: Wayland. FLTK applications under Linux/Unix run, unchanged, as Wayland or X11 clients depending on what's available at run-time.

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

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

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
- FLTK Runtime Options
- Advanced FLTK
- Unicode and UTF-8 Support
- Constants and Enumerations
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 - 2023 by Bill Spitzak and others. Use and distribution of FLTK is governed by the GNU Library General Public License with 4 exceptions, located in Software License.

UNIX is a registered trademark of the X Open Group, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. OpenGL is a registered trademark of Silicon Graphics, Inc. Apple, Macintosh, MacOS, and Mac OS X are registered trademarks of Apple Computer, Inc.
Chapter 3

Introduction to FLTK

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

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

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
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.

Generated by Doxygen
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, Cairo, Windows, or macOS Quartz.

- OpenGL/Mesa drawing area widget.

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

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

- Compatibility header file for the GLUT library.

- Compatibility header file for the XForms library.

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/Wayland platform

--enable-shared
Enable generation of shared libraries

--enable-threads
Enable multithreading support

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

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

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

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

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

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

--enable-cairoext
Enable the FLTK instrumentation for cairo extended use (requires --enable-cairo).
3.7 Building FLTK Under Microsoft Windows

--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

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

NNTP Newsgroups

- [https://groups.google.com/forum/#!forum/fltkgeneral](https://groups.google.com/forum/#!forum/fltkgeneral) [Google Groups interface]
- [https://www.fltk.org/newsgroups.php](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 https://www.fltk.org/bugs.php, and click on "Submit Bug or Feature Request".

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

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

FLTK Basics

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

4.1 Writing Your First FLTK Program

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

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

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

Listing 1 - "hello.cxx"

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

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

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

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

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

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

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

```c++
box->box(FL_UP_BOX);
box->labelfont(FL_BOLD + FL_ITALIC);
box->labelsize(36);
box->labeltype(FL_SHADOW_LABEL);
```
We tell FLTK that we will not add any more widgets to `window`.
```
window->end();
```

Finally, we show the window and enter the FLTK event loop:
```
window->show(argc, argv);
return Fl::run();
```

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

![Hello, World! Window](image)

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

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

### 4.1.1 Creating the Widgets

The widgets are created using the C++ `new` operator. For most widgets the arguments to the constructor are:
```
Fl_Widget(x, y, width, height, label)
```

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

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

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

### 4.1.2 Creating Widget Hierarchies

Widgets are commonly ordered into functional groups, which in turn may be grouped again, creating a hierarchy of widgets. FLTK makes it easy to fill groups by automatically adding all widgets that are created between a `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(...)`. 

Generated by Doxygen
4.1 Writing Your First FLTK Program

4.1.3 Get/Set Methods

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

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

4.1.4 Redrawing After Changing Attributes

Almost all of the 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 ...
```

**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).
4.4 Compiling Programs that Use FLTK

4.4.2 Compiling Programs with the 'fltk-config' Script

The fltk-config script included with FLTK can be used on systems with a Posix compliant shell, for instance Unix/Linux, macOS, Windows with MinGW, MSYS2, or Cygwin.

Note

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

fltk-config --help

displays all available options.

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

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.

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

Note

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

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

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'
c++ ... 'fltk-config --use-forms --use-gl --use-images --ldflags'

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

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

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

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

Note

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

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

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

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

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

where

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

The final commandline is composed like this example:

```
$ fltk-config --compile main.cxx button.o -Wextra x1.a --link -L/usr/include/cairo/ -lcairo
g++ {fltk-flags} -o main -Wextra main.cxx button.o x1.a {fltk-libs} -L/usr/include/cairo/ -lcairo
```

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

4.4.4 Compiling Programs with Makefiles

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

```
CXX = $(shell fltk-config --cxx)
DEBUG = -g
CXXFLAGS = $(shell fltk-config --use-gl --use-images --cxxflags) -I.
LDFLAGS = $(shell fltk-config --use-gl --use-images --ldflags)
LDSTATIC = $(shell fltk-config --use-gl --use-images --ldstaticflags)
LINK = $(CXX)
TARGET = cube
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)
$(TARGET): $(OBJJS)
  $(CXX) $(CXXFLAGS) $(DEBUG) -o $(TARGET) $(OBJJS)
   $(TARGET): $(OBJJS)
   rm -f *.o 2>/dev/null
   rm -f $(TARGET) 2>/dev/null
```

Generated by Doxygen
4.4 Compiling Programs that Use FLTK

4.4.5 Compiling Programs with Microsoft Visual C++

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

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

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

Note

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

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

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

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

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

```
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:

```
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.

Generated by Doxygen
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:

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

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

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

5.3 Valuators

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

- **Fl_Counter** - A widget with arrow buttons that shows the current value.
- **Fl_Dial** - A round knob.
- **Fl_Roller** - An SGI-like dolly widget.
- **Fl_Scrollbar** - A standard scrollbar widget.
- **Fl_Slider** - A scrollbar with a knob.
- **Fl_Value_Slider** - A slider that shows the current value.
The `value()` method gets and sets the current value of the widget. The `minimum()` and `maximum()` methods set the range of values that are reported by the widget.

5.4 Groups

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

- **FL_Double_Window** - A double-buffered window on the screen.
- **FL_Gl_Window** - An OpenGL window on the screen.
- **FL_Group** - The base container class; can be used to group any widgets together.
- **FL_Pack** - A collection of widgets that are packed into the group area.
- **FL_Scroll** - A scrolled window area.
- **FL_Tabs** - Displays child widgets as tabs.
- **FL_Tile** - A tiled window area.
- **FL_Window** - A window on the screen.
- **FL_Wizard** - Displays one group of widgets at a time.
5.5 Setting the Size and Position of Widgets

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

You can change the size and position by using the \texttt{position()}, \texttt{resize()}, \texttt{and 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 \textit{not} the X or MS Windows colormap, but instead is an internal table with fixed contents.

See the \texttt{Colors} section of \texttt{Drawing Things in FLTK} for implementation details.

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

\begin{itemize}
  \item FL\_BLACK
  \item FL\_RED
  \item FL\_GREEN
  \item FL\_YELLOW
  \item FL\_BLUE
  \item FL\_MAGENTA
  \item FL\_CYAN
  \item FL\_WHITE
\end{itemize}

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

\begin{itemize}
  \item FL\_FOREGROUND\_COLOR
  \item FL\_BACKGROUND\_COLOR
  \item FL\_INACTIVE\_COLOR
  \item FL\_SELECTION\_COLOR
\end{itemize}

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 RGBA, so it is also possible to specify a color using a hex constant as a color map index:

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

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

\begin{verbatim}
  button->color(0xff000000);  // RGB: red
  button->color(0x00ff0000);  // RGB: green
  button->color(0x0000ff00);  // RGB: blue
  button->color(0xffffff00);  // RGB: white
\end{verbatim}
Note
If TrueColor is not available, any RGB colors will be set to the nearest entry in the colormap.

5.7 Box Types

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

These are the standard box types included with FLTK:

![FLTK Standard Box Types](image)

**Figure 5.3 FLTK Standard Box Types**

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

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()`, `labelsiz()`e, `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.
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_Icon`) associated with the text.
- `FL_IMAGE_LABEL` - draws the image (`Fl_Icon`) 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_Item` 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_Icon`.

Making Your Own Label Types

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

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

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

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

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

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

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

Adding Your Label Type

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

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

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

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

Making your own symbols

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

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

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

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

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

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

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

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

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

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

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

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

Note:

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

Hint:

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

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

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

5.10 Shortcuts

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

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

The shortcut value is the key event value - the ASCII value or one of the special keys described in Fl::event_key() Values combined with any modifiers like Shift, Alt, and Control.
# Chapter 6

## 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 \texttt{Fl_Window} and \texttt{Fl_Group} container widgets.

This position is always relative to the enclosing \texttt{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.

![FLTK coordinate system](image)

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:

```
<table>
<thead>
<tr>
<th>Fl_Window main window</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Window TL subwindow</td>
</tr>
<tr>
<td>Fl_Box tl box</td>
</tr>
<tr>
<td>Fl_Window BR subwindow</td>
</tr>
<tr>
<td>Fl_Box br box</td>
</tr>
<tr>
<td>Fl_Group TR group</td>
</tr>
<tr>
<td>Fl_Box tr box</td>
</tr>
<tr>
<td>Fl_Group BL group</td>
</tr>
<tr>
<td>Fl_Box bl box</td>
</tr>
</tbody>
</table>
```

Generated by Doxygen
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.
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.

Figure 6.3 Fl_Scroll container widget

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.

Figure 6.4 Fl_Tabs container widget

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?.

Generated by Doxygen
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.
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:

```cpp
    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:

```cpp
    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:
How Does Resizing Work?

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
btns->resizable(R); // make *R* parent group resizable
btns->end();
dialog->resizable(text);
dialog->end();
```

Imagine instead that you have a group that has a button, an input field, another button and a second input field, all next to each other, and you want the input fields to resize equally, but not the buttons. How could you achieve this?
Setting either of the input fields to be the **resizable** leaves the other one fixed, as shown below:

![Figure 7.6 Resizing input fields example (b)](image)

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 Resizing input fields example (b)](image)

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 Resizing a row of widgets (a)](image)

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

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

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

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

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

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

![Figure 7.9 Resizing a row of widgets (b)](image)

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

![Figure 7.10 Resizing a row of widgets (c)](image)

To summarize the key points of the new technique:

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

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

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 },
    { "New &View", FL_ALT + 'v', (Fl_Callback *)view_cb, 0 },
    { "Close View", FL_COMMAND + 'w', (Fl_Callback *)close_cb, 0, FL_MENU_DIVIDER },
    { "Exit", FL_COMMAND + 'q', (Fl_Callback *)quit_cb, 0 },
    { 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 },
    { 0 },
    { "Search", 0, 0, 0, FL_SUBMENU },
    { "Find...", FL_COMMAND + 'f', (Fl_Callback *)find_cb },
    { "Find Again", FL_COMMAND + 'g', find2_cb },
    { "Replace...", FL_COMMAND + 'r', replace_cb },
    { "Replace Again", FL_COMMAND + 't', replace2_cb },
    { 0 },
    { 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:

![Figure 8.1 The search and replace dialog](image)

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

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

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

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.\nWould 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 \"%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 \"%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(argc, 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:

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

or by using the fltk-config script with:

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

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

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

8.11 The Final Product

The final editor window should look like this:

![Figure 8.2 The completed editor window](image-url)
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
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 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;
    }
```
// 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] = '');
    stylebuf->replace(pos, pos + nDeleted, style);
    delete[] style;
} else {
    // Just delete characters in the style buffer...
    stylebuf->remove(pos, pos + nDeleted);
}

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

The style_parse() function scans a copy of the text in the buffer and generates the necessary style characters for display. It assumes that parsing begins at the start of a line:
//
// 'style_parse()' - Parse text and produce style data.
//
void
style_parse(const char *text,
            char *style,
            int length) {
char current;
int col;
int last;
char buf[255],
    *bufptr;
const char *temp;
for (current = *style, col = 0, last = 0; length > 0; length--, text++) {
    if (current == 'A') {
        // Check for directives, comments, strings, and keywords...
    } else if (strncmp(text, "//", 2) == 0) {
        current = 'B';
    } else if (strncmp(text, "/*", 2) == 0) {
        current = 'C';
    } else if (strncmp(text, "\\", 2) == 0) {
        // Quoted quote...
        *style++ = current;
        *style++ = current;
    } else if (*text == ')
        current = 'D';
    } else if (!last && islower(*text)) {
        // Might be a keyword...
        for (temp = text, bufptr = buf;
            islower(*temp) && bufptr < (buf + sizeof(buf) - 1);
            bufptr++, *temp++);
    } else if (*text == '
        current = 'E';
    } else if (!islower(*temp)) {
        continue;
    } else if (*text == '"
        current = 'F';
    } else if (!islower(*temp)) {
        continue;
    }
}
bufptr = '0';
bufptr = buf;
if (bsearch(bufptr, code_types,
    sizeof(code_types) / sizeof(code_types[0]),
    sizeof(code_types[0]), compare_keywords)) {
    while (text < temp) {
        *style++ = 'F';
        text++;
        length--;
        col++;
    }
    text--;
    length++;
    last = 1;
    continue;
} else if (bsearch(bufptr, code_keywords,
    sizeof(code_keywords) / sizeof(code_keywords[0]),
    sizeof(code_keywords[0]), compare_keywords)) {
    while (text < temp) {
        *style++ = 'G';
        text++;
        length--;
        col++;
    }
    text--;
    length++;
    last = 1;
    continue;
}
} else if (current == 'C' &&strncmp(text, "*/", 2) == 0) {
    // Close a C comment...
    *style++ = current;
    *style++ = current;
    text++;
    length--;
    current = 'A';
    col = 2;
    continue;
} else if (current == 'D') {
    // Continuing in string...
    if (strncmp(text, "\\\", 2) == 0) {
        // Quoted end quote...
        *style++ = current;
        *style++ = current;
        text++;
        length--;
        col = 2;
        continue;
    } else if (*text == '\') {
        // End quote...
        *style++ = current;
        col++;
        current = 'A';
        continue;
    }
    // Copy style info...
    if (current == 'A' && (*text == '{' || *text == '}')) *style++ = 'G';
    else *style++ = current;
    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 `labelltypes`. 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 $v \times$ scale units. Thus, a window with width 500 units is 500 scale pixels wide, a line of length 10 units is 10 scale pixels long, lines of text written using a 14-point font are 14 scale pixels below each other. Consider a system with two screens, one with regular DPI and one with a twice higher DPI. If the first screen's scale factor is set to 1 and that of the second screen to 2, the GUI of any FLTK app appears equally sized on the two screens.

FLTK uses several units to measure graphical elements:

- All data visible by the public API (e.g., window widths, line lengths, font sizes, clipping regions) are in FLTK units which are both system- and DPI-independent.

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

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

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

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

void fl_push_no_clip()

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

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.

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.

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.

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 Drawing Functions

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)

**Figure 9.1 FLTK default colormap (Fl_Color 0x00 - 0xff)**

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.

**Fl\_Color fl\_color()**

Returns the last color that was set using *fl\_color*. This can be used for state save/restore.

**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.

**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.

**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.

**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(0x00000000)
-------- |
 | | Color map index (8 bits)
 Must be zero

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

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

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

button->color(0xff000000); // RGB: red
button->color(0x00ff0000); // RGB: green
button->color(0x0000ff00); // RGB: blue
button->color(0xffffffff); // 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  
• FL_CAP_ROUND  
• FL_CAP_SQUARE (extends past end point 1/2 line width)  
• FL_JOIN_MITER (pointed)  
• FL_JOIN_ROUND  
• FL_JOIN_BEVEL (flat)

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

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

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

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).

void fl_point(int x, int y)

Draw a single pixel at the given coordinates.

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

Color a rectangle that exactly fills the given bounding box.

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

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

Draw a 1-pixel border inside this bounding box.

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

Draw an outlined or filled rectangle with rounded corners.

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

Draw one or two lines between the given points.

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

Outline a 3 or 4-sided polygon with lines.

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

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

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

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

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

Draw vertical and horizontal lines. A vertical line is drawn first, then a horizontal, then a vertical.
Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and MS Windows. The advantage over using \texttt{fl\_arc()} with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3'oclock and are the starting and ending angle of the arc, a2 must be greater or equal to a1.

\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 to the other \texttt{fl\_arc()} function described later in this chapter.

\texttt{fl\_pie()} draws a filled-in pie slice. This slice may extend outside the line drawn by \texttt{fl\_arc()}; to avoid this use \texttt{w-1} and \texttt{h-1}.

\texttt{void fl\_scroll(int X, int Y, int W, int H, int dx, int dy, void (draw\_area)\textbf{)}(void, int,int,int), void* data)}

Scroll a rectangle and draw the newly exposed portions. The contents of the rectangular area is first shifted by \texttt{dx} and \texttt{dy} FLTK units. The callback is then called for every newly exposed rectangular area,
9.3 Drawing Functions

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.

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

Set the current transformation.

void fl_push_matrix()  void fl_pop_matrix()

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

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

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

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

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

void fl_begin_points()  void fl_end_points()

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

void fl_begin_line()  void fl_end_line()
Start and end drawing lines.

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

Start and end drawing a closed sequence of lines.

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

Start and end drawing a convex filled polygon.

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

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

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

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

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

Add a single vertex to the current path.

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

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

```c
void fl_arc(double x, double y, double r, double start, double end)
```
Add a series of points to the current path on the arc of a circle; you can get elliptical paths by using scale and rotate before calling \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.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fl_arc.png}
\caption{fl\_arc\((x,y,r,a_1,a_2)\)}
\end{figure}

\begin{verbatim}
void fl\_circle(double x, double y, double r)
\end{verbatim}

\texttt{fl\_circle\((x,y,r)\)} is equivalent to \texttt{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 \texttt{fl\_arc()}.

\textbf{Note:} \texttt{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.

\begin{verbatim}
void fl\_draw(const char*, int x, int y)
void fl\_draw(const char*, int n, int x, int y)
\end{verbatim}

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 \texttt{fl\_descent()} from \(y\). To align to the top, subtract \texttt{fl\_descent()} and add \texttt{fl\_height()}. This version of \texttt{fl\_draw()} provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

\begin{verbatim}
void fl\_rtl\_draw(const char* str, int n, int x, int y)
\end{verbatim}
Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.

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

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

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

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

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

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

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

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

See also

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

```c
int fl_height()
```

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

See also

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

\begin{verbatim}
int fl_descent()
\end{verbatim}

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

\begin{verbatim}
double fl_width(const char* txt)
double fl_width(const char* txt, int n)
double fl_width(unsigned int unicode_char)
\end{verbatim}

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

See also

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

\begin{verbatim}
void fl_text_extents(const char* txt, int& dx, int& dy, int& w, int& h)
\end{verbatim}

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

\begin{verbatim}
const char* fl_shortcut_label(int shortcut)
\end{verbatim}

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: FL_HELVETICA, FL_TIMES, FL_COURIER, and modifier values FL_BOLD and FL_ITALIC which can be added to these, and FL_SYMBOL and FL_ZAPF_DINGBATS. Faces greater than 255 cannot be used in Fl_Widget labels, since Fl_Widget stores the index as a byte.

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

void fl_font(int face, int size)

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

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

int fl_font()
int fl_size()

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

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.

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

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

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

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

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

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

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

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

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

To use an XPM, do:

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

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

```
int fl_measure_pixmap(char const* data, int &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 Drawing Images

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.

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.

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.

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);
```

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

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

**void** **fl_delete_offscreen**(Fl_Offscreen)

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

**void** **fl_begin_offscreen**(Fl_Offscreen)

Send all subsequent drawing commands to this offscreen buffer.

**void** **fl_end_offscreen**()

Quit sending drawing commands to this offscreen buffer.

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

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

**void** **fl_rescale_offscreen**(Fl_Offscreen &osrc)

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

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.
10.8 Other events

10.7.1 Dropped filenames

Files are received as a list of full path and file names.

- On some X11 platforms, files are received as a URL-encoded UTF-8 string, that is, non-ASCII bytes (and a few others such as space and %) are replaced by the 3 bytes "%XY" where XY are the byte's hexadecimal value. The fl_decode_uri() function can be used to transform in-place the received string into a proper UTF-8 string. On these platforms, strings corresponding to dropped files are further prepended by `file://` (or other prefixes such as `computer://`).
- Other X11 situations put all dropped filenames in a single line, separated by spaces.
- On non-X11 platforms, including Wayland, files dropped are received one pathname per line, with no `\n` after the last pathname.

10.7.2 FL_DND_ENTER

The mouse has been moved to point at this widget. A widget that is interested in receiving drag’n’drop data must return 1 to receive FL_DND_DRAG, FL_DND_LEAVE and FL_DND_RELEASE events.

10.7.3 FL_DND_DRAG

The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag’n’drop data should indicate the possible drop position.

10.7.4 FL_DND_LEAVE

The mouse has moved out of the widget.

10.7.5 FL_DND_RELEASE

The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following FL_PASTE event.

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_LOWER_RELEASE, FL_KEYBOARD, FL_SHORTCUT, FL_UNFOCUS, and FL_LEAVE) directly to leaf widgets. These procedures control those leaf widgets:

- Fl::add_handler()
- Fl::belowmouse()
- Fl::focus()
FLTK propagates events along the widget hierarchy depending on the kind of event and the status of the UI. Some events are injected directly into the widgets, others may be resent as new events to a different group of receivers.

Mouse click events are first sent to the window that caused them. The window then forwards the event down the hierarchy until it reaches the widget that is below the click position. If that widget uses the given event, the widget is marked "pushed" and will receive all following mouse motion (FL_DRAG) events until the mouse button is released.

Mouse motion (FL_MOVE) events are sent to the Fl::belowmouse() widget, i.e. the widget that returned 1 on the last FL_ENTER event.

Mouse wheel events are sent to the window that caused the event. The window propagates the event down the tree, first to the widget that is below the mouse pointer, and if that does not succeed, to all other widgets in the group. This ensures that scroll widgets work as expected with the widget furthest down in the hierarchy getting the first opportunity to use the wheel event, but also giving scroll bars, that are not directly below the mouse a chance.

Keyboard events are sent directly to the widget that has keyboard focus. If the focused widget rejects the event, it is resent as a shortcut event, first to the top-most window, then to the widget below the mouse pointer, propagating up the hierarchy to all its parents. Those send the event also to all widgets that are not below the mouse pointer. Now if that did not work out, the shortcut is sent to all registered shortcut handlers.

If we are still unsuccessful, the event handler flips the case of the shortcut letter and starts over. Finally, if the key is “escape”, FLTK sends a close event to the top-most window.

All other events are pretty much sent right away to the window that created the event.

Widgets can "grab" events. The grabbing window gets all events exclusively, but usually by the same rules as described above.

Windows can also request exclusivity in event handling by making the window modal.

### 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...
    }
Adding and Extending Widgets

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 $draw()$. 

11.4 Protected Methods of Fl_Widget

The following methods are provided for subclasses to use:

- clear_visible()
- damage()
- draw_box()
- draw_focus()
- draw_label()
- set_flag()
- set_visible()
- test_shortcut()
- type()
When redrawing your widgets you should look at the damage bits to see what parts of your widget need redrawing. The `handle()` method can then set individual damage bits to limit the amount of drawing that needs to be done:

```cpp
MyClass::handle(int event) {
    ...
    if (change_to_part1) damage(1);
    if (change_to_part2) damage(2);
    if (change_to_part3) damage(4);
}
MyClass::draw() {
    if (!damage() & FL_DAMAGE_ALL) {
        ... draw frame/box and other static stuff ...
    }
    if (!damage() & (FL_DAMAGE_ALL | 1)) draw_part1();
    if (!damage() & (FL_DAMAGE_ALL | 2)) draw_part2();
    if (!damage() & (FL_DAMAGE_ALL | 4)) draw_part3();
}
```

**Todo** Clarify `Fl_Window::damage(uchar)` handling - seems confused/wrong? ORing value doesn't match setting behavior in `Fl_Widget.H`.

### Protected Methods of `Fl_Widget`

#### void Fl_Widget::draw_box() const

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

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

#### void Fl_Widget::draw_focus() const

Draws a focus box inside the widget's bounding box. The second form allows you to specify a different bounding box.

```cpp
void Fl_Widget::draw_focus()
void Fl_Widget::draw_focus(Fl_Boxtype t, int x, int y, int w, int h) const
```

#### void Fl_Widget::draw_label() const

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

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

The third form draws the label anywhere. It acts as though `FL_ALIGN_INSIDE` has been forced on so the label will appear inside the passed bounding box. This is designed for parent groups to draw labels with.

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

```c
void Fl_Widget::set_visible()
void Fl_Widget::clear_visible()
```

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

```c
int Fl_Widget::test_shortcut()
static int Fl_Widget::test_shortcut(const char *s)
```

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

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

```c
uchar Fl_Widget::type() const
void Fl_Widget::type(uchar t)
```

The property `Fl_Widget::type()` can return an arbitrary 8-bit identifier, and can be set with the protected method `type(uchar t)`. This value had to be provided for Forms compatibility, but you can use it for any purpose you want. Try to keep the value less than 100 to not interfere with reserved values.

FLTK does not use RTTI (Run Time Typing Information) to enhance portability. But this may change in the near future if RTTI becomes standard everywhere.

If you don't have RTTI you can use the clumsy FLTK mechanism, by having `type()` use a unique value. These unique values must be greater than the symbol `FL_RESERVED_TYPE` (which is 100) and less than `FL_WINDOW` (unless you make a subclass of `Fl_Window`). Look through the header files for `FL_RESERVED_TYPE` to find an unused number. If you make a subclass of `Fl_Window` you must use `FL_WINDOW + n` (where `n` must be in the range 1 to 7).
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:

```cpp
int MyClass::handle(int event) {
    switch(event) {
    case FL_PUSH:
        highlight = 1;
        redraw();
        return 1;
    case FL_DRAG: {
        int t = Fl::event_inside(this);
        if (t != highlight) {
            highlight = t;
            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.:

```cpp
#include <FL/names.h>  // defines fl_eventnames[]
[...]
int MyClass::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e);  // e.g. "Event was FL_PUSH (1)"
    [...]
}
```

Generated by Doxygen
Adding and Extending Widgets

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:

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

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

MyClass::MyClass(int x, int y, int w, int h) :
    Fl_Group(x, y, w, h),
    the_button(x + 5, y + 5, 100, 20),
    the_slider(x, y + 50, w, 20)
{...
    (you could add dynamically created child widgets here)...
    end(); // don’t forget to do this!
}

The child widgets need callbacks. These will be called with a pointer to the children, but the widget itself may be found in the parent() pointer of the child. Usually these callbacks can be static private methods, with a matching private method:

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

```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.

void Fl_Group::draw_children()

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

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

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

void Fl_Group::update_child(Fl_Widget & widget) const

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

Generated by Doxygen
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) : Fl_Gl_Window(X, Y, W, H, L) {} }
```

The draw() and handle() methods are described below. Like any widget, you can include additional private and public data in your class (such as scene graph information, etc.)

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_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;
    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;
    default:
        // pass other events to the base class...
        return Fl_Gl_Window::handle(event);
    }
}
```
When `handle()` is called, the OpenGL context is not set up! If your display changes, you should call `redraw()` and let `draw()` do the work. Don't call any OpenGL drawing functions from inside `handle()`!

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

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

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

### 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:
```
  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 boxtype or other places with some care.

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

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

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

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

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

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

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

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

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

### 12.5 Using FLTK widgets in OpenGL Windows

FLTK widgets can be added to Fl_Gl_Windows just as they would be added to Fl_Windows. 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();
...};
...
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myGlWindow->end();
```
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.

12.6 OpenGL Drawing Functions

FLTK provides some useful OpenGL drawing functions. They can be freely mixed with any OpenGL calls, and are defined by including <FL/gl.h> which you should include instead of the OpenGL header <GL/gl.h>.

**void gl_color(Fl_Color)**

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

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

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

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

**void gl_font(Fl_Font fontid, int size)**

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

**int gl_height()**

**int gl_descent()**

**float gl_width(const char *s)**

**float gl_width(const char *s, int n)**

**float gl_width(uchar c)**
Returns information about the current OpenGL font.

```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 \( \wedge X \), 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:

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

    void draw();

public:
    OptimizerWindow(int X, int Y, int W, int H, const char *L)
        : Fl_Gl_Window(X, Y, W, H, L) {
        context_ = (csContext *)0;
        draw_action_ = (csDrawAction *)0;
        scene_ = (csGroup *)0;
        camera_ = (csCamera *)0;
    }
    void scene(csGroup *g) { scene_ = g; redraw(); }
    void camera(csCamera *c) {
        camera_ = c;
        if (context_) {
            draw_action_->setCamera(camera_);
            camera_ ->draw(draw_action_);
            redraw();
        }
    }
};
```

The camera() Method

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

The draw() Method

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

```cpp
void OptimizerWindow::draw() {
    if (!context_) {
        // This is the first time we've been asked to draw; create the
        // Optimizer context for the scene...
        ifndef __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/;

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

Source-level changes for OpenGL 3:

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

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

• Put this in the handle(int event) member function of the first to be created among your Fl_Gl←_Window-derived classes:
  #ifndef __APPLE__
  static int first = 1;
  if (first && event == FL SHOW && shown()) {
    first = 0;
    make_current();
    glewInit(); // defines pters to functions of OpenGL V 1.2 and above
  }
  #endif

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

after the first glutCreateWindow() call.

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

  #ifdef __APPLE__
  glewExperimental = GL_TRUE;
  #endif

before the glewInit() call.

Testing for success of the glewInit() call

Testing whether the glewInit() call is successful is to be done as follows:

  #include <FL/platform.H> // defines FLTK_USE_WAYLAND under the Wayland platform
  #include <FL/Fl.H> // for Fl::warning()
  #ifdef __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
  // glewInit returns GLEW_ERROR_NO_GLX_DISPLAY with 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); 
  #endif // ! __APPLE__

Changes in the build process

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

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. The .fl file format is explained in detail in fluid/README_fl.txt

FLUID can "compile" the .fl file into a .cxx and a .h file. The .cxx file defines all the objects from the .fl file and the .h file declares all the global ones. FLUID also supports localization (Internationalization) of label strings and tooltips using message files and the GNU gettext or POSIX catgets interfaces.

A simple program can be made by putting all your code (including a `main()` function) into the .fl file and thus making the .cxx file a single source file to compile. Most programs are more complex than this, so you write other .cxx files that call the FLUID functions. These .cxx files must `#include` the .h file or they can `#include` the .cxx file so it still appears to be a single source file.

Generated by Doxygen
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:nn
-geometry WxX+X+Y
-title windowtitle
-name classname
-icoic
-fg color
-bg color
-bg2 color
-scheme schemename
```

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

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

### 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:

```bash
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; }
        /* Set the rotation about the horizontal (x) axis.
         * This function is called by the vertical roller in
         * CubeViewUI and the initialize button in CubeViewUI.
         */
        void h_angle(double angle) { hAng = angle; }
        // The rotation about the horizontal (x) axis.
        double h_angle()const { return hAng; }
        /* Sets the x shift of the cube view camera.
         * This function is called by the slider in CubeViewUI
         * and the initialize button in CubeViewUI.
         */
        void panx(double x) { xshift = x; }
        /* Sets the y shift of the cube view camera.
         * This function is called by the slider in CubeViewUI
         * and the initialize button in CubeViewUI.
         */
        void pany(double y) { yshift = y; }
        /* The widget class draw() override.
         * The draw() function initializes Gl for another round of
         * drawing, then calls specialized functions for drawing each
         * of the entities displayed in the cube view.
         */
        void draw();
    private:
        /* Draw the cube boundaries.
        * Draw the faces of the cube using the boxv[] vertices,
        * using GL_LINE_LOOP for the faces.
        */
        void drawCube();
        double vAng, hAng;
        double xshift, yshift;
        float boxv0[3]; float boxv1[3];
        float boxv2[3]; float boxv3[3];
        float boxv4[3]; float boxv5[3];
        float boxv6[3]; float boxv7[3];
};
```

The CubeView Class Implementation

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

```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;
```
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
  glShadeModel(GL_FLAT);
  glBegin(GL_QUADS);
  glColor4f(0.0, 0.0, 1.0, ALPHA);
  glVertex3fv(boxv0);
  glVertex3fv(boxv1);
  glVertex3fv(boxv2);
  glVertex3fv(boxv3);
  glColor4f(1.0, 1.0, 0.0, ALPHA);
  glVertex3fv(boxv0);
  glVertex3fv(boxv4);
  glVertex3fv(boxv5);
  glVertex3fv(boxv1);
  glColor4f(0.0, 1.0, 1.0, ALPHA);
  glVertex3fv(boxv2);
  glVertex3fv(boxv6);
  glVertex3fv(boxv7);
  glVertex3fv(boxv3);
  glColor4f(1.0, 0.0, 0.0, ALPHA);
  glVertex3fv(boxv4);
  glVertex3fv(boxv5);
  glVertex3fv(boxv6);
  glVertex3fv(boxv7);
  glColor4f(1.0, 0.0, 1.0, ALPHA);
  glVertex3fv(boxv0);
  glVertex3fv(boxv3);
  glVertex3fv(boxv7);
  glVertex3fv(boxv4);
  glColor4f(0.0, 1.0, 0.0, ALPHA);
  glVertex3fv(boxv1);
  glVertex3fv(boxv5);
  glVertex3fv(boxv6);
  glVertex3fv(boxv2);
  glEnd();
  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, 10000);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef((GLfloat)xshift, (GLfloat)yshift, 0);
  }
  drawCube();
  glEnd();
}

// drawCube
void CubeView::draw() {
  if (!valid()) {
    glLoadIdentity();
    glViewport(0, 0, pixel_w(), pixel_h());
    glOrtho(-10, 10, -10, 10, -20050, 10000);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef((GLfloat)xshift, (GLfloat)yshift, 0);
  }
  drawCube();
  glEnd();
}
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.

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 \texttt{vrot}
• A horizontal slider named \texttt{xpan}
• A vertical slider named \texttt{ypan}
• A horizontal value slider named \texttt{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 \texttt{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 \texttt{Fl_Gl_Window} class, which is created in the same way as an \texttt{Fl_Box} widget. Use \texttt{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 \texttt{Class:} text entry box. This tells FLUID that it is not an \texttt{Fl_Box}, but a similar widget with the same constructor.

In the \texttt{Extra Code:} field enter \texttt{#include "CubeView.h"}

This \texttt{#include} is important, as we have just included CubeView as a member of CubeViewUI, so any public CubeView methods are now available to CubeViewUI.
Defining the Callbacks

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

```cpp
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 FLUID Reference

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/Save A Copy...

Save a copy of the .fl data to a different file.

File/Revert...

Revert the .fl data to a previously saved state.

File/New From Template...

Create a new user interface design from a previously saved template. This can be useful for including a predefined enterprise copyright message for projects, or for managing boilerplate code for repeating project code.

File/Save As Template...
Save the current project as a starting point for future projects.

File/Print... (Ctrl-P)

Generate a printout containing all currently open windows within your project.

File/Write Code (Ctrl+Shift+C)

"Compiles" the data into a .cxx and .h file. These are exactly the same as the files you get when you run FLUID with the \(-c\) switch.

The output file names are the same as the .fl file, with the leading directory and trailing ".fl" stripped, and ".h" or ".cxx" appended.

File/Write Strings (Ctrl+Shift+W)

Writes a message file for all of the text labels defined in the current file.

The output file name is the same as the .fl file, with the leading directory and trailing ".fl" stripped, and ".txt", ".po", or ".msg" appended depending on the Internationalization Mode.

File/Quit (Ctrl+q)

Exits FLUID. You are asked for confirmation if you have changed the current file.

Edit/Undo (Ctrl+z) and Redo (Shift+Ctrl+z)
FLUID saves the project state for undo and redo operations after every major change.

**Edit/Cut (Ctrl+x)**

Deletes the selected widgets and all of their children. These are saved to a "clipboard" file and can be pasted back into any FLUID window.

**Edit/Copy (Ctrl+c)**

Copies the selected widgets and all of their children to the "clipboard" file.

**Edit/Paste (Ctrl+v)**

Pastes the widgets from the clipboard file.

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

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

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

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

**Edit/Select All (Ctrl+a)**

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

Edit/Properties... (F1 or double click)

Displays the current widget in the attributes panel. If the widget is a window and it is not visible then the window is shown instead.

Edit/Sort

Sorts the selected widgets into left to right, top to bottom order. You need to do this to make navigation keys in FLTK work correctly. You may then fine-tune the sorting with "Earlier" and "Later". This does not affect the positions of windows or functions.

Edit/Earlier (F2)

Moves all of the selected widgets one earlier in order among the children of their parent (if possible). This will affect navigation order, and if the widgets overlap it will affect how they draw, as the later widget is drawn on top of the earlier one. You can also use this to reorder functions, classes, and windows within functions.

Edit/Later (F3)

Moves all of the selected widgets one later in order among the children of their parent (if possible).

Edit/Group (F7)

Creates a new FL_Group and make all the currently selected widgets children of it.

Edit/Ungroup (F8)
Deletes the parent group if all the children of a group are selected.

*Edit/Show or Hide Overlays (Ctrl+Shift+O)*

Toggles the display of the red overlays off, without changing the selection. This makes it easier to see box borders and how the layout looks. The overlays will be forced back on if you change the selection.

*Edit/Show or Hide Guides (Ctrl+Shift+G)*

Guides can be used to arrange a widget layout easily and consistently. They indicating preferred widget positions and sizes with user definable margins, grids, and gap sizes. See the "Layout" tab in the "Settings" dialog.

This menu item enables and disable guides and the snapping action when dragging widgets and their borders.

*Edit/Show or Hide Restricted (Ctrl+Shift+R)*

It is recommended to ensure that user interface elements within FLTK do not exceed the size of their container group or overlap with each other. By activating this button, a hatch pattern is enabled, visually highlighting areas where restricted or undefined behavior may occur.

*Edit/Show or Hide Widget Bin (Alt+B)*

The widget bin provides quick access to all widget types supported by FLUID. Layouts can be created by clicking on elements in the widget bin, or by dragging them from the bin to their position within the layout. This button shows or hides the widget bin.

*Edit/Show or Hide Source Code (Shift+Alt+S)*
This button shows or hides the source code preview window. Any change to the layout or code in the layout editors can be previewed and verified immediately in the Code View window.

Edit/Settings... (Alt+p)

Displays the FLUID settings panel. The settings panel contains tabs for general FLUID settings, project settings, layout preferences, custom shell commands, and the internationalisation settings for GNU Gettext and POSIX catgets.

Under the "Project" tab you control the extensions or names of the files that are generated by FLUID. If you check the "Include Header from Code" button the code file will include the header file automatically.

Under the "Locale" tab are the internationalization options, described later in this chapter.

![FLUID General Settings Tab](image)

Figure 13.7 FLUID General Settings Tab
New/Code/Function

Creates a new C function. You will be asked for a name for the function. This name should be a legal C++ function template, without the return type. You can pass arguments which can be referred to by code you type into the individual widgets.

If the function contains any unnamed windows, it will be declared as returning an Fl_Window pointer. The unnamed window will be returned from it (more than one unnamed window is useless). If the function contains only named windows, it will be declared as returning nothing (void).

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

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

New/Window

Figure 13.8 FLUID Project Settings Tab
Creates a new Fl_Window widget. The window is added to the currently selected function, or to the function containing the currently selected item. The window will appear, sized to 100x100. You can resize it to whatever size you require.

The widget panel will also appear and is described later in this chapter.

New/...

All other items on the New menu are subclasses of Fl_Widget. Creating them will add them to the currently selected group or window, or the group or window containing the currently selected widget. The initial dimensions and position are chosen by copying the current widget, if possible.

When you create the widget you will get the widget's control panel, which is described later in this chapter.

Layout/Align/...

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

Layout/Space Evenly/...

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

Layout/Makce Same Size/...

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

Layout/Center in Group/...
Center all selected widgets relative to their parent widget

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

Displays the grid settings panel.

This panel controls dimensions that all widgets snap to when you move and resize them, and for the "snap" which is how far a widget has to be dragged from its original position to actually change.

Layout preferences are defined using margins to parent groups and window, gaps between widget, and /orr by overlaying a grid over a group or window. A layout comes as a suite of three presets, one for the main application window, one for dialog boxes, and one for toolboxes.

FLUID comes with two include layout suites. FLTK was used to design FLUID and other included apps, and Grid is a more ridgid grid layout. Users can add more suites, import and export them, and include them into their .fl project files.
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.
Run the shell command again.

Help/Repid development with FLUID

Shows this chapter of the manual.

Help/FLTK Programmers Manual

Shows the contents page of the manual

Help/About FLUID

Pops up a panel showing the version of FLUID.

### 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 "Hide Overlays" button (or type Ctrl+Shift+O) to hide the red overlay so you can see the widgets more accurately, especially when setting the box type.

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

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

Figure 13.10 The FLUID widget GUI attributes
Image

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

Inactive

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

Alignment (buttons)

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

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

Position (text fields)

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

These fields understand basic math and variables.Appending +10 to the X coordinate will move a widget 10 units to the right without having to reenter the value. Entering the formula \( w + 3 \) in the **Width** field will widen all selected Widgets by 3 units. The formula \( py + i * 20 \) in the Y field will order all selected widgets vertically in their group by increments of 20 units.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>zero based counter of selected widgets</td>
</tr>
<tr>
<td>x, y, w, h</td>
<td>position and size of the current widget</td>
</tr>
<tr>
<td>px, py, pw, ph</td>
<td>dimensions of the parent widget</td>
</tr>
<tr>
<td>sx, sy, sw, sh</td>
<td>dimensions of the previous sibling</td>
</tr>
<tr>
<td>cx, cy, cw, ch</td>
<td>bounding box of all children</td>
</tr>
</tbody>
</table>
Values (text fields)

The values and limits of the current widget. Depending on the type of widget, some or all of these fields may be inactive.

Shortcut

The shortcut key to activate the widget. Click on the shortcut button and press any key sequence to set the shortcut.

Attributes (buttons)

The **Visible** button controls whether the widget is visible (on) or hidden (off) initially. Don't change this for windows or for the immediate children of a Tabs group.

The **Active** button controls whether the widget is activated (on) or deactivated (off) initially. Most widgets appear greyed out when deactivated.

The **Resizable** button controls whether the window is resizeable. In addition all the size changes of a window or group will go "into" the resizable child. If you have a large data display surrounded by buttons, you probably want that data area to be 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.
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.

![FLUID widget Style attributes](image)

**Figure 13.11 The FLUID widget Style attributes**

### 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.

![FL_Group Properties](image)

**Figure 13.12 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.
13.7 GUI Attributes

Type (upper-right pulldown menu)

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

Name (text field)

Name of a variable to declare, and to store a pointer to this widget into. This variable will be of type "<class> ∗". If the name is blank then no variable is created.

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

Public (button)

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

Extra Code (text fields)

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

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

All other lines are "code" lines. The current widget is pointed to by the local variable o. The window being constructed is pointed to by the local variable w. You can also access any arguments passed to the function here, and any named widgets that are before this one.
FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler. If you need more than four lines you probably should call a function in your own .cxx code.

**Callback (text field)**

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

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

A code snippet is inserted into a static function in the .cxx output file. The function prototype is `void name(class *o, void *v)` so that you can refer to the widget as `o` and the `user_data()` as `v`. FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler.

If the callback is blank then no callback is set.

**User Data (text field)**

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

**Type (text field)**

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

**When (pulldown menu)**

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

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

**No Change (button)**

The *No Change* button means the callback is done on the matching event even if the data is not changed.
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: none, GNU gettext, and POSIX catgets. The "none" method is the default and just passes the label strings as-is to the widget constructors.

The "GNU gettext" method uses GNU gettext (or a similar text-based I18N library) to retrieve a localized string before calling the widget constructor.

The "POSIX catgets" method uses the POSIX catgets function to retrieve a numbered message from a message catalog before calling the widget constructor.
13.11 Internationalization with FLUID

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.

![FLUID Settings](image)

**Figure 13.13 Internationalization using GNU gettext**

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

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

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

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

See also

GNU gettext special cases

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.

![FLUID Settings](image)

**Figure 13.14 Internationalization using POSIX catgets**

The **#include** field controls the header file to include for I18N; by default this is `<nl_types.h>`, the standard I18N file for POSIX catgets.

The **File**: field controls the name of the catalog file variable to use when retrieving localized messages; by default the file field is empty which forces a local (static) catalog file to be used for all of the windows defined in your `.fl` file.

The **Set**: field controls the set number in the catalog file. The default set is 1 and rarely needs to be changed.

### 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>
</tbody>
</table>
### Key Combo

<table>
<thead>
<tr>
<th>Key Combo</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<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>

### Key Combo in interactive Window

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

### 13.14 Licenses

FLUID uses graphical images based on the Zendesk Garden Stroke icon set: https://github.com/zendeskgarden. Garden Stroke is licensed under the Apache License, Version 2.0: https://
://www.apache.org/licenses/LICENSE-2.0.html.
Chapter 14

FLTK Runtime Options

In this chapter, we will cover how to access and alter settings for applications created using FLTK, both as an administrator and as a regular user.

Subchapters:

• Runtime Options
• Obtaining Current Settings
• Administrative Tool
• List of Options

14.1 Runtime Options

FLTK keeps track of various aspects of the user interface in a system-wide database. Users have the ability to set their own preferences and override default or system settings. For instance, FLTK will display a dotted rectangle around the widget with current focus. This might not be desirable for users who do not use keyboard navigation and do not need the rectangle. This can be turned off by setting the OPTION_VISIBLE_FOCUS option to ‘off’ for that user, which will disable the focus rectangle in all FLTK-based applications.

14.2 Obtaining Current Settings

Options are kept in preference files using the signature Fl_Preferences::CORE_SYSTEM, "fltk.org", "fltk" for system-wide settings and Fl_Preferences::CORE_USER, "fltk.org", "fltk" for individual users. They can be accessed by using the function bool Fl::option(Fl_Option opt). If an application needs to temporarily override user or system settings, it can use the function void option(Fl_← Option opt, bool val).

To make changes to options permanently, FLTK provides an administrative tool called fltk-options.
14.3 Administrative Tool

fltk-options is a hybrid app that is part of FLTK and can be installed on the target system. It includes an up-to-date man page.

![fltk-options Application](image)

Figure 14.1 fltk-options Application

When fltk-options is called without any command-line arguments, it opens in interactive mode and provides a user interface to view and alter all system and current user options.

Starting the tool from a shell, the command-line options -S and -U can be used to display or change system or user options. On MS-Windows, fltk-options is also available as fltk-options-cmd.exe.

14.4 List of Options

Calling fltk-options --help gives a list of all available commands, and options and their values. fltk-options --help OPTION prints more detailed information for OPTION if available. In interactive mode, tooltips provide this additional information.

A full list of options can be found in the manual at Fl::Fl_Option.
Chapter 15

Advanced FLTK

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

15.1 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.

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

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

FLTK supports multithreaded applications using a locking mechanism internally. This allows a worker thread to lock the rendering context, preventing any drawing from taking place, whilst it changes the value of its widget.
The converse is also true; whilst a worker thread holds the lock, the main() thread may not be able to process any drawing requests, nor service any events. So a worker thread that holds the FLTK lock must contrive to do so for the shortest time possible or it could impair operation of the application.

The lock operates broadly as follows.

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

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

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

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

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

15.3 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) {   /* do thread work */
  while (thread_still_running) {         /* 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
  }
}
```
15.3 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:

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

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

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

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

Using Fl::awake callback messages

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

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

```c
void do_something_cb(void *userdata) {
    // Will run in the context of the main thread
    ... do_stuff ... 
}
```

// running in worker thread
```c
void *data; // "data" is a pointer to your user data
Fl::awake(do_something_cb, data); // call to execute cb in main thread
```

Note

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

15.4 FLTK multithreaded "lockless programming"

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

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

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

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

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

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

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

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

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

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

Using Fl::awake is not, strictly speaking, entirely "lockless" since the awake handler mechanism incorporates resource locking internally to protect the queue of pending awake messages. These resource locks are held transiently and generally do not trigger the pathological blocking issues described here.

However, aside from using Fl::awake, there are many other ways that a "lockless" design can be implemented, including message passing, various forms of IPC, etc.

If you need high performing multithreaded programming, then take some time to study the options and understand the advantages and disadvantages of each; we can't even begin to scratch the surface of this huge topic here!

And of course occasional, sparse, use of the FLTK lock from worker threads will do no harm; it is "excessive" locking (whatever that might be) that triggers the failing behaviour.

It is always a Good Idea to update the GUI at the lowest rate that is acceptable when processing bulk data (or indeed, in all cases!) Updating at a few frames per second is probably adequate for providing feedback during a long calculation. At the upper limit, anything faster than the frame rate of your monitor and the updates will never even be displayed; why waste CPU computing pixels that you will never show?

15.5 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.)

• Starting with FLTK 1.4, it's possible to start (or cancel) a timer from a worker thread under the condition that the call to Fl::add_timeout (or Fl::remove_timeout) is wrapped in Fl::lock() and Fl::unlock().

• 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 16

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.

16.1 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+00FFFF. UCS-4 uses four bytes, which lets it handle higher characters, but this is even more wasteful for ASCII or Latin1.

**UTF-8**

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

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

All UCS characters above U+007F are encoded as a sequence of several bytes. The top bits of the first byte are set to show the length of the byte sequence, and subsequent bytes are always in the range 0x80 to 0xBF. This combination provides some level of synchronisation and error detection.
16.2 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().

Generated by Doxygen
16.3 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.

16.4 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.

```c
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
```
fl_utf8decode() attempts to decode the UTF-8 character that starts at \texttt{p} and may not extend past \texttt{end}. It returns the Unicode value, and the length of the UTF-8 character sequence is returned via the \texttt{len} argument. fl_utf8encode() writes the UTF-8 encoding of \texttt{ucs} into \texttt{buf} and returns the number of bytes in the sequence. See the main documentation for the treatment of illegal Unicode and UTF-8 sequences.

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

fl_utf8froma() converts a character string containing single bytes per character (i.e. ASCII or ISO-8859-1) into UTF-8. If the \texttt{src} string contains only ASCII characters, the return value will be the same as \texttt{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. \texttt{dstlen} provides a means of limiting the number of bytes written, so setting \texttt{dstlen} to zero is a means of measuring how much storage would be needed before doing the real conversion.

cchar* fl_utf2mbcs(const char *src) OksiD

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

unsigned int fl_utf8fromwc(char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen) FLTK2
unsigned int fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen) FLTK2
unsigned int fl_utf8tolUtf16(const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen) FLTK2

These routines convert between UTF-8 and \texttt{wchar_t} or "wide character" strings. The difficulty lies in the fact that \texttt{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 \texttt{wchar_t}.

fl_utf8fromwc() converts from a "wide character" string to UTF-8. Note that \texttt{srclen} is the number of \texttt{wchar_t} elements in the source string and on Windows this might be larger than the number of characters. \texttt{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. \texttt{dstlen} specifies the maximum number of \texttt{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?]
16.5 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 17

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.

17.1 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).
17.2 Events

Events are identified by an \texttt{FL\_Event} enumeration value. The following events are currently defined:

- \texttt{FL\_NO\_EVENT} - No event (or an event fltk does not understand) occurred (0).
- \texttt{FL\_PUSH} - A mouse button was pushed.
- \texttt{FL\_RELEASE} - A mouse button was released.
- \texttt{FL\_ENTER} - The mouse pointer entered a widget.
- \texttt{FL\_LEAVE} - The mouse pointer left a widget.
- \texttt{FL\_DRAG} - The mouse pointer was moved with a button pressed.
- \texttt{FL\_FOCUS} - A widget should receive keyboard focus.
- \texttt{FL\_UNFOCUS} - A widget loses keyboard focus.
- \texttt{FL\_KEYBOARD} - A key was pressed.
- \texttt{FL\_CLOSE} - A window was closed.
- \texttt{FL\_MOVE} - The mouse pointer was moved with no buttons pressed.
- \texttt{FL\_SHORTCUT} - The user pressed a shortcut key.
- \texttt{FL\_DEACTIVATE} - The widget has been deactivated.
- \texttt{FL\_ACTIVATE} - The widget has been activated.
- \texttt{FL\_HIDE} - The widget has been hidden.
- \texttt{FL\_SHOW} - The widget has been shown.
- \texttt{FL\_PASTE} - The widget should paste the contents of the clipboard.
- \texttt{FL\_SELECTIONCLEAR} - The widget should clear any selections made for the clipboard.
- \texttt{FL\_MOUSEWHEEL} - The horizontal or vertical mousewheel was turned.
- \texttt{FL\_DND\_ENTER} - The mouse pointer entered a widget dragging data.
- \texttt{FL\_DND\_DRAG} - The mouse pointer was moved dragging data.
- \texttt{FL\_DND\_LEAVE} - The mouse pointer left a widget still dragging data.
- \texttt{FL\_DND\_RELEASE} - Dragged data is about to be dropped.
- \texttt{FL\_SCREEN\_CONFIGURATION\_CHANGED} - The screen configuration (number, positions) was changed.
- \texttt{FL\_FULLSCREEN} - The fullscreen state of the window has changed.
17.3 Callback "When" Conditions

The following constants determine when a callback is performed:

- FL_WHEN_NEVER - Never call the callback (0).
- FL_WHEN_CHANGED - Do the callback only when the widget value changes.
- FL_WHEN_NOT_CHANGED - Do the callback whenever the user interacts with the widget.
- FL_WHEN_RELEASE - Do the callback when the button or key is released and the value changes.
- FL_WHEN_ENTER_KEY - Do the callback when the user presses the ENTER key and the value changes.
- FL_WHEN_RELEASE_ALWAYS - Do the callback when the button or key is released, even if the value doesn't change.
- FL_WHEN_ENTER_KEY_ALWAYS - Do the callback when the user presses the ENTER key, even if the value doesn't change.

17.4 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

17.5 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.

### 17.6 Fl::event_state() Values

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

- FL_SHIFT - One of the shift keys is down.
- FL_CAPS_LOCK - The caps lock is on.
- FL_CTRL - One of the ctrl keys is down.
- FL_ALT - One of the alt keys is down.
- FL_NUM_LOCK - The num lock is on.
- FL_META - One of the meta/Windows keys is down.
- FL_COMMAND - An alias for FL_CTRL on Windows, X11 and Wayland, or FL_META on MacOS X.
- FL_CONTROL - An alias for FL_META on Windows, X11 and Wayland, or FL_CTRL on MacOS X.
- FL_SCROLL_LOCK - The scroll lock is on.
- FL_BUTTON1 - Mouse button 1 is pushed.
- FL_BUTTON2 - Mouse button 2 is pushed.
- FL_BUTTON3 - Mouse button 3 is pushed.
- FL_BUTTONS - Any mouse button is pushed.
- FL_BUTTON(n) - Mouse button n (where n > 0) is pushed.
17.7 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.

17.8 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.

17.9 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.

17.9.1 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
17.10 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.

17.10 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
17.11 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.

17.12 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 18

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.

18.1 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.

18.2 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)`

Generated by Doxygen
• 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").

### 18.3 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.
18.4 class Fl_Glut_Window

18.4.1 Class Hierarchy

Fl_Gl_Window
   +----Fl_Glut_Window

18.4.2 Include Files

#include <FL/glut.H>

18.4.3 Description

Each GLUT window is an instance of this class. You may find it useful to manipulate instances directly rather than use GLUT window id's. These may be created without opening the display, and thus can fit better into FLTK's method of creating windows.

The current GLUT window is available in the global variable glut_window.

new Fl_Glut_Window(...) is the same as glutCreateWindow() except it does not show() the window or make the window current.

window->make_current() is the same as glutSetWindow(number). If the window has not had show() called on it yet, some functions that assume an OpenGL context will not work. If you do show() the window, call make_current() again to set the context.

~Fl_Glut_Window() is the same as glutDestroyWindow().

18.4.4 Members

The Fl_Glut_Window class contains several public members that can be altered directly:

<table>
<thead>
<tr>
<th>member</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>A pointer to the function to call to draw the normal planes.</td>
</tr>
<tr>
<td>entry</td>
<td>A pointer to the function to call when the mouse moves into or out of the window.</td>
</tr>
<tr>
<td>keyboard</td>
<td>A pointer to the function to call when a regular key is pressed.</td>
</tr>
<tr>
<td>menu[3]</td>
<td>The menu to post when one of the mouse buttons is pressed.</td>
</tr>
<tr>
<td>mouse</td>
<td>A pointer to the function to call when a button is pressed or released.</td>
</tr>
<tr>
<td>motion</td>
<td>A pointer to the function to call when the mouse is moved with a button down.</td>
</tr>
<tr>
<td>overlaydisplay</td>
<td>A pointer to the function to call to draw the overlay planes.</td>
</tr>
<tr>
<td>passivemotion</td>
<td>A pointer to the function to call when the mouse is moved with no buttons down.</td>
</tr>
<tr>
<td>reshape</td>
<td>A pointer to the function to call when the window is resized.</td>
</tr>
<tr>
<td>special</td>
<td>A pointer to the function to call when a special key is pressed.</td>
</tr>
<tr>
<td>visibility</td>
<td>A pointer to the function to call when the window is iconified or restored (made visible.)</td>
</tr>
</tbody>
</table>
18.4.5 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.
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**

### 19.1 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>`.

### 19.2 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.
19.3 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()
19.4 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>LEFALTKEY,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 20

Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

- Accessing the OS Interfaces
- The Wayland/X11 hybrid library
- The UNIX (X11) Interface
- The Windows Interface
- The Apple OS X Interface
- The Wayland Interface

20.1 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.
20.2 The Wayland/X11 hybrid library

By default, the FLTK library is, under Linux and Unix, a Wayland/X11 hybrid which can run FLTK-based apps as Wayland clients or as X11 clients. The choice between running an app as a Wayland or an X11 client is done as follows, when the app runs function `fl_open_display()` (that function can be called explicitly by the app or implicitly by FLTK, for example the first time an `Fl_Window` is `show()`):

- if environment variable `FLTK_BACKEND` is not defined, Wayland is used if a Wayland compositor is available, otherwise X11 is used;
- if `$FLTK_BACKEND` equals "wayland", the library makes the app a Wayland client, and stops with error if no Wayland compositor is available;
- if `$FLTK_BACKEND` equals "x11", the library makes the app an X11 client even if a Wayland compositor is available.

After function `fl_open_display()` has been called, exactly one of the functions `fl_wl_display()` and `fl_x11_display()` returns a non-NULL value. When the former function does, the app runs as a Wayland client, and Wayland-specific functions and symbols described below (The Wayland Interface) can be used, whereas X11-specific functions and symbols cannot. Otherwise, the app runs as an X11 client, and only X11-specific functions and symbols below (The UNIX (X11) Interface) can be used.

Because a single app can be expected to run either Wayland or X11, it's necessary to use distinct names for global variables and functions in the X11- and the Wayland-specific source code.

Non-default configurations of the FLTK library under Linux/Unix are described in file README.Wayland.txt.

20.3 The UNIX (X11) Interface

Cross-platform applications should bracket X11-specific source code between `#if defined(FLTK_USE_X11) / #endif` and should ensure function `fl_x11_display()` returns non-NULL before calling X11-specific functions and using X11-specific symbols.

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

20.3.1 Handling Other X Events

```c
void Fl::add_handler(int (*f)(int))
```

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

FLTK calls this for any X events it does not recognize, or X events with a window ID that FLTK does not recognize. You can look at the X event in the `fl_xevent` variable.
The argument is the FLTK event type that was not handled, or zero for unrecognized X events. These handlers are also called for global shortcuts and some other events that the widget they were passed to did not handle, for example FL_SHORTCUT.

```c
extern XEvent *fl_xevent
```

This variable contains the most recent X event.

```c
extern ulong fl_event_time
```

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

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

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

**Deprecated** Kept for compatibility with FLTK versions before 1.4. Use preferentially `fl_x11_xid(const Fl_Window *)` with versions 1.4 and above.

```c
Fl_Window *fl_find(ulong xid)
```

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

**Deprecated** Kept for compatibility with FLTK versions before 1.4. Use preferentially `fl_x11_find(Window)` with versions 1.4 and above.

```c
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.
20.3.2 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*.

20.3.3 Changing the Display, Screen, or X Visual

FLTK uses only a single display, screen, X visual, and X colormap. This greatly simplifies its internal structure and makes it much smaller and faster. You can change which it uses by setting global variables before the first `Fl_Window::show()` is called. You may also want to call `Fl::visual()`, which is a portable interface to get a full color and/or double buffered visual.

```
int Fl::display(const char *)
```
Set which X display to use. This actually does `putenv("DISPLAY=...")` so that child programs will display on the same screen if called with `exec()`. This must be done before the display is opened. This call is provided under MacOS and Windows but it has no effect.

```c
extern Display *fl_display
```

The open X display. This is needed as an argument to most Xlib calls. Don't attempt to change it! This is `NULL` before the display is opened.

```c
void fl_open_display()
```

Opens the display. Does nothing if it is already open. This will make sure `fl_display` is non-zero. You should call this if you wish to do X calls and there is a chance that your code will be called before the first `show()` of a window.

This may call `Fl::abort()` if there is an error opening the display.

```c
void fl_close_display()
```

This closes the X connection. You do not need to call this to exit, and in fact it is faster to not do so! It may be useful to call this if you want your program to continue without the X connection. You cannot open the display again, and probably cannot call any FLTK functions.

```c
extern int fl_screen
```

Which screen number to use. This is set by `fl_open_display()` to the default screen. You can change it by setting this to a different value immediately afterwards. It can also be set by changing the last number in the `Fl::display()` string to "host:0.#".

```c
extern XVisualInfo *fl_visual
extern Colormap fl_colormap
```

The visual and colormap that FLTK will use for all windows. These are set by `fl_open_display()` to the default visual and colormap. You can change them before calling `show()` on the first window. Typical code for changing the default visual is:

```c
Fl::args(argc, argv); // do this first so $DISPLAY is set
fl_open_display();
fl_visual = find_a_good_visual(fl_display, fl_screen);
if (!fl_visual) Fl::abort("No good visual");
fl_colormap = make_a_colormap(fl_display, fl_visual->visual, fl_visual->depth);
// it is now ok to show() windows:
window->show(argc, argv);
```
20.3.4 Using a Subclass of Fl_Window for Special X Stuff

FLTK can manage an X window on a different screen, visual and/or colormap, you just can’t use FLTK’s drawing routines to draw into it. But you can write your own `draw()` method that uses Xlib (and/or OpenGL) calls only.

FLTK can also manage XID’s provided by other libraries or programs, and call those libraries when the window needs to be redrawn.

To do this, you need to make a subclass of Fl_Window and override some of these virtual functions:

```c
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);
}
```

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::set_xid(Fl_Window*, Window xid)
```

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.

```c
virtual void Fl_Window::flush()
```

This virtual function is called by Fl::flush() to update the window. For FLTK’s own windows it does this by setting the global variables fl_window and fl_gc and then calling the `draw()` method. For your own windows you might just want to put all the drawing code in here.
The X region that is a combination of all damage() calls done so far is in Fl_X::flx(this)->region. If NULL then you should redraw the entire window. The undocumented function fl_clip_region(XRegion) will initialize the FLTK clip stack with a region or NULL for no clipping. You must set region to NULL afterwards as fl_clip_region() will own and delete it when done.

If damage() & FL_DAMAGE_EXPOSE then only X expose events have happened. This may be useful if you have an undamaged image (such as a backing buffer) around.

Here is a sample where an undamaged image is kept somewhere:

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

Note

For compatibility with FLTK versions before 1.4, member function Fl_X::flx(Fl_Window*) can also be written Fl_X::i(Fl_Window*).

virtual void Fl_Window::hide()

Destroy the window server copy of the window. Usually you will destroy contexts, pixmaps, or other resources used by the window, and then call Fl_Window::hide() to get rid of the main window identified by xid(). If you override this, you must also override the destructor as shown:

```cpp
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):

```cpp
MyWindow::~MyWindow() {
    hide();
}
```

Note

Access to the Fl_X hidden class requires to #define FL_INTERNALS before compilation.
20.3.5 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:

```c
#include "icon.xbm"
fl_open_display(); // needed if display has not been previously opened
Pixmap p = XCreateBitmapFromData(fl_display, DefaultRootWindow(fl_display),
                                  icon_bits, icon_width, icon_height);
window->icon((const void*)p);
```

To use a multi-colored icon, the XPM format and library should be used as follows:

```c
#include <X11/xpm.h>
#include "icon.xpm"
fl_open_display(); // needed if display has not been previously opened
Pixmap p, mask;
XpmCreatePixmapFromData(fl_display, DefaultRootWindow(fl_display),
                         icon_xpm, &p, &mask, NULL);
window->icon((const void *)p);
```

When using the Xpm library, be sure to include it in the list of libraries that are used to link the application (usually "-lXpm").

**NOTE:**
You must call `Fl_Window::show(int argc, char ** argv)` for the icon to be used. The `Fl_Window::show()` method does not bind the icon to the window.

Any window icon must be set with the above methods before the window is shown.

20.3.6 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*).
20.3.7 Display Scaling Factor

FLTK uses the value of the Xft.dpi resource divided by 96. to initialize the display scaling factor. That is also what is done by the gnome and KDE desktops.

20.4 The Windows Interface

Cross-platform applications should bracket Windows-specific source code between `#ifdef _WIN32 / #endif`.

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

20.4.1 Using filenames with non-ASCII characters

In FLTK, all strings, including filenames, are UTF-8 encoded. The utility functions fl_fopen() and fl_open() allow to open files potentially having non-ASCII names in a cross-platform fashion, whereas the standard fopen()/open() functions fail to do so.

20.4.2 Responding to WM_QUIT

FLTK will intercept WM_QUIT messages that are directed towards the thread that runs the main loop. These are converted to SIGTERM signals via `raise()`. This allows you to deal with outside termination requests with the same code on both Windows and UNIX systems. Other processes can send this message via `PostThreadMessage()` in order to request, rather than force your application to terminate.

20.4.3 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.

```c
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).

```c
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.

20.4.4 Drawing Things Using the Windows GDI

When the virtual function Fl_Widget::draw() is called, FLTK stores all the extra arguments you need to make a proper GDI call in some global variables:

```
extern HINSTANCE fl_display; // for compatibility with previous FLTK versions
extern HINSTANCE fl_win32_display(); // preferred access starting with FLTK 1.4
extern HWND fl_window;
extern HDC fl.gc; // for compatibility with previous FLTK versions
extern HDC fl_win32.gc(); // preferred access starting with FLTK 1.4
COLORREF fl_RGB();
HPEN fl_pen();
HBRUSH fl_brush();
```

These global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current(). You can refer to them when needed to produce GDI calls, but don't attempt to change them. The functions return GDI objects for the current color set by fl_color() and are created as needed and cached. A typical GDI drawing call is written like this:

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

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

20.4.5 HighDPI support

FLTK apps for the Windows platform are by default "Per-monitor DPI-aware V2". This means that any window automatically adjusts its physical size in relation to the scaling factor of the display where it maps. This also means that all drawings (e.g., text, lines, images) take advantage of the full resolution of the display in use. FLTK apps may also use the manifest mechanism to declare their level of DPI awareness. The FLTK library adapts to the DPI awareness level set in the app's manifest, which can be lower than the default level if the manifest sets it so.

20.4.6 Display Scaling Factor

FLTK uses the value given by function GetDpiForMonitor() divided by 96. to initialize the scaling factor of each display in the system. This matches the value of "Change the size of text, apps and other items" found in section "System" subsection "Display" of Windows settings.
20.4.7 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: `FL_Window::icon(const Fl_RGB_Image *)` and `FL_Window::icons(const Fl_RGB_Image *[ ], int)`. See also methods setting default window icons `FL_Window::default_icon(const Fl_RGB_Image *)` and `FL_Window::default_icons(const Fl_RGB_Image *[ ], int).

FLTK on Windows also supports, for backward compatibility, use of the deprecated method `FL_Window::icon(const void *)` as follows:
Set the icon for the window to the passed pointer. You will need to cast the HICON handle to a char* when calling this method. To set the icon using an icon resource compiled with your application use:

```c
window->icon((const void *)LoadIcon(fl_display, MAKEINTRESOURCE(IDI_ICON));
```

You can also use the `LoadImage()` and related functions to load specific resolutions or create the icon from bitmap data.

```
NOTE:
You must call `Fl_Window::show(int argc, char ** argv)` for the icon to be used. The `Fl_Window::show()` method does not bind the icon to the window.
```

Any window icon must be set with the above methods before the window is shown.

20.4.8 How to Not Get a MSDOS Console Window

Windows has a really stupid mode switch stored in the executables that controls whether or not to make a console window.

To always get a console window you simply create a console application (the "/SUBSYSTEM:CONSOLE" option for the linker). For a GUI-only application create a Windows application (the "/SUBSYSTEM:WINDOWS" option for the linker).

FLTK includes a `WinMain()` function that calls the ANSI standard `main()` entry point for you. This function creates a console window when you use the debug version of the library.

Windows applications without a console cannot write to `stdout` or `stderr`, even if they are run from a console window. Any output is silently thrown away. Additionally, Windows applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.

20.4.9 Known Windows Bugs and Problems

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

- If a program is deactivated, `Fl::wait()` does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in Windows and probably impossible to get around.

- `Fl_Gl_Window::can_do_overlay()` returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.

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

- Compilation with `gcc 3.4.4` and `-Os` exposes an optimisation bug in gcc. The symptom is that when drawing filled circles only the perimeter is drawn. This can for instance be seen in the symbols demo. Other optimisation options such as `-O2` and `-O3` seem to work OK. More details can be found in STR#1656
20.5 The Apple OS X Interface

Cross-platform applications should bracket macOS-specific source code between 
```c
#if defined(__APPLE__) && !defined(FLTK_USE_X11) / #endif
```

FLTK supports Apple OS X using the Apple Cocoa library. Older versions of MacOS are no longer supported.

Control, Option, and Command Modifier Keys

FLTK maps the Mac 'control' key to FL_CTRL, the 'option' key to FL_ALT and the 'Apple' key to FL_META. Furthermore, FL_COMMAND designates the 'Apple' key on Mac OS X and the 'control' key on other platforms. Keyboard events return the key name in Fl::event_key() and the keystroke translation in Fl::event_text(). For example, typing Option-Y on a Mac US keyboard will set FL_ALT in Fl::event_state(), set Fl::event_key() to 'y' and return the Yen symbol in Fl::event_text().

Right Click simulation with Ctrl Click

The Apple HIG guidelines indicate applications should support 'Ctrl Click' to simulate 'Right Click' for e.g. context menus, so users with one-button mice and one-click trackpads can still access right-click features. However, paraphrasing Manolo's comment on the fltk.coredev newsgroup:

- **FLTK does /not/ support Ctrl-Click == Right Click itself because Mac OS X event processing doesn’t support this at the system level: the system reports left-clicks with the ctrl modifier when the user ctrl-clicks, and OS X system preferences don’t allow changing this behavior. Therefore, applications must handle simulation of Right Click with Ctrl Click in the application code.**

Ian MacArthur provided the following handle() method code snippet showing an example of how to do this:

```c
case FL_PUSH:
{
  int btn = Fl::event_button();
  #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

Generated by Doxygen
When the user presses `Cmd-Q` or requests a termination of the application, FLTK sends an `FL_CLOSE` event to all open windows. If any window remains open, the termination request aborts. If all windows close, the application's event loop terminates, that is, `Fl::run()` returns. The application can then follow FLTK's normal termination path executing cleanup code that may be programmed after termination of the event loop, and returning from `main()`. Function `Fl::program_should_quit()` allows to detect whether the event loop terminated because of a program termination request.

Apple "Open" Event

Whenever the user drops a file onto an application icon, OS X generates an Apple Event of the type "Open". You can have FLTK notify you of an Open event by calling the `fl_open_callback()` function.

```c
void fl_open_display()
```

Opens the display. Does nothing if it is already open. You should call this if you wish to do Cocoa or Quartz calls and there is a chance that your code will be called before the first `show()` of a window.

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

Returns the window reference for an `Fl_Window`, or `NULL` if the window has not been shown. This reference is a pointer to an instance of the subclass FLWindow of Cocoa's NSWindow class.

```c
Fl_Window *fl_find(Window xid)
```

Returns the `Fl_Window` that corresponds to the given window reference, or `NULL` if not found.

```c
void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut)
```

Attaches the callback `cb` to the "About myprog" item of the system application menu. `cb` will be called with `NULL` first argument and `user_data` second argument. This MacOS-specific function is deprecated in FLTK 1.4 and replaced by `Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *data)` which is cross-platform.

**Fl_Sys_Menu_Bar class**

The `Fl_Sys_Menu_Bar` class allows to build menu bars that, on Mac OS X, are placed in the system menu bar (at top-left of display), and, on other platforms, at a user-chosen location of a user-chosen window.

### 20.5.1 Setting the icon of an application

- First, create a .icns file containing several copies of your icon of decreasing sizes. This can be done using the Preview application or the Icon Composer application available in "Graphics Tools for Xcode". To create a high resolution icon file, it is necessary to use the iconutil command-line utility.
- Put your .icns file in the Resources subdirectory of your application bundle.
- Add these two lines to the Info.plist file of your application bundle

```xml
<key>CFBundleIconFile</key>
<string>foo.icns</string>
```

replacing `foo` by your application name. If you use Xcode, just add your .icns file to your application target.
20.5.2 Drawing Things Using Quartz

All code inside `Fl_Widget::draw()` is expected to call Quartz drawing functions. The Quartz coordinate system is flipped to match FLTK's coordinate system. The origin for all drawing is in the top left corner of the enclosing `Fl_Window`. The function `fl_mac_gc()` returns the appropriate Quartz 2D drawing environment (of type `CGContextRef`). For compatibility with previous FLTK versions, deprecated global variable `fl_gc` gives the same value.

Include `FL/platform.H` to declare the `fl_mac_gc()` function (or the `fl_gc` variable).

20.5.3 Internationalization

All FLTK programs contain an application menu with, e.g., the About xxx, Hide xxx, and Quit xxx items. This menu can be internationalized/localized by any of two means.

- using the `Fl_Window::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.

20.5.4 OpenGL and 'retina' displays

It is possible to have OpenGL produce graphics at the high pixel resolution allowed by the so-called 'retina' displays present on recent Apple hardware. For this, call

```c
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

```c
glViewport(0, 0, w(), h());
```

by

```c
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.
20.5.5 Fl_Double_Window

OS X double-buffers all windows automatically. On OS X, Fl_Window and Fl_Double_Window are handled internally in the same way.

20.5.6 Mac File System Specifics

Resource Forks

FLTK does not access the resource fork of an application. However, a minimal resource fork must be created for OS X applications. Starting with OS X 10.6, resource forks are no longer needed.

Caution (OS X 10.2 and older):
When using UNIX commands to copy or move executables, OS X will NOT copy any resource forks! For copying and moving use CpMac and MvMac respectively. For creating a tar archive, all executables need to be stripped from their Resource Fork before packing, e.g. "DeRez fluid > fluid.r". After unpacking the Resource Fork needs to be reattached, e.g. "Rez fluid.r -o fluid".

It is advisable to use the Finder for moving and copying and Mac archiving tools like Sit for distribution as they will handle the Resource Fork correctly.

Mac File Paths

FLTK uses UTF-8-encoded UNIX-style filenames and paths.

See also

Mac OS X-specific symbols

20.6 The Wayland Interface

Cross-platform applications should bracket Wayland-specific source code between #ifdef FLTK_USE_WAYLAND / #endif and should ensure function fl_wl_display() returns non-NULL before calling Wayland-specific functions and using Wayland-specific symbols.

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 wid_window *fl_wl_xid(const Fl_Window *)
Returns a pointer to an FLTK-defined structure holding Wayland-related data created when a window gets show()'n, or NULL if not show()'n.

```c
Fl_Window *fl_wl_find(struct wld_window *wld_win)
```

Returns the Fl_Window that corresponds to the given Window, or NULL if not found.

```c
struct wl_surface *fl_wl_surface(struct wld_window *wld_win)
```

Returns a pointer to the struct wl_surface corresponding to a show()'n top-level window or subwindow.

```c
cairo_t *fl_wl_gc()
```

Drawing natively to a Wayland window: Within an overridden Fl_Widget::draw() method, or after a call to Fl_Window::make_current(), it's possible to draw using the Cairo library. Function fl_wl_gc() returns the adequate cairo_t* value. Regular FLTK coordinates, with top-left origin, are to be used. All FLTK-defined drawing functions (e.g., fl_rect(), fl_draw()) can be used too.

```c
void fl_close_display()
```

This closes the Wayland connection. You do not need to call this to exit. It may be useful to call this if you want your program to continue without the Wayland connection. You cannot open the display again, and cannot call any FLTK functions.

See also

wayland.H for all functions specific of the Wayland platform.

### 20.6.1 HiDPI display support

FLTK Wayland apps automatically scale according to the Wayland-defined, integer-valued scale factor. On a HiDPI display, it's enough to set this factor to 2 for any FLTK app to be drawn using twice as many pixels and thus to be as readable as it is on a regular display. With the gnome 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.
20.6 The Wayland Interface

20.6.2 Window icons

Standard FLTK functions Fl_Window::icon(const Fl_RGB_Image*), Fl_Window::icons(const Fl_RGB_Image*[ ], int), Fl_Window::default_icon(const Fl_RGB_Image*) and Fl_Window::default_icons(const Fl_RGB_Image*[ ], int) have no effect on the Wayland platform. The equivalent of a call to Fl_Window::default_icon(const Fl_RGB_Image*) to set the application-specific window icon can be obtained as follows, using FLTK's editor app as an example:

- create a text file named editor.desktop containing:

```
[Desktop Entry]
Version=1.0
Type=Application
Name=Editor
Name[fr]=Editeur
Comment=FLTK editor
Exec=editor %F
Icon=/path/to/icon/file/editor.svg
MimeType=text/plain
```

- The Name= line therein determines the string displayed when the app runs.

- Optionally, one or more Name[locale]= lines can be used to set locale-specific app names.

- The Icon= line accepts also .png files.

- Put this file in /usr/local/share/applications/ so it's available to all system users or in $HOME/.local/share/applications/ so it's available to a single user.

20.6.3 Window titlebars

Wayland supports both client-side window decoration (CSD), where client applications are responsible for drawing window titlebars, and server-side window decoration (SSD), where the Wayland compositor itself draws window titlebars. Among 4 tested Wayland compositors, Mutter (gnome's compositor) and Weston use CSD mode whereas the KWin and Sway compositors use SSD mode. When running in CSD mode, FLTK uses a library called libdecor to draw titlebars. The libdecor library has been conceived to use various plug-ins 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 21

Migrating Code from FLTK 1.3 to 1.4

This appendix describes the differences between FLTK 1.3.x and FLTK 1.4.x functions and classes and potential requirements to change source code.

We also explain how code can be made compatible so it can be compiled by both FLTK 1.3.x and 1.4.x.

If you need to migrate your code from prior FLTK versions to FLTK 1.4, then you should first consult the relevant appendices in the FLTK 1.3 online documentation or by downloading the FLTK 1.3 documentation. See https://www.fltk.org/doc-1.3/index.html and/or https://www.fltk.org/software.php, respectively.

21.1 Changes in Header Files

We strive to include only necessary header files in the public headers of the FLTK library to reduce dependencies and hence compile times.

We try to avoid including system header files as far as possible. Known exceptions are `<stdio.h>` where file system structures and functions are visible in the public API, for instance `FILE*`, and sometimes essential header files like `<stdlib.h>` and/or `<stddef.h>`. Some required system headers may be included in platform specific header files like `<FL/platform.H>` or `<FL/platform_types.h>`.

In earlier versions (1.3.x) some of the public FLTK headers included some not strictly required system headers by accident.

The consequence for building user programs with FLTK 1.4 is that if you require a system or FLTK header in your user program that you don't `#include` explicitly but which has been included by FLTK 1.3.x your FLTK 1.3 program may issue compiler errors or warnings about missing header files or missing declarations when compiled with FLTK 1.4.

This is not a fault of FLTK 1.4 but a fault of the source code that did not include all required headers.

In FLTK 1.4 inclusion of `<FL/Fl.H>` is no longer a strict requirement as it was required and documented in FLTK 1.3.x. In FLTK 1.4 you may still need to `#include `<FL/Fl.H>` if you are using enumerations or methods of class `Fl` like `Fl::run()` but there are exceptions where this header is included by other FLTK headers, like `Fl_Window.H` and other subclasses.

Suggested solution: include all FLTK and system header files your source code requires explicitly and don't rely on FLTK headers to include a particular header file. If you want your code to be as much as possible compatible with FLTK 1.3.x, then you should `#include `<FL/Fl.H>` as required by 1.3.x.

You don't need to include headers of base classes - this is done by all FLTK headers as required. Besides that you need to include some support headers if you use FLTK functions like `fl_choice()` and others. This is described in the function's documentation (if a required header is missing in the docs this is a bug).

If you follow these rules your program will be compatible with both FLTK 1.3.x and FLTK 1.4.x as long as you use only functions and classes defined in FLTK 1.3.
21.2  Fl_Preferences

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see Fl_Preferences::Fl_Preferences(Root, const char*, const char*) for details.

On Unix/Linux platforms new FLTK preference files are stored using the XDG Base Directory Specification which means in essence that user preference files are stored in the user's home directory under the subdirectory .config, i.e. in $HOME/.config/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.

21.3  Fl::add_timeout and friends

Since FLTK 1.4.0 internal timeout handling has been unified across platforms. This ensures equal timeout handling, improved accuracy of Fl::repeat_timeout(), and easier maintenance (less potential for errors).

This will very likely not affect user code, however there is one subtle exception on macOS and Windows: in FLTK 1.3.x these platforms used system timers to schedule timeouts. Since FLTK 1.4.0 all platforms use the same internal timer management that was previously only used on Unix/Linux/X11. The consequence of this change is that the FLTK event loop needs to be executed to trigger timeout events, i.e. you must either call Fl::wait() repeatedly or start the event loop with Fl::run().

Code that did not execute the event loop and relied on the system timers has never been cross platform compatible, i.e. it wouldn't work on Unix/Linux. An example would be code that opened a splash window, scheduled a timeout with Fl::add_timeout(), and waited for the timer event w/o running the FLTK event loop. Such code must be modified to execute Fl::run() and/or use Fl::wait().

21.4  New FL_OVERRIDE Macro

FLTK 1.4 defines a new macro FL_OVERRIDE as "override" if a recent C++ standard (C++11 or higher) is used to compile your code.

This macro is currently defined in FL/fl_attr.h but this may change in a future release. It is enough to '#include <FL/Fl.H>' to enable this macro.

Unfortunately Visual Studio does not define a meaningful value of __cplusplus to detect the C++ standard. Hence we use the Visual Studio version (2015 or higher) to decide whether we can define FL_OVERRIDE or not.

The FL_OVERRIDE macro is used to decorate declarations of overridden virtual methods in subclasses. Example code from FL/Fl_Window.H:
int handle(int) FL_OVERRIDE;
The FL_OVERRIDE macro translates to 'override' on newer compilers and to an empty string for older compilers.

We recommend to add this to your overridden virtual methods in subclasses derived from FLTK base classes (widgets) and to compile with C++ standard C++11 or higher to enable the compiler to detect some errors if methods are not overridden correctly.

You don't need to declare the overridden methods 'virtual' if you use FL_OVERRIDE or the keyword override.

Hint: For the GCC and clang compilers you can enable the warning '-Wsuggest-override' to detect where you may (want to) add the FL_OVERRIDE macro.

21.5 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 the declaration of copy() in your header file and in the implementation.

We suggest to add the new FL_OVERRIDE macro or the keyword 'override' (see above) to your own overridden method declarations to enable the compiler to detect such incompatibilities.

Code example in header file:

```cpp
class Your_Image {
    // ...
    copy() const FL_OVERRIDE;
    copy(int w, int h) const FL_OVERRIDE;
};
```

Note the 'const' attribute and the FL_OVERRIDE macro.
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
provides you legal permission to copy, distribute and/or modify the library.
Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty
for this free library. If the library is modified by someone else and passed on, we want its recipients to know that
what they have is not the original version, so that any problems introduced by others will not reflect on the original
authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies
distributing free software will individually obtain patent licenses, thus in effect transforming the program into 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.
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 if the following three conditions are met: (a) the distribution and use of the Library is restricted to non-commercial purposes, (b) the directory containing the Library contains a copy of the notice describing the restriction of use, and (c) you made clear to everyone associated with you that the Library as you received it might be bigger than the one you can make available to others.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE
COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
Chapter 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>boxtypex</td>
<td>browserx</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>fullcreen</td>
</tr>
<tr>
<td>gl_overlay</td>
<td>glpuzzle</td>
<td>hello</td>
<td>help_dialog</td>
<td>icon</td>
<td>iconize</td>
</tr>
<tr>
<td>image</td>
<td>inactive</td>
<td>input</td>
<td>input_choice</td>
<td>keyboard</td>
<td>label</td>
</tr>
<tr>
<td>line_style</td>
<td>list_visuals</td>
<td>mandelbrot</td>
<td>menubar</td>
<td>message</td>
<td>minimum</td>
</tr>
<tr>
<td>native-filechooser</td>
<td>navigation</td>
<td>offscreen</td>
<td>output</td>
<td>overlay</td>
<td>pack</td>
</tr>
<tr>
<td>pixmap</td>
<td>pixmap_browser</td>
<td>preferences</td>
<td>radio</td>
<td>resize</td>
<td>resizebox</td>
</tr>
<tr>
<td>rotated_text</td>
<td>scroll</td>
<td>shape</td>
<td>subwindow</td>
<td>sudoku</td>
<td>symbols</td>
</tr>
<tr>
<td>table</td>
<td>tabs</td>
<td>threads</td>
<td>tile</td>
<td>tiled_image</td>
<td>tree</td>
</tr>
<tr>
<td>twowin</td>
<td>unittests</td>
<td>utf8</td>
<td>valuators</td>
<td>windowfocus</td>
<td></td>
</tr>
</tbody>
</table>

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 tripiple 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_sslow

fast_sslow 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
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_Images` 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](image)

### 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)](image)

### 23.2.3 unittests
Select "drawing images" in the browser at the left side to see the image drawing example:

Figure 23.3 Image Drawing
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
FAQ (Frequently Asked Questions)

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():
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

Development of the FLTK library

• The Wayland backend for its developer
• Developer info for bundled libs
• Developer Information

25.1 The Wayland backend for its developer

This chapter describes how the Wayland backend of FLTK works from a developer's viewpoint.

25.1.1 Introduction to Wayland

Wayland usage involves communication via a socket between a client application and another process called the Wayland compositor which creates, moves, resizes and draws windows on the display. Diverse Wayland compositors exist. They can follow rather diverse logics. For example, FreeBSD offers Sway which is a tiling compositor where the display is always entirely filled with whatever windows are mapped at any given time. Compositors follow either the client-side decoration (CSD) rule where client apps draw window titlebars, or the server-side decoration (SSD) rule where the compositor draws titlebars. FLTK supports both CSD and SSD compositors. It bundles a library called libdecor charged of determining whether a CSD or a SSD compositor is active, and of drawing titlebars in the first case.

Wayland is divided in various protocols that a given compositor may or may not support, although they all support the core protocol. Each protocol adds functionality not available in the core protocol. The core protocol allows a client app to discover what protocols the connected compositor supports. Protocols can be stable, which means they have a defined API that will not change but can be expanded, or unstable. For example, mapping a window on a display is not done by the core protocol but by the xdg shell protocol which is stable. Unstable protocols are named beginning with letter 'z'. For example, the protocol FLTK uses to support CJK input methods is called zwp_text_input_v3 and is, unfortunately, unstable.

Wayland makes intensive use of the listener mechanism. A listener is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object; Wayland calls these functions when defined events occur (more at Listeners below).

Wayland differs noticeably from X11 in that the position of a window in the display is completely hidden to the client app. This prevents function Fl_Window::position() from having any effect on a top-level window. Wayland also prevents a client app from knowing whether a window is minimized: Fl_Window::show() has no effect on a minimized window. Subwindows can be positioned as usual relatively to their parent window. FLTK uses that for the small, yellow windows that display the new scale factor value when it's changed: these are created as short-lived subwindows centered above Fl::first_window(). Wayland allows to create popup windows positioned relatively to a previously mapped other window. This allows FLTK to position adequately menu and tooltip windows, see Menu windows and other popups.

Wayland uses a trick of its own to handle lists of linked records. It defines type struct wl_list and a few macros (wl_list_init(), wl_list_for_each(), wl_list_insert(), wl_list_for_each_safe(), wl_list_remove()) to manage linked lists. Records put in these lists must contain a member variable of type struct wl_list used to link records together and often named 'link'. Access to such a list is possible memorizing a value of type struct wl_list computed by macro wl_list_init(). Macro wl_list_for_each(arg1, arg2, arg3) allows to run through all list elements with:
• **arg1** is a pointer variable of the type of elements of the linked list;

• **arg2** is the address of a variable of type `struct wl_list` identifying the targeted list;

• **arg3** is the name of the member variable of these elements used to link them together.

For example, `wl_list_for_each()` can be used as follows to scan the linked list of all displays of the system (see `Fl_Wayland_Screen_Driver::output()`):

```c
Fl_Wayland_Screen_Driver::output *output;
Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
wl_list_for_each(output, &(scr_driver->outputs), link) {
    // ... work with output, an item of the linked list of all displays in the system ...
}
```

Overall, and ignoring for now OpenGL usage, FLTK interacts with Wayland in 3 ways:

• Calling C functions of the `libwayland-client.so`, `libwayland-cursor.so` and `libxkbcommon.so` shared libraries and of the bundled libdecor library. The names of these functions begin with `wl_`, `xkb_` or `libdecor_`.

• Being called by these libraries via the 'listener' mechanism.

• Listening, after a call to `Fl::add_fd()`, to data sent by the compositor to the client via the socket.

The core protocol defines also a number of mostly opaque structures whose names begin with `wl_`. The names of symbols and types defined by the other protocols FLTK uses begin with `xdg_`, `xkb_` or `libdecor_`.

25.1.2 Building libfltk as a Wayland client

**Classes** `Fl_Wayland_Window_Driver`, `Fl_Wayland_Screen_Driver`, `Fl_Wayland_Graphics_Driver`, `Fl_Wayland_Copy_Surface_Driver`, `Fl_Wayland_Image_Surface_Driver` and `Fl_Wayland_GL_Window_Driver` and file `fl_wayland_platform_init.cxx` contain all the Wayland-specific code of the FLTK library. This code is located at `src/drivers/Wayland/` in the FLTK source tree. A single C++ source file generally contains all the code of a given class. The code related to copy, paste and drag-and-drop operations, however, is gathered in file `fl_wayland_clipboard_dnd.cxx` and contains a few member functions of class `Fl_Wayland_Screen_Driver`. Furthermore, class `FLUnix_System_Driver` is used by both the Wayland and the X11 FLTK platforms.

The public C API to Wayland, xcb, EGL and libdecor libraries are obtained with

```c
#include <wayland-client.h>
#include <wayland-cursor.h>
#include <xkbcommon/xkbcommon.h>
#include <xkbcommon/xkbcommon-compose.h>
#include <linux/input.h> // for BTN_LEFT, BTN_RIGHT, BTN_MIDDLE
#include "../../../libdecor/src/libdecor.h" // libdecor/plugin
#include "../../../libdecor/src/libdecor-plugin.h"
```

```c
#if HAVE_GL
#include <wayland-eql.h>
#include <EGL/eql.h>
#endif // HAVE_GL
```

as necessary.

File `README.Wayland.txt` details what software packages are needed on Debian-based, Fedora and FreeBSD systems for FLTK to use Wayland. Wayland protocols are packaged as XML files accompanied by a utility program, `wayland-scanner`, able to generate a header file and a necessary glue C source file from a given XML file. For example, for FLTK to use the xdg shell protocol, these commands are run at build time to generate a .c file that will be compiled into libfltk and a header file that FLTK code will include:

```bash
PROTOCOLS='pkg-config --variable=pkgdatadir wayland-protocols'
wayland-scanner private-code $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-protocol.c
wayland-scanner public-code $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-client-protocol.c
```

Similar operations are performed for FLTK to use protocols `xdg-decoration-unstable.v1` and `text-input-unstable.v3`.

25.1.3 The hybrid Wayland/X11 platform

The Wayland platform of FLTK is normally a two-legged hybrid able to use either Wayland or X11 and to choose between these possibilities at run-time, without any change to the client application. The Wayland/X11 hybrid
is essentially a version of the FLTK library containing both all Wayland-specific and all X11-specific code. This creates the constraint that Wayland and X11 cannot use the same type name for different purposes or the same symbol name. That is why function \texttt{fl_xid(const Fl\_Window*)} is deprecated in FLTK 1.4 and replaced by \texttt{fl\_wl\_xid()} for Wayland and \texttt{fl\_x11\_xid()} for X11. Also, global variable \texttt{Window fl\_window} is not used by the Wayland platform which instead uses \texttt{static struct wld\_window *Fl\_Wayland\_← Window\_Driver:: wld\_window}. The FLTK library contains also a short source file, \texttt{fl\_wayland← platform\_init.cxx}, that determines, at startup time, whether the app will run as a Wayland or as an X11 client. Function \texttt{attempt\_wayland()} therein performs this choice as follows:

- if the app defines a global bool variable called \texttt{fl\_disable\_wayland} and this variable is true, the X11 leg is chosen;
- if environment variable \texttt{FLTK\_BACKEND} is defined to string "wayland", the Wayland leg is chosen;
- if environment variable \texttt{FLTK\_BACKEND} is defined to string "x11", the X11 leg is chosen;
- otherwise, a connection to a Wayland compositor is attempted; if it's successful, the Wayland leg is chosen; if it's not, the X11 leg is chosen.

The first condition listed above is meant to facilitate transition to FLTK 1.4 of source code written for FLTK 1.3 and containing X11-specific code: it's enough to put \texttt{FL\_EXPORT bool fl\_disable\_wayland = true;} anywhere in the source code, for the app to run with 1.4, using the x11 leg of the hybrid platform, without any other change in the source code nor to the application's environment.

In special situations, such as with embedded systems equipped with the Wayland software but lacking the X11 library, it's possible to build the FLTK library such as it contains only the Wayland backend. This is achieved building FLTK with \texttt{cmake -D OPTION\_WAYLAND\_ONLY=on} or with \texttt{configure -disable-x11}.

The rest of this chapter describes what happens when the Wayland leg has been chosen.

### 25.1.4 Listeners

A Wayland 'listener' is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object; Wayland calls these functions when defined events occur, and transmits relevant information to the client app as parameters of these calls. Each listener is associated to its corresponding Wayland object, usually right after the object's creation, by a call to a specific Wayland function named following the form \texttt{wl\_XXX\_add\_listener()}. For example, this code:

```c
static void surface\_enter(......) { ...... } // called when a surface enters a display
static void surface\_leave(......) { ...... } // called when a surface leaves a display
static struct wl\_surface\_listener surface\_listener = {
    surface\_enter,
    surface\_leave,
};
some\_pointer\_type pter\_to\_data;
struct wl\_surface *my\_wl\_surface;
my\_wl\_surface = wl\_compositor\_create\_surface(scr\_driver=wl\_compositor);
w\_wl\_surface\_add\_listener(my\_wl\_surface, &surface\_listener, pter\_to\_data);
```

creates a Wayland object of type \texttt{struct wl\_surface}, and associates it with a 2-member listener called \texttt{surface\_listener}. After this, Wayland is expected to call the 2 listener members, \texttt{surface\_enter} or \texttt{surface\_leave}, each time \texttt{my\_wl\_surface} will enter or leave, respectively, a display. The arguments of these calls, not detailed here, allow the member functions to identify which surface enters or leaves which display. The \texttt{wl\_surface\_add\_listener()} call above also associates \texttt{pter\_to\_data} to \texttt{my\_wl\_surface} as \texttt{user\_data}. The \texttt{wl\_surface} object's "user data" can be obtained later calling function \texttt{wl\_surface\_get← user\_data()}.

Wayland function \texttt{wl\_proxy\_get\_listener()} returns a pointer to a Wayland object's listener provided that object is transmitted cast to type \texttt{struct wl\_proxy \*}. This gives a handy way to distinguish FLTK-created Wayland objects from objects of other origin: the listener of an FLTK-created object is a known FLTK listener. For example, function \texttt{Fl\_Wayland\_Window\_Driver::surface\_to\_window()} uses this possibility calling \texttt{wl\_proxy\_get\_listener( (struct wl\_proxy \*)wl\_surface \}) for any object of type \texttt{struct \_listener static variable.

### 25.1.5 Opening a Wayland connection

Establishing a Wayland connection requires environment variable \texttt{XDG\_RUNTIME\_DIR} to be defined and to point to a directory containing a socket connected to a Wayland compositor. This variable is usually set by the login
procedure of Wayland-friendly desktops. The name of the Wayland socket is determined as follows:

- the client may call `Fl::display(const char *display_name)` before `fl_open_display()` runs or use the `-display` command line argument and transmit there the socket name;
- environment variable `WAYLAND_DISPLAY` can be defined to the socket name;
- otherwise, "wayland-0" is used.

What socket is selected determines what compositor will be used by the client application.

Function `Fl_Wayland_Screen_Driver::open_display_platform()` establishes the connection to the Wayland socket identified above calling `wl_display_connect(NULL)` which returns a struct `wl_display` pointer or NULL in case of failure. Such NULL return is the hint that allows the FLTK display opening procedure of the Wayland/X11 hybrid to recognize when Wayland access is not possible and to fallback to X11.

Then, function `wl_registry_add_listener()` associates a 2-member listener, whose 1st member, `registry_handle_global()`, will be called by Wayland a number of times to indicate each time a protocol supported by the compositor or a system feature such as displays and keyboards. This code allows to run the client until all calls to `registry_handle_global()` have occurred:

```c
static void sync_done(void *data, struct wl_callback *cb, uint32_t time) {
    struct wl_callback **data = (struct wl_callback **)data;
    wl_callback_destroy(cb);
}

static const struct wl_callback_listener sync_listener = {
    sync_done,
};

struct wl_callback *registry_cb = wl_display_sync(wl_display);
while (registry_cb) wl_display_dispatch(wl_display);
```

A pointer to an object of type `struct wl_callback` created by function `wl_display_sync()` is assigned to variable `registry_cb`. Then a 1-member listener is attached to this object. Wayland will run this listener's member function, `sync_done()`, after all calls to `registry_handle_global()` have occurred. Function `sync_done()` sets to null variable `registry_cb` and destroys the `wl_callback`. Finally, function `wl_display_dispatch()` is called as long as variable `registry_cb` is not null. This makes Wayland process all its pending requests until `sync_done()` runs.

The prototype of function `registry_handle_global()` is:

```c
static void registry_handle_global(void *user_data, struct wl_registry *wl_registry,
                                  const char *interface, uint32_t version)
```

Each time Wayland calls `registry_handle_global()`, `interface` and `version` give the name and version of a component or feature of the Wayland system. It's necessary to call each time function `wl_registry_bind()` which returns a pointer to a Wayland structure that will be the client's access point to the corresponding Wayland protocol or system feature. This pointer is stored in a dedicated member variable of the unique `Fl_Wayland_Screen_Driver` object of an FLTK app, or of another object accessible from this object. For example, when `interface` equals "wl_compositor", the value returned by `wl_registry_bind()` is stored as member `wl_compositor` of the `Fl_Wayland_Screen_Driver` object. `registry_handle_global()` also identifies whether the Mutter, Weston, or KWin compositor is connected and stores this information in static member variable `Fl_Wayland_Screen_Driver::compositor`.

Wayland calls `registry_handle_global()` with its parameter `interface` equals to "wl_output" once for each screen connected to the system. Each time, an object of type `struct wl_output` is created, to which a 4-member listener is associated by function `wl_output_add_listener()`. The 3rd member of this 4-function listener, `output_done()`, runs after all initialization steps of the screen have completed and turns to true member `done()` of a record of type `struct Fl_Wayland_Screen_Driver::output` associated to the screen. Function `sync_done()` mentioned above therefore also calls `wl_display_dispatch()` until the done member of all `Fl_Wayland_Screen_Driver::output` records are true. Overall, after return from function `sync_done()`, FLTK has been made aware of all optional protocols and features of its connected Wayland compositor, and has initialized all screens of the system.

Finally, function `wl_display_get_fd()` is called to obtain the file descriptor of the Wayland socket and a call to `Fl::display(const char *display_name)` makes FLTK listen to this descriptor in `FL_READ` mode and associates function `wayland_socket_callback()` from file `Fl_Wayland_Screen_Driver.cxx` with it. This function calls `wl_display___dispatch()` which reads and interprets data available from the file descriptor, and calls corresponding listeners. The `wl_display_dispatch()` call is repeated as long as data are available for reading.

The event loop is run by function `Fl_Unix_System_Driver::wait()` which is used by both the Wayland and X11 FLTK backends. Among various tasks, this function waits for data arriving on the file descriptors FLTK is listening. Overall, the event loop of the Wayland backend is nearly exactly the same as that used by the X11 backend. The Wayland backend differs only in the callback function handling data read from the Wayland con-
section socket, and in overridden functions Fl_Wayland_Screen_Driver::poll_or_select_with←_delay() and Fl_Wayland_Screen_Driver::poll_or_select().

25.1.6 Wayland windows and surfaces

Wayland defines objects called surfaces of type struct wl_surface. A Wayland surface "has a rectangular area which may be displayed on zero or more displays, present buffers, receive user input, and define a local coordinate system". Buffers allow the client app to draw to surfaces (see Wayland buffers). FLTK creates a surface each time an Fl_Window is show'(n). Static member function Fl_Wayland_Window_Driver::surface_to←_window(struct wl_surface *) gives the Fl_Window* corresponding to the surface given in argument. FLTK recognizes 4 distinct kinds of surfaces named DECORATED, UNFRAMED, POPUP and SUBWINDOW. DECORATED are toplevel windows with a titlebar. UNFRAMED have no titlebar. POPUP correspond to menus and tooltips, SUBWINDOW to an Fl_Window. Function Fl_Wayland_Window←_Driver::makeWindow() creates all these surfaces, creates for each a record of type struct wld←_window (see wld_window), and stores the window kind in member variable kind of this record. Member variable xid of the window's Fl_X record stores the adress of this record.

Except for SUBWINDOW's, each surface needs a Wayland object of type struct xdg_surface used to make it become a mapped window and stored in member xdg_surface of the window's wld_window record. For DECORATED windows, this object is created inside libdecor and transmitted to FLTK by function libdecor←_frame_get_xdg_surface(). For UNFRAMED and POPUP windows, it's created by function xdg_wm←base_get_xdg_surface(). Finally, each surface is also associated to one more Wayland object whose type varies with the window's kind. These explain this part of the wld_window record:

union {
    struct libdecor_frame *frame; // created for DECORATED by libdecor_decorate()
    struct wl_subsurface *subsurface; // created for SUBWINDOW by wl_subcompositor_get_subsurface()
    struct xdg_popup *xdg_popup; // created for POPUP by xdg_surface_get_popup()
    struct xdg_toplevel *xdg_toplevel; // created for UNFRAMED by xdg_surface_get_toplevel()
};

Except for SUBWINDOW's, each surface is associated to a 'configure' function that Wayland calls one or more times when the window is going to be mapped on the display. The 'configure' function of DECORATED surfaces is handle_configure() which is the 1st member of a 4-member listener named libdecor←_frame associated to a decorated window when it's created calling libdecor_decorate(). Finally, a call to libdecor←_frame_map() triggers the process of mapping the newly created DECORATED surface on a display. Wayland calls handle_configure() twice during this process. The first handle_configure() run allows to set the window's xdg_surface object which is returned by function libdecor←_xdg_surface(). FLTK distinguishes the first from the second run of handle_configure() by looking at the xdg_surface member variable that's NULL at the beginning of the 1st run and not NULL later. Wayland calls handle_configure() also during operations such as resizing, minimizing (see below). With the help of a few calls to libdecor functions, FLTK obtains in this function all needed information about the size and state of the mapped window. The 'configure' functions of UNFRAMED and POPUP surfaces are xdg_surface←_configure(), xdg_toplevel_configure() and popup_configure(). The mapping process of these surfaces is triggered by a call to wl_surface_commit(). These 'configure' functions transmit effective window size information to FLTK. Also, they are where the window's Fl_Wayland_Driver::wait_for←_expose_value member variable is set to 0 to indicate that the window has been mapped to display. Caution←: there are some small differences between how and when the various Wayland compositors call handle←_configure().

When a decorated window changes size, whatever the cause of it, Wayland calls handle_configure() which sets member variable Fl_Wayland_Window_Driver::in_handle_configure to true and calls the window's virtual resize() function which ultimately runs Fl_Wayland_Window_Driver::resize() which calls Fl_Group::resize() to perform FLTK's resize operations and Fl_Wayland_Graphics_Driver←_buffer_release() to delete the existing window buffer that's not adequate for the new window size. At the end of the run of handle_configure(), in_handle_configure is set back to false. When the window size change is caused by the app itself calling the window's resize() function, Fl_Wayland_Window←_Driver::in_handle_configure is false. This allows Fl_Wayland_Window_Driver::resize() to detect that Wayland needs to inform of the desired size change, which gets done by a call to libdecor←_frame_commit(). Wayland later calls handle_configure() and events described above unfold. Wayland generally does not provide a way to control where the compositor should map a window in the system displays. Nevertheless, for multi-display systems, Wayland allows to control on what display should the compositor map a fullscreen window. That is done inside function handle_configure() which calls libdecor←_frame_set_fullscreen() for DECORATED windows and inside function xdg_toplevel_configure() which
calls `xdg_toplevel_set_fullscreen()` for UNFRAMED. The struct `wl_output` pointer for the targeted display is transmitted as 2nd argument of these calls.

### 25.1.7 Menu windows and other popups

Menu windows, tiny menu title windows, and tooltip windows are implemented using Wayland's popup mechanism which allows to position a popup window relatively to a previously mapped window, itself a popup or another kind of window, with the restriction that any popup must overlap or at least touch that other window. Member function `Fl_Wayland_Window_Driver::makeWindow` calls member function `Fl_Wayland_Window_Driver::process_menu_or_tooltip` to create all popups.

This function gets called after FLTK has computed using a given algorithm the desired (x,y) position of the popup window's top-left corner, using coordinates centered on the top-left corner of the toplevel window from which the popup originates. This algorithm is able to prevent popups from being positioned beyond the screen borders under the assumption that the position of a toplevel window inside a screen is known. While this assumption holds for other platforms, it does not for the Wayland platform. The FLTK code for the Wayland platform therefore modifies the algorithm that FLTK uses to compute the position of menu windows. The key information used by this algorithm is obtained by member function `Fl_Window_Driver::menu_window_area` which computes the coordinates of the rectangle where menu windows are allowed to be positioned. Under other platforms, this function just returns the origin and size of the work area of the screen in use. In contrast, the Wayland platform handles two situations differently:

- For menu windows that are not taller than the display in use, the Wayland-overridden member function `Fl_Wayland_Window_Driver::menu_window_area` returns large negative origin and large width and height values. This lets the standard FLTK algorithm position the menu relatively to its window of origin without concern about screen limits, and relies on Wayland's constraint mechanism described below to prevent the menu from going beyond these limits, without FLTK having to know where they are.

- Menu windows taller than the screen where they are mapped need special handling described in detail in a comment above the source code of function `Fl_Wayland_Window_Driver::process_menu_or_tooltip`.

Function `Fl_Wayland_Window_Driver::process_menu_or_tooltip` first computes `origin_win`, pointer to the `Fl_Window` relatively to which the popup is to be positioned. `Window origin_win` is the parent menu window when the popup is a sub-menu; it's the tiny windowtitle when the popup is a menu with a title; otherwise, it's the window containing the point of origin of the popup. An object of type `struct xdg_positioner` created by function `xdg_wm_base_create_positioner()` is used to express the rules that will determine the popup position relatively to `origin_win` as follows:

- Function `xdg_positioner_set_anchor_rect()` determines a rectangle in `origin_win` relatively to which the popup is to be positioned. When the popup to be created is a menu window spawned by an `Fl_Menu_Bar`, that rectangle is the full area of the menu title window. Otherwise, that rectangle is an adequately located point.

- Function `xdg_positioner_set_size()` sets the popup size.

- The `xdg_positioner_set_anchor(positioner, XDG_POSITIONER_ANCHOR_BOTTOM_LEFT);` and `xdg_positioner_set_gravity(positioner, XDG_POSITIONER_GRAVITY_BOTTOM_RIGHT);` calls position the popup so that its top-left corner is initially below and at right of the bottom-left corner of the `origin_win`'s anchor rectangle.

- The call to `xdg_positioner_set_offset()` further changes the popup vertical position.

- The call to `xdg_positioner_set_constraint_adjustment()` uses constraint flags `XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_SLIDE_X` and `XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_SLIDE_Y` which mean that the compositor will move the popup horizontally and vertically if its initial position would make it expand beyond the edges of the screen. Furthermore, flag `XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_FLIP_Y` is added when the popup is a menu window spawned by an `Fl_Menu_Bar`; this has the popup flipped above the `Fl_Menu_Bar` if there's not enough screen room below it for the popup.

- Finally, a call to function `xdg_surface_get_popup()` creates the popup accounting for position rules listed above. The positioner is then deleted by `xdg_positioner_destroy()`, a listener is associated
25.1 The Wayland backend for its developer

...to the popup surface with xdg_popup_add_listener(), and a call to wl_surface_commit() triggers the mapping of the popup on the display.

Overall, the expected coordinates of the top-left corner of the popup relatively to origin_win are popup_x, popup_y. They are memorized in a record of FLTK-defined type struct win_positioner that's associated to the popup listener. When the compositor maps the popup, function popup_configure, the first element of the popup listener, runs and receives as arguments the coordinates of the popup top left and its size. These values account for the positioning constraints of the popup which may have moved it to avoid screen borders. This function can therefore detect whether constraints applied have modified the effective popup location in comparison to the expected coordinates which are available as member variables of the struct win_positioner record mentioned above. That's key to the handling by FLTK of tall menu windows.

Groups of popups containing a menutitle, the associated menuwindow, and optionally a submenu window and that don't belong to an Fl_Menu_Bar are mapped in a different order: the menuwindow is mapped first, and the menutitle is mapped second above it as a child popup. Function Fl_Window_Driver::is_floating←title() detects when such a menutitle is created, static member variable previous_floatingtitle is assigned the value of this menutitle, and the menutitle is mapped only after the menuwindow has been mapped, as a child of it. This positions better the popup group in the display relatively to where the popup was created.

25.1.8 Fl_Wayland_Graphics_Driver and Fl_Cairo_Graphics_Driver

The Wayland platform of FLTK uses an Fl_Wayland_Graphics_Driver object for all its on-screen drawing operations. This object is created by function Fl_Graphics_Driver::newMainGraphics←Driver() called by Fl_Display_Device::display_device() when the library opens the display. New Fl_Wayland_Graphics_Driver objects are also created for each Fl_Image_Surface and each Fl_Copy_Surface used, and deleted when these objects are deleted.

Class Fl_Wayland_Graphics_Driver derives from class Fl_Cairo_Graphics_Driver which implements all the FLTK drawing API for a Cairo surface. Function Fl_Wayland_Graphics_Driver::cairo←init() creates the Cairo surface used by each Fl_Wayland_Graphics_Driver object by calling cairo←_image_surface_create_for_data() for the window's or offscreen's draw_buffer (see below). Class Fl_Cairo_Graphics_Driver is also used by the X11 leg of the hybrid Wayland-X11 platform because this leg draws to the display with an Fl_X11_Cairo_Graphics_Driver object which derives from class Fl_Cairo_Graphics_Driver. Finally, Fl_Cairo_Graphics_Driver is also used, in the form of an object from its derived class Fl_PostScript_Graphics_Driver, when the hybrid Wayland-X11 platform draws PostScript, or when the classic X11 platform uses Pango and draws PostScript. This happens when classes Fl_PostScript_File_Device and Fl_Printer are used.

25.1.9 Wayland buffers

Wayland uses buffers, objects of type struct wl_buffer, to draw to surfaces. In principle, one or more buffers can be associated to a surface, and functions wl_surface_attach() and wl_surface_commit() are called to first attach one such buffer to the surface and then inform the compositor to map this buffer on the display. Wayland buffers can use various memory layouts. FLTK uses WL_SHM_FORMAT_ARGB8888, which is the same layout as what Cairo calls CAIRO_FORMAT_ARGB32.

FLTK calls function Fl_Wayland_Window_Driver::make_current() before drawing to any Fl_Window. Member buffer of this Fl_Window's struct wl_window (see wl_window) is NULL when the window has just been created or resized. In that case, FLTK calls member functions create_shm_buffer() and cairo←_init() of Fl_Wayland_Graphics_Driver to create

- a Wayland buffer;
- a Cairo image surface.

Each of these two objects encapsulates a byte array of the same size and the same memory layout destined to contain the Fl_Window's graphics. The Cairo surface object is where FLTK draws. The Wayland buffer is what Wayland maps on the display. FLTK copies the Cairo surface's byte array to the Wayland buffer's byte array before beginning the mapping operation. If width and height are a window's dimensions in pixels,

\[
\text{int stride = cairo_format_stride_for_width(CAIRO_FORMAT_ARGB32, width);} \\
\text{int size = stride * height;}
\]

give size, the common size of both byte arrays.

Section Buffer factories below details how FLTK creates wl_buffer objects.
FLTK associates to each surface a struct `fl_wld_buffer` (see `fl_wld_buffer`) containing a pointer to the byte array of the Cairo image surface (member `draw_buffer`), information about the Wayland buffer (members `wl_buffer` and `data`), the common size of the Cairo surface’s and Wayland buffer’s byte arrays (member `data_size`), and other information. A pointer to this struct `fl_wld_buffer` is memorized as member `buffer` of the `Fl_Window`’s `wld_window`. All drawing operations to the `Fl_Window` then modify the content of the Cairo image surface.

Function `Fl_Wayland_Window_Driver::flush()` is in charge of sending FLTK graphics data to the display. That is done by calling function `Fl_Wayland_Graphics_Driver::buffer_commit()` which copies the byte array of the Cairo surface to the Wayland buffer’s starting memory address, and calls functions `wl_surface_attach()` and `wl_surface_commit()`. Before calling `Fl_Window::flush()`, FLTK has computed a damaged region. If that region is not null, `Fl_Wayland_Graphics_Driver::buffer_commit()` copies only the damaged parts of the Cairo surface to the Wayland buffer and calls function `wl_surface_damage_buffer()` for these parts to inform the compositor of what parts of the surface need its attention.

An important detail here is that FLTK uses Wayland’s synchronization mechanism to make sure the surface’s `wl_buffer` is not changed until the surface is fully mapped on the display. This 3-step mechanism works as follows:

- `Fl_Wayland_Graphics_Driver::buffer_commit()` first calls function `wl_surface_frame()` to obtain a struct `wl_callback` object and stores it as member `cb` of the surface’s `fl_wld_buffer`.
- Then it calls `wl_callback_add_listener()` to associate this object to the FLTK-defined, callback function `surface_frame_done()` that Wayland calls when it’s ready for another mapping operation.
- Finally `surface_frame_done()` destroys the `wl_callback` object by function `wl_callback_destroy()` and sets member `cb` to NULL.

This procedure ensures that FLTK never changes the surface’s Wayland buffer while it’s being used by the compositor because `Fl_Wayland_Window_Driver::flush()` checks that `cb` is NULL before calling `Fl_Wayland_Graphics_Driver::buffer_commit()`. If it’s not NULL, FLTK calls function `wl_callback_destroy()` which instructs the compositor to abort the mapping operation and to get ready for processing of a new byte buffer.

FLTK supports progressive drawing when an app calls function `Fl_Window::make_current()` at any time and then calls the FLTK drawing API. This is made possible in function `Fl_Wayland_Window_Driver::make_current()` with

```c
// to support progressive drawing
if (!Fl_Wayland_Window_Driver::in_flush_ && window->buffer && (!window->buffer->cb) && !wait_for_expose_value ) {
  Fl_Wayland_Graphics_Driver::buffer_commit(window);
}
```

Thus, `buffer_commit()` runs only when `cb` is NULL. If an app rapidly performs calls to `Fl_Window::make_current()` and to drawing functions, FLTK will copy `draw_buffer` to the Wayland buffer and instruct Wayland to map it to the display when `cb` is NULL which means that the compositor is ready to start performing a mapping operation. In contrast, FLTK will only modify `draw_buffer` when `cb` is not NULL, letting the compositor complete its ongoing mapping task. For example, FLTK’s mandelbrot test app set to fullscreen can be seen to progressively fill its window from top to bottom by blocks of lines, each block appearing when the compositor is ready to map a new buffer.

When the compositor is not ready, the app does not block but continues computing and drawing in memory but not on display more lines of the desired Mandelbrot graph.

### 25.1.10 Buffer factories

Wayland names `buffer factory` a software procedure that constructs objects of type `struct wl_buffer` for use by a client application. FLTK creates a `wl_buffer` object each time an `Fl_Window` is mapped on a display or resized. That’s done by member function `Fl_Wayland_Graphics_Driver::create_shm_buffer()` which follows this 3-step procedure to create a “buffer factory” for FLTK and to construct Wayland buffers from it:

- Libdecor function `os_create_anonymous_file(off_t size)` creates an adequate file and mmap’s it. This file lives in RAM because it is created by function `memfd_create()`. FLTK sets this file size to 10 MB unless the size of the buffer to be created is larger; in that case the anonymous file is sized to twice the buffer size.
- Wayland function `wl_shm_create_pool()` shares this mmap’ed memory with the Wayland compositor and returns an object of type `struct wl_shm_pool` which encapsulates this memory. A record of type `struct Fl_Wayland_Graphics_Driver::wld_shm_pool_data` is created and associated to
the newly created \texttt{wl\_shm\_pool} by \texttt{wl\_shm\_pool\_set\_user\_data()}. This record stores the starting address (pool\_memory) and size (pool\_size) of the pool's encapsulated memory. The record also contains member buffers of type \texttt{struct\ wl\_list} which stores the access point to the linked list of \texttt{wl\_buffer} objects that will be created from the \texttt{wl\_shm\_pool}.

- A variable named chunk\_offset represents the offset within the pool's shared memory available for the buffer being constructed. It equals 0 when the pool has just been created and is updated as detailed below when one or more buffers have been previously created from the pool. A record of type \texttt{struct\ fl\_wld\_buffer} is created. This record will contain (member \texttt{wl\_buffer}) the address of a \texttt{wl\_buffer} object that's created by function \texttt{wl\_shm\_pool\_create\_buffer()}. This \texttt{wl\_buffer} object encapsulates a section of a given size of the pool's shared memory beginning at offset chunk\_offset in it. Quantity pool\_memory + chunk\_offset is therefore the address of the beginning of the mmap'ed memory section encapsulated by this \texttt{wl\_buffer}. Member \texttt{shm\_pool} of the newly constructed \texttt{fl\_wld\_buffer} object is set to the address of the current \texttt{wl\_shm\_pool} object. This record is added to the head of the linked list of current pool's buffers by a call to \texttt{wl\_list\_insert()}. At that point, a \texttt{struct\ fl\_wld\_buffer} record is part of the linked list of all such records corresponding to \texttt{wl\_buffer} objects created from the same \texttt{wl\_shm\_pool} object, and member \texttt{shm\_pool} of this record gives the address of this \texttt{wl\_shm\_pool}. When a new \texttt{struct\ fl\_wld\_buffer} record is to be created,

\begin{verbatim}
struct \texttt{wld\_shm\_pool\_data} *pool_data = (struct \texttt{wld\_shm\_pool\_data} *)\texttt{wl\_shm\_pool\_get\_user\_data(pool);}
struct \texttt{fl\_wld\_buffer} *record = \texttt{wl\_container\_of(pool\_data->buffers.next, record, link);}
int chunk\_offset = (char*)record->data - pool\_data->pool\_memory) + record->data\_size;
\end{verbatim}


gives the offset within the current pool's mmap'ed memory available for a new \texttt{wl\_buffer}. Macro \texttt{wl\_container\_of()} gives the address of a record belonging to a linked list of records of the same type.

A window's \texttt{wl\_buffer} is re-used each time the window gets redrawn, and is destroyed by function \texttt{Fl\_Wayland\_Graphics\_Driver::buffer\_release()} when \texttt{Fl\_Window::hide()} runs or the window is resized. Function \texttt{Fl\_Wayland\_Graphics\_Driver::buffer\_release()} destroys the \texttt{wl\_buffer} with \texttt{wl\_buffer\_destroy()} and removes the corresponding \texttt{fl\_wld\_buffer} record from the linked list of buffers from the same \texttt{wl\_shm\_pool}. Since new \texttt{fl\_wld\_buffer} records are added at the head of the linked list, and since the record at the head of this list is used to compute the offset within the pool's mmap'ed memory available for a new \texttt{wl\_buffer}, destruction of the last created \texttt{wl\_buffer} allows to re-use the destroyed buffer's pool's memory for a new \texttt{wl\_buffer}. When the linked list results empty, the \texttt{wl\_shm\_pool} is destroyed by \texttt{wl\_shm\_pool\_destroy()}, the pool's mmap'ed memory is munmap'ed, and the pool's associated \texttt{struct\ wld\_shm\_pool\_data} is freed.

If the sum of chunk\_offset plus the buffer size is larger than the current pool's size when function create\_shm\_buffer() is called, chunk\_offset is reset to 0, and a new \texttt{wl\_shm\_pool} object is created and used by FLTK's "buffer factory". This mechanism allows to access new mmap'ed memory when chunk\_offset reaches the end of the previous mmap'ed section.

Wayland uses also \texttt{wl\_buffer} objects to support cursors. FLTK uses the "buffer factory" described here when creating custom cursors (see \texttt{custom\_cursor}) with function \texttt{Fl\_Wayland\_Window\_Driver::set\_cursor()} which calls \texttt{create\_shm\_buffer()}. In contrast, standard shaped-cursors (e.g., \texttt{FL\_CURSOR\_INSERT}) use their own "buffer factory" inside Wayland functions such as \texttt{wld\_cursor\_theme\_get\_cursor()}. Therefore, the fact that the \texttt{wl\_buffer} objects behind standard cursors are never destroyed doesn't prevent disused \texttt{struct\ wld\_shm\_pool} objects from being freed because those buffers come from a distinct "buffer factory". The "buffer factory" described here is also used by function \texttt{offscreen\_from\_text()} when displaying dragged text in a DnD operation.

### 25.1.11 Displays and HighDPI support

Wayland uses the concept of \texttt{seat} of type \texttt{struct\ wl\_seat} which encompasses displays, a keyboard, a mouse, and a trackpad. Although Wayland may be in principle able to deal with several seats, FLTK's Wayland platform is conceived for one seat only. That seat may contain one or more displays, which Wayland calls \texttt{outputs}, of type \texttt{struct\ wl\_output}.

As written above, function \texttt{registry\_handle\_global()} discovers the available seat at start-up time. This function also associates a listener to each display connected to the system by calling function \texttt{wl\_output\_add\_listener()}. This listener's member functions run at program startup when Wayland discovers its displays (see \texttt{Opening\ a\ Wayland\ connection}). Member \texttt{output\_scale} runs also when the display is resized and \texttt{output\_mode} also when the Wayland scale factor (see below) is changed. FLTK defines type \texttt{struct\ Fl\_Wayland\_Screen\_Driver::output} to store display size and scaling information. One such record is created for
Development of the FLTK library

Each display. These records are put in a `struct wl_list` accessible from member `outputs` of the single `Fl_Wayland_Screen_Driver` object.

FLTK uses 2 distinct scaling parameters for each display:

- **int wld_scale;** This member variable of the `struct Fl_Wayland_Screen_Driver::output` record typically equals 1 for standard, and 2 for HighDPI displays. Its value is set by the Wayland compositor for each display with the effect that 1 Wayland graphics unit represents a block of `nxn` pixels when the value is `n`. Another effect is that a drawing buffer for a surface of size `WxH` units contains `W * n * H * n * 4` bytes. This is enough to make FLTK apps HighDPI-aware because the Wayland compositor automatically initializes parameter `wld_scale` to the value adequate for each display’s DPI. Under the gnome desktop, this parameter is visible in the “Settings” app, “Displays” section, “Scale” parameter which is 200% on HighDPI displays.

- **float gui_scale;** This other member variable is where FLTK’s own GUI scaling mechanism with `ctrl/+/-/0/` keystrokes and with environment variable `FLTK_SCALING_FACTOR` operates: when FLTK is scaled at 150%, `gui_scale` is assigned value 1.5. Function `Fl_Wayland_Screen_Driver::scale(int n, float f)` assigns value `f` to the `gui_scale` member variable of display # `n`. This variable is used by function `Fl_Wayland_Window_Driver::makeCurrent()` when it calls `Fl_Wayland←Graphics_Driver::setBuffer()` that scales the graphics driver by this factor with `cairo←scale()`.

Overall, an FLTK object, say an `Fl_Window`, of size `WxH` FLTK units occupies `W * wld_scale * gui_scale x H * wld_scale * gui_scale` pixels on the display.

When an `Fl_Window` is to be show()ed, `Fl_Wayland_Window_Driver::makeWindow()` creates a `struct wl_surface` with `wl_compositor_create_surface()` and associates it calling `wl←surface_add_listener()` with a 2-member listener called `surface_listener` encharged of managing as follows the list of displays where this `wl_surface` will map. The `Fl_Window` possesses an initially empty linked list of displays accessible at member `outputs` of the window’s `wl_window` record. When the `Fl_Window`, or more exactly its associated `struct wl_surface` is mapped on a display, member `surfaceEnter()` of `surface_listener` runs. This function adds the display where the surface belongs to the end of the linked list of displays for this surface. When a surface is dragged or enlarged across the edge of a display in a multi-display system and expands on a second display, `surfaceEnter()` runs again, and this surface’s list of displays contains 2 items. When a surface leaves a display, member `surfaceLeave()` of `surface←listener` runs. It removes that display from the surface’s list of displays. Each time the first item of a surface’s list of displays changes, function `change_scale()` is called and applies that display’s `gui_scale` value to that surface calling `Fl_Window_Driver::screenNum(int)`. When a window is unmapped by function `Fl_Wayland_Window_Driver::hide()`, the surface’s list of displays is emptied.

### Fractional scaling

The KWin compositor, and gnome too if specially set, allow to use **fractional scaling** that can take intermediate values between 100% and 200%. Wayland implements this rendering all `wl_surface`s as if the scaling was at 200%, and downsizing them to the desired fractional scale value at the compositing stage. Seen from FLTK, everything runs as when `wld_scale = 2`.

These commands make gnome accept fractional scaling, and turn that off:

```bash
$ gsettings set org.gnome.mutter experimental-features "['scale-monitor-framebuffer']"
$ gsettings reset org.gnome.mutter experimental-features
```

### 25.1.12 Mouse and trackpad handling

FLTK receives information about mouse and pointer events via a ‘listener’ made up of 5 pointers to functions which Wayland calls when events listed in table below occur. These functions receive from Wayland enough information in their parameters to generate corresponding FLTK events, that is, calls to `Fl::handle(int event_type, Fl_Window *)`.

<table>
<thead>
<tr>
<th>listener function</th>
<th>called by Wayland when</th>
<th>resulting FLTK events</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer_enter</td>
<td>pointer enters a window</td>
<td>FL_ENTER</td>
</tr>
<tr>
<td>pointer_leave</td>
<td>pointer leaves a window</td>
<td>FL_LEAVE</td>
</tr>
<tr>
<td>pointer_motion</td>
<td>pointer moves inside a window</td>
<td>FL_MOVE</td>
</tr>
<tr>
<td>pointer_button</td>
<td>state of mouse buttons changes</td>
<td>FL_PUSH, FL_RELEASE</td>
</tr>
</tbody>
</table>

Generated by Doxygen
25.1 The Wayland backend for its developer

<table>
<thead>
<tr>
<th>listener function</th>
<th>called by Wayland when</th>
<th>resulting FLTK events</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer_axis</td>
<td>trackpad is moved vertically or horizontally</td>
<td>FL_MOUSEWHEEL</td>
</tr>
</tbody>
</table>

pointer_listener is installed by a call to function `wl_pointer_add_listener()` made by function `seat_capabilities()` which is itself another 'listener' made up of 2 function pointers

```
static struct wl_seat_listener seat_listener = {
    seat_capabilities,
    seat_name
};
```

installed by a call to function `wl_seat_add_listener()` made by function `registry_handle_global()` when it receives a "wl_seat" interface.

### 25.1.13 Wayland cursors

Wayland defines types `struct wl_cursor` and `struct wl_cursor_theme` to hold cursor-related data. FLTK uses function `init_cursors()` from file `Fl_Wayland_Screen_Driver.cxx` to obtain the 'cursor theme' name using function `libdecor_get_cursor_settings()` of library `libdecor`. Function `wl_cursor_theme_load()` then returns a pointer to an object of type `struct wl_cursor_theme` stored in member variable `cursor_theme` of the `Fl_Wayland_Screen_Driver::seat` record. Function `init_cursors()` is itself called by a 'listener' called `seat_capabilities()` installed when function `registry_handle_global()` receives a "wl_seat" interface, at program startup. It is also called when the value of the Wayland scaling factor changes: `output_done()` calls `try_update_cursor()` calls `init_cursors()`. Function `output_done()` belongs to a 'listener' installed when function `registry_handle_global()` receives a "wl_output" interface.

Each time `Fl_Window::cursor(Fl_Cursor)` runs, FLTK calls `Fl_Wayland_Window_Driver::set_cursor(Fl_Cursor)` which calls `wl_cursor_theme_get_cursor()` to set the current cursor shape to one of the standard shapes from the `Fl_Cursor` enumeration. This Wayland function selects a cursor shape based on the current `wl_cursor_theme` object and a cursor name and returns a pointer to a `struct wl_cursor`. Under the gnome desktop, cursor names are the files of directory `/usr/share/icons/XXXX/cursors/` where `XXXX` is the 'gnome cursor theme' (default= Adwaita). For example, what FLTK calls `FL_CURSOR_INSERT` corresponds to file `xterm` therein. The full correspondance between `Fl_Cursor` values and names of files therein is found in function `Fl_Wayland_Window_Driver::set_cursor(Fl_Cursor)`. FLTK stores in member variable `default_cursor` of the `Fl_Wayland_Screen_Driver::seat` record a pointer to the currently used `wl_cursor` object, and the current `Fl_Cursor` value in member `standard_cursor` of the `Fl_Wayland_Window_Driver` object.

Finally, function `do_set_cursor()` of file `Fl_Wayland_Screen_Driver.cxx` makes the system pointer use the current `wl_cursor` object to draw its shape on screen. That's done with a call to `wl_pointer_set_cursor()` and a few other functions.

### Custom cursor shapes

To support custom cursors, FLTK presently uses a non-public type, `struct cursor_image`, defined in file `Fl_Wayland_Window_Driver.cxx` as follows:

```
struct cursor_image {
    struct wl_cursor_image image;
    struct wl_cursor_theme *theme;
    struct wl_buffer *buffer;
    int offset;
};
```

This definition has been copied to the FLTK source code from file `wayland-cursor.c` of the Wayland project source code because it's not accessible via Wayland header files. It shows that a pointer to a `cursor_image` object can also be viewed as a pointer to the embedded `struct wl_cursor_image` object, this one being part of the public Wayland API. It also shows that a `struct cursor_image` object has an associated `struct wl_buffer` object used to contain the cursor's graphics.

Function `Fl_Wayland_Window_Driver::set_cursor(const Fl_RGB_Image *rgb, int hotx, int hoty)` gives FLTK support of custom cursor shapes. It calls `Fl_Wayland_Window_Driver::set_cursor_4args()` that creates a `cursor_image` object, allocates the corresponding `wl_buffer` by a call to `Fl_Wayland_Graphics_Driver::create_shm_buffer()` and draws the cursor shape into that buffer using the offscreen-drawing method of FLTK.

The public type `struct wl_cursor` is essentially an array of `wl_cursor_image` objects and a name:

```
struct wl_cursor {
    unsigned int image_count;
```
Function `Fl_Wayland_Window_Driver::set_cursor_4args()` also creates a struct `wl_cursor` object containing a single `wl_cursor_image`, which is in fact the cursor image. Finally, a struct `custom_cursor_` (see `wl_window`) is allocated and used to memorize the struct `wl_cursor` and the cursor's image and hotspot. A pointer to this struct `custom_cursor_` object is stored in member `custom_cursor_` of the window's `wl_window`.

Function `Fl_Wayland_Window_Driver::set_cursor_4args()` is also called when a window with a custom cursor is moved between distinct displays or when a display is rescaled to adapt the cursor size to the new display's scale factor.

Member function `Fl_Wayland_Window_Driver::delete_cursor_()` is used to delete any custom cursor shape. This occurs when a window associated to a custom cursor is unmapped and when such a window gets associated to a standard cursor or to a new custom cursor.

### 25.1.14 Text input

The "Mouse handling" section above mentioned function `seat_capabilities()` that Wayland calls when the app discovers its "seat". Presence of flag `WL_SEAT_CAPABILITY_KEYBOARD` in argument `capabilities` of this function indicates that a keyboard is available. In that case, a call to `wl_seat_get_keyboard()` returns a pointer stored in member `wl_keyboard` of the `Fl_Wayland_Screen_Driver::seat` object, and a call to `wl_keyboard_add_listener()` installs a 6-member listener of type struct `wl_keyboard_listener`.

These 6 FLTK-defined, callback functions are used as follows.

Function `wl_keyboard_keymap()` runs when the app starts and also if the keyboard layout is changed during run-time. It allows initialization of access to this keyboard. Noticeably, member `xkb_state` of type struct `xkb_state` of the current `Fl_Wayland_Screen_Driver::seat` record is adequately initialized.

Functions `wl_keyboard_enter()` and `wl_keyboard_leave()`, called when focus enters and leaves a surface, send `FL_FOCUS` and `FL_UNFOCUS` events to the `Fl_Window` object corresponding to this surface.

Function `wl_keyboard_key()` runs each time a keyboard key is pressed or released. Its argument, to which 8 must be added, provides the keycode via function `xkb_state_key_get_one_sym()` and then the corresponding text via function `xkb_state_key_get_utf8()` which is put in `Fl::e_text`. Then, a few calls to functions whose name begin with `xkb_compose_` are necessary to support dead and compose keys.

Finally a call to `Fl::handle()` sends an `FL_KEYDOWN` or `FL_KEYUP` event to the appropriate `Fl_Window`. Also, function `wl_keyboard_key()` uses global variable `Fl_Int_Vector key_vector` to record all currently pressed keys. This is the base of the implementation of `Fl_Wayland_Screen_Driver::event←key(int)`, `wl_keyboardModifiers()` runs when a modifier key (e.g., shift, control) is pressed or released. Calls to functions `xkb_state_update_mask()` and `xkb_state_mod_name_is_active()` allow FLTK to set `Fl::e_state` adequately.

Function `wl_keyboard_repeat_info()` does not run, for now, because this would require version 4 of the `wl_keyboard` object which is at version 3 in all tested Wayland compositors.

### 25.1.15 Support of text input methods

When the connected Wayland compositor supports text input methods, function `registry_handle←global()` gets called with its `interface.name` argument equal to `zwp_text_input_manager_v3←interface.name`. The following call to `wl_registry_bind()` returns a pointer to type `struct zwp_text_input_manager_v3` that is stored as member `text_input_base` of the `Fl_Wayland_Screen_Driver` object.

Later, when function `seat_capabilities()` runs, `text_input_base` is found not NULL, which triggers a call to function `zwp_text_input_manager_v3←get_text_input()` returning a value of type `struct zwp_text_input_v3` and stored as member `text_input` of the `Fl_Wayland_Screen_Driver::seat` object. Next, a call to `zwp_text_input_v3←add_listener()` associates this `text_input` with a 6-member listener of type struct `zwp_text_input_v3_listener`. These 6 FLTK-defined, callback functions are used as follows.

Functions `text_input_enter()` and `text_input_leave()` are called when text input enters or leaves a surface (which corresponds to an `Fl_Window`).

Functions `text_input_preedit_string()` and `text_input_commit_string()` are called when the text input method asks the client app to insert 'marked' text or regular text, respectively. Complex text input
often begins by inserting temporary text which is said to be 'marked' before replacing it with the text that will stay in the document. FLTK underlines marked text to distinguish it from regular text.

Functions text_input_delete_surrounding_text() and text_input_done() have no effect at present, without this preventing input methods that have been tested with FLTK to work satisfactorily.

It's necessary to inform text input methods of the current location of the insertion point in the active surface. This information allows them to map their auxiliary windows next to the insertion point, where they are expected to appear. The flow of information on this topic is as follows:

- The two FLTK widgets supporting text input, Fl_Input_ and Fl_Text_Display, transmit to FLTK the window coordinates of the bottom of the current insertion point and the line height each time they change calling function fl_set_spot().

- fl_set_spot() calls the platform override of virtual member function Fl_Screen_Driver::set_spot(). Under Wayland, this just calls Fl_Wayland_Screen_Driver::insertion_point_rectangle(int x, int y, int height) which calls zwp_text_input_v3_set_cursor_rectangle() to inform the text input method about the surface position and size of the insertion point and also memorizes this information in static member variables of class Fl_Wayland_Screen_Driver.

- Callback function text_input_enter() calls Fl_Wayland_Screen_Driver::insertion_point_location(int *x, int *y, int *height) which gives it the stored position information, and then calls zwp_text_input_v3_set_cursor_rectangle() to inform the text input method about the position of the insertion point.

### 25.1.16 Interface with libdecor

FLTK uses a library called libdecor to determine whether the Wayland compositor uses CSD or SSD mode, and also to draw window titlebars when in CSD mode (see libdecor). Libdecor is conceived to be present in a shared library linked to the Wayland client application which itself, and if the running Wayland compositor uses CSD mode, loads another shared library intended to draw titlebars in a way that best matches the Desktop. As of early 2023, two titlebar-drawing libdecor plugins are available:

- libdecor-gtk intended for the Gnome desktop;
- libdecor-cairo for other situations.

Because libdecor is not yet in Linux packages, or only in a preliminary state, FLTK bundles the most recent source code of libdecor and its plugins. This code is included in libfltk. FLTK uses libdecor-gtk when software package libgtk-3-dev is present in the build system, and libdecor-cairo otherwise.

Libdecor uses the Wayland protocol xdg decoration unstable v1 to request being decorated by a supporting compositor. If the running compositor supports SSD, libdecor doesn't draw window titlebars because the compositor does it. That is what happens with the KWin and Sway compositors. However, if environment variable LIBDECOR_FORCE_CSD is defined to value 1 when an FLTK app runs, libdecor instructs an SSD-capable compositor to refrain from decorating its windows and decorates windows itself.

CMake OPTION_USE_SYSTEM_LIBDECOR has been defined to allow FLTK in the future, when libdecor and libdecor-gtk will be part of Linux packages, to use these packages rather than the libdecor code bundled in FLTK. When this option is ON, preprocessor variable USE_SYSTEM_LIBDECOR is 1, and both libdecor and its plugin are loaded at run-time from shared libraries. This option is OFF by default.

Whatever the value of OPTION_USE_SYSTEM_LIBDECOR, FLTK and libdecor use environment variable LIBDECOR_PLUGIN_DIR as follows: if this variable is defined and points to the name of a directory, this directory is searched for a potential libdecor plugin in the form of a shared library; if one is found, FLTK and libdecor load it and use it.

The libdecor source code bundled in FLTK is identical to that of the libdecor repository. Nevertheless, FLTK uses this code with some minor changes. For example, except if USE_SYSTEM_LIBDECOR is 1, FLTK needs to modify function libdecor_new() charged of loading the plugin, to make it use the plugin code that is included in libfltk if none is found as a dynamic library. This is done as follows in file libdecor/build/fl_libdecor.c:

```c
#include "src/libdecor.c"
#undef libdecor_new
void libdecor_new() { // FLTK rewrite of this function
    ...}
```

FLTK compiles file fl_libdecor.c which includes libdecor.c to the effect that all of the libdecor code becomes part of libfltk except that function libdecor_new() is substituted by its FLTK rewrite, without file
libdecor.c being modified at all. This trick is also used to modify function libdecor_frame_set∗-
mimized() to bypass a bug in the Weston compositor before version 10. Similarly, FLTK compiles file fl←
libdecor-plugins.c which includes either libdecor-gtk.c or libdecor-cairo.c to the effect that the
desired plugin becomes part of libfltk.

To support function Fl_Widget_Surface::draw_decorated_window() that draws a mapped window
and its titlebar, FLTK needs to perform two operations: 1) identify what plugin is operating, and 2) call a function that is
specific of that plugin and that returns the pixels of the drawn titlebar.

FLTK performs operation 1) above using its function get_libdecor_plugin_description() of file fl←
libdecor-plugins.c that returns a human readable string describing the running plugin. Each plugin puts
its own string in member description of a record of type struct libdecor_plugin_description.

Although this function is public in header file libdecor-plugin.h, accessing the symbol defined by the plugin to
store a pointer to a value of this type is complicated for a reason and solved by a method detailed in a comment
before function get_libdecor_plugin_description().

Function get_libdecor_plugin_description() also determines whether the compositor uses CSD
or SSD mode. This information is stored in member decoration_mode of struct libdecor←
frame_private which is not part of the public libdecor API. For this reason, FLTK copies to fl←
libdecor-plugins.c the definition of this type present in libdecor.c.

Operation 2) above is done by FLTK-defined function fl_libdecor_titlebar_buffer() from file fl←
libdecor-plugins.c. This function calls get_libdecor_plugin_description() seen above
to get the running plugin's descriptive string. That is "GTK3 plugin" with libdecor-gtk. FLTK func
tion gtk_titlebar_buffer() is then called, and returns a pointer to the start of a byte buffer con
taining the titlebar graphics. That is, again, not possible with the public libdecor API. Therefore, FLTK
copies to fl_libdecor-plugins.c the definitions of several types given in libdecor-gtk.c or
libdecor-cairo.c such as type struct border_component.

25.1.17 Copy/Paste/Drag-n-Drop

FLTK follows the procedure that is very well described in item "Wayland clipboard and drag & drop" of the
Documentation resources. All corresponding source code is in file src/drivers/Wayland/fl_wayland←
_clipboard_dnd.cxx.

This part of the Fl_Wayland_Screen_Driver::seat record stores pointers to Wayland objects used for clipboard and
D-n-D operations:
struct wl_data_device_manager ∗data_device_manager;
struct wl_data_device ∗data_device;
struct wl_data_source ∗data_source;

FLTK can copy or paste plain UTF-8 text or image data to/from the clipboard. Images are copied to the clipboard as
image/bmp or image/png mime type. Images in image/bmp or image/png mime types from the clipboard can be pasted
to FLTK apps. Files dropped are received one pathname per line, with no ' \n' after the last pathname.

25.1.18 EGL as support for OpenGL

Wayland uses EGL™ to interface OpenGL with the underlying native platform window system. OpenGL-using FLTK
apps are therefore linked to libwayland-egl.so and libEGL.so in addition to libGL.so and libGLU.←
so.

EGL is initialized calling member function Fl_Wayland_Gl_Window_Driver::init() once, the first time
the Fl_Wayland_Gl_Window_Driver c’tor runs. That is done with calls to eglGetDisplay(), egl←
Initialize(), and eglBindAPI().

Member function Fl_Wayland_Gl_Window_Driver::find() calls eglChooseConfig() to filter the
set of GL configurations that match the Fl_Gl_Window's mode(), and puts in the returned Fl_Gl_Choice
object the first matching configuration. The filtering gets done with bits EGL_WINDOW_BIT, to support the creation
of window surfaces, and EGL_OPENGL_BIT, to support the creation of OpenGL contexts.

EGL needs 2 more objects created for each Fl_Gl_Window. They have types struct wl_eegl_window
and EGLSurface, and are created by member function Fl_Wayland_Gl_Window_Driver::make←
current_before() which runs at the beginning of Fl_Gl_Window::make_current(). The first ar-
guent of the call to wl_eegl_window_create() therein has type struct wl_surface ∗ and is what
connects EGL with the targeted Wayland window.

EGL creates with eglCreateContext() an object of type EGLContext via member function Fl←
Wayland_Gl_Window_Driver::create_gl_context() called by Fl_Gl_Window::make_current().
Types `EGLContext` and `GLContext` are 2 names for the same object. The call to `eglCreateContext()` is made asking for a GL context of version at least 2. This does not prevent from obtaining contexts of higher versions, namely above 3.2, which are compatible with version 2 (the so-called compatibility profile) under all tested Linux systems.

FLTK function `Fl_Gl_Window::make_current()` calls overridden function `Fl_Wayland_Gl_Window::set_gl_context()` which calls EGL function `eglMakeCurrent()` when the cached context changes.

FLTK calls function `Fl_Wayland_Gl_Window::swap_buffers()` each time it wants a GL context to be sent to the display. This function contains some pure GL code to emulate an overlay buffer to support `Fl_Gl_Window` objects overriding their `draw_overlay()` member function. Then, it calls function `eglSwapBuffers()`.

The overridden `Fl_Wayland_Gl_Window::resize()` function is implemented with calls to `wl_egl_window_get_attached_size()` and `wl_egl_window_resize()`.

Class `Fl_Wayland_Gl_Plugin` exists to allow libfltk to call functions from `libfltk_gl`, `libwayland-egl.so` or `libEGL.so` and without having libfltk force linking any FLTK app with these GL-related libraries. For example, `Fl_Wayland_Window_Driver::flush()` needs to call `Fl_Gl_Window::valid(0)`.

### 25.1.19 FLTK-defined, Wayland-specific types

**struct wld_window**

Defined in `Fl_Wayland_Window_Driver.H`. One such record is created for each shown()'n `Fl_Window` by `Fl_Wayland_Window_Driver::makeWindow()`. Function `fl_wl_xid(Fl_Window *)` returns a pointer to the `struct wld_window` of its argument.

```
struct wld_window {  
    Fl_Window *fl_win;  
    struct wl_list outputs; // linked list of displays where part or whole of window maps  
    struct wl_surface *wl_surface; // the window’s surface  
    struct fl_wld_buffer *buffer; // see fl_wld_buffer  
    struct xdg_surface *xdg_surface;  
    enum Fl_Wayland_Window_Driver::kind kind; // DECORATED or POPUP or SUBWINDOW or UNFRAMED  
    union {  
        struct libdecor_frame *frame; // for DECORATED windows  
        struct wl_subsurface *subsurface; // for SUBWINDOW windows  
        struct xdg_popup *xdg_popup; // for POPUP windows  
        struct xdg_toplevel *xdg_toplevel; // for UNFRAMED windows  
    };  
    struct custom_cursor {  
        struct wl_cursor *wl_cursor;  
        const Fl_RGB_Image *rgb;  
        int hotx, hoty;  
    } *custom_cursor; // non-null when using custom cursor  
    int configured_width; // used when negotiating window size with the compositor  
    int configured_height;  
    int floating_width; // helps restoring size after un-maximizing  
    int floating_height;  
    int state; // indicates whether window is fullscreen, maximized. Used otherwise for POPUPS  
};
```

**struct fl_wld_buffer**

Defined in `Fl_Wayland_Graphics_Driver.H`. One such record is created when and by:

- an `Fl_Window` is show()'n or resized, by `Fl_Wayland_Graphics_Driver::create_shm_buffer();`
- an `Fl_Image_Surface` object is created, by the `Fl_Wayland_Image_Surface_Driver` c'tor;
- a custom cursor shape is created or text is dragged, by `Fl_Wayland_Graphics_Driver::create_shm_buffer();`. 

Generated by Doxygen
struct fl_wld_buffer {
    struct wl_buffer *wl_buffer; // the Wayland buffer
    void *data; // address of the beginning of the Wayland buffer’s byte array
    size_t data_size; // of wl_buffer and draw_buffer
    int stride; // bytes per line
    int width;
    unsigned char *draw_buffer; // address of the beginning of the Cairo image surface’s byte array
    struct wl_callback *cb; // non-NULL while Wayland buffer is being committed
    bool draw_buffer_needs_commit; // true when draw_buffer has been modified but not yet committed
    cairo_t *cairo_; // used when drawing to the Cairo image surface
    struct wl_shm_pool *shm_pool; // pointer to wl_shm_pool from which this wl_buffer comes
    struct wl_list link; // links all buffers from the same wl_shm_pool
};

struct Fl_Wayland_Screen_Driver::output

Defined in Fl_Wayland_Screen_Driver.H. One such record is created for each display of the system by function registry_handle_global() when it receives a "wl_output" interface. These records are kept in a linked list of them all, and an identifier of this linked list is stored in member outputs of the unique Fl_Wayland_Screen_Driver object FLTK uses. Thus, Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
struct wl_list list_of_all_displays = scr_driver->outputs;
gives access, the Wayland way, to the linked list of displays in the system.

struct Fl_Wayland_Screen_Driver::output {
    // one record for each display
    uint32_t id; // an identifier of the display
    int x, y; // logical position of the top-left of display
    int width; // nber of horizontal pixels
    int height; // nber of vertical pixels
    float dpi; // at this point, always 96.
    struct wl_output *wl_output; // the Wayland object for this display
    int wld_scale; // Wayland scale factor
    float gui_scale; // FLTK scale factor
    bool done; // true means record members have been initialized
    struct wl_list link; // links these records together
};

It's possible to get the FLTK-defined record associated to a display from the Wayland-associated object for the same display, say struct wl_output *wl_output, by this call: (struct Fl_Wayland_Screen_Driver::output *)wl_output_get_user_data(wl_output).

struct Fl_Wayland_Screen_Driver::seat

Defined in file Fl_Wayland_Screen_Driver.H. One record is created by function registry_handle_global() when it receives a "wl_seat" or wl_data_device_manager_interface.name interface. A pointer to this struct is stored in member seat of the client's unique Fl_Wayland_Screen_Driver object.

struct Fl_Wayland_Screen_Driver::seat {
    struct wl_seat *wl_seat;
    struct wl_pointer *wl_pointer;
    struct wl_keyboard *wl_keyboard;
    uint32_t keyboard_enter_serial;
    struct wl_surface *keyboard_surface;
    struct wl_list pointer_outputs;
    struct wl_cursor_theme *cursor_theme;
    struct wl_cursor *default_cursor;
    struct wl_surface *cursor_surface;
    struct wl_surface *pointer_focus;
    int pointer_scale;
    uint32_t serial;
    uint32_t pointer_enter_serial;
    struct wl_data_device_manager *data_device_manager;
    struct wl_data_device *data_device;
    struct wl_data_source *data_source;
};
struct xkb_state *xkb_state;
struct xkb_context *xkb_context;
struct xkb_keymap *xkb_keymap;
struct xkb_compose_state *xkb_compose_state;
char *name;
struct zwp_text_input_v3 *text_input;
}

25.2 Developer info for bundled libs

This chapter details the procedure to update the libraries which are bundled inside FLTK.

25.2.1 Introduction

This file is mainly intended for FLTK developers and contains information about the current versions of all bundled libraries and about how to upgrade these bundled libraries.

Starting with FLTK 1.4.0 the bundled libraries jpeg, png, and zlib use "symbol prefixing" with the prefix "fltk_" for all external symbols to distinguish the bundled libraries from existing system libraries and to avoid runtime errors.

User code compiled correctly with the header files provided by the bundled image libraries need not be changed.

The nanosvg library is not affected.

25.2.2 Current status

Current versions of bundled libraries (as of July 24, 2023):

<table>
<thead>
<tr>
<th>Library</th>
<th>Version/git commit</th>
<th>Release date</th>
<th>FLTK Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg</td>
<td>jpeg-9e</td>
<td>2022-01-16</td>
<td>1.4.0</td>
</tr>
<tr>
<td>nanosvg</td>
<td>abcd277ea4</td>
<td>2022-12-22</td>
<td>1.4.0</td>
</tr>
<tr>
<td>png</td>
<td>libpng-1.6.40</td>
<td>2023-06-21</td>
<td>1.4.0</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib-1.2.13</td>
<td>2022-10-13</td>
<td>1.4.0</td>
</tr>
<tr>
<td>libdecor</td>
<td>f43652c7</td>
<td>2023-07-15</td>
<td>1.4.0</td>
</tr>
</tbody>
</table>

Previous versions of bundled libraries (FLTK 1.3.x):

<table>
<thead>
<tr>
<th>Library</th>
<th>Version/git commit</th>
<th>Release date</th>
<th>FLTK Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg</td>
<td>jpeg-9d</td>
<td>2020-01-12</td>
<td>1.3.6 - 1.3.8</td>
</tr>
<tr>
<td>png</td>
<td>libpng-1.6.37</td>
<td>2019-06-14</td>
<td>1.3.6 - 1.3.8</td>
</tr>
<tr>
<td>zlib</td>
<td>zlib-1.2.11</td>
<td>2017-01-15</td>
<td>1.3.6 - 1.3.8</td>
</tr>
</tbody>
</table>

See also git tag 'fltk_yyyy-mm-dd' where yyyy-mm-dd == "Release date" and file nanosvg/README.txt.

---

25.1.20 Documentation resources

<table>
<thead>
<tr>
<th>The Wayland book</th>
<th>Extensive introduction to Wayland programming written by the author of the sway compositor, unfortunately unachieved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wayland Explorer</td>
<td>Documentation of all Wayland protocols, both stable and unstable. A language-independent syntax is used which makes function names usable from C or C++ not always obvious. Some useful functions seem undocumented here for an unclear reason.</td>
</tr>
<tr>
<td>Wayland clipboard and drag &amp; drop</td>
<td>Detailed explanation of how clipboard and drag-and-drop work under Wayland.</td>
</tr>
<tr>
<td>Wayland and input methods</td>
<td>Blog article introducing to the issue of text input methods under Wayland.</td>
</tr>
<tr>
<td>Input Method Hub</td>
<td>Entry page for input method support giving newcomers a first understanding of what input methods are and how they are implemented in Wayland.</td>
</tr>
</tbody>
</table>
**General information:**

FLTK does not include the entire library distributions. We only provide the source files necessary to build the FLTK library and some README and/or CHANGELOG files. There are no test programs or other contributed files.

We use our own build files, hence a few files MUST NOT be upgraded when the library source files are upgraded. We strive to keep changes to the library source files as small as possible. Patching library code to work with FLTK should be a rare exception. Symbol prefixing with prefix 'ftlk_' is one such exception to the rule.

If patches are necessary all changes in the library files should be marked with "FLTK" in a comment so a developer who upgrades the library later is aware of changes in the source code for FLTK. Look for ‘FLTK’ and/or ‘ftlk_’ to find the differences.

Additional comments should be added to show the rationale, i.e. why a particular change was necessary. If applicable, add a reference to a Software Trouble Report, GitHub Issue or Pull Request (PR) like “STR 3456”, “Issue #123”, or “PR #234”.

**25.2.3 How to update the bundled libraries**

It is generally advisable to use a graphical merge program. I'm using ‘meld’ under Linux, but YMMV.

Do not add any source files unless they are required to build the library.

Some config header files may be pre-generated in the FLTK sources. These header files should be left untouched, but it may be necessary to update these files if new items were added to the new library version. In this case the new header should be pre-generated on a Linux system with default options unless otherwise mentioned below for a specific library. Currently there are no known exceptions.

**Merging source files:**

Please check if some source and header files contain “FLTK” comments and/or ‘ftlk_’ symbol prefixing to be aware of necessary merges. It is also good to download the distribution tar ball or Git source files of the previous version and to run a (graphical) diff or merge tool on the previous version and the bundled version of FLTK to see the "previous" differences.

Files that were not patched in previous versions should be copied to the new version w/o changes. Files that had FLTK specific patches must be merged manually. FLTK patches should be verified (if still necessary) and should be kept in the new source files.

Source and header files that have been added in the new library version should be added in FLTK as well if they are necessary to build the library. A simple "trial and error" should be sufficient to find files that need to be added. Added files must be added to FLTK’s build files as well, usually to both ‘Makefile’ and ‘CMakeLists.txt’ to be used in configure/make and in CMake based builds, respectively.

**Upgrade order:**

There is only one dependency between all bundled libraries: libpng depends on zlib. Hence zlib should be upgraded first, then all other libs can be upgraded in arbitrary order.

**Tests after merge:**

Tests should be done on as many platforms as possible, both with autotools (configure/make) and CMake. Windows (Visual Studio) and macOS (Xcode) builds need CMake to generate the IDE files.

**Upgrade notes for specific libraries:**

The following chapters contain informations about specific files and how they are upgraded. Since the changes in all bundled libraries are not known in advance this information may change in the future. Please verify that no other changes are necessary.

**25.2.4 zlib:**

Website: [https://zlib.net/](https://zlib.net/)

Download: See website and follow links.

Repository: git clone [https://github.com/madler/zlib.git](https://github.com/madler/zlib.git)

zlib should be upgraded first because libpng depends on zlib.
25.2 Developer info for bundled libs

Download the latest zlib sources, 'cd' to /path-to/zlib and run

```
$ ./configure --prefix
```

This creates the header file `zconf.h` with definitions to enable the standard 'z_' symbol prefix.

Unfortunately zlib requires patching some source and header files to convert this 'z_' prefix to 'fltk_z_' to be more specific. As of this writing (Nov. 2021) three files need symbol prefix patches:

- gzread.c
- zconf.h
- zlib.h

You may want to compare these files and/or the previous version to find out which changes are required. The general rule is to change all occurrences of 'z_' to 'fltk_z_' but there are exceptions. The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- Makefile: Same as CMakeLists.txt.
- gzread.c: Merge changes (see above, manual merge recommended).
- zconf.h: Merge changes (see above, manual merge recommended).
- zlib.h: Merge changes (see above, manual merge recommended).
- makedepend: Keep this file.

Run 'make depend' in the zlib folder on a Linux system after the upgrade to update this file.

### 25.2.5 png:

**Website:** [http://libpng.org/pub/png/libpng.html](http://libpng.org/pub/png/libpng.html)

Download: See website and follow links.

Repository: git clone [https://git.code.sf.net/p/libpng/code](https://git.code.sf.net/p/libpng/code)

Libpng should be upgraded after zlib because it depends on zlib.

Download the latest libpng sources, 'cd' to /path-to/libpng and run

```
$ ./configure --with-libpng-prefix=fltk_
$ make
```

This creates the header files `pnglibconf.h` and `pngprefix.h` with the 'fltk_' symbol prefix.

The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- Makefile: Same as CMakeLists.txt.
- pnglibconf.h: Generate on a Linux system and merge (see above).
- pngprefix.h: Generate on a Linux system and merge (see above).
- makedepend: Keep this file.
- png.c: Keep a change labelled with "FLTK"
- pngerror.c: Keep two changes labelled with "FLTK"

Run 'make depend' in the png folder on a Linux system after the upgrade to update this file.

Generated by Doxygen
25.2.6 jpeg:

Website: https://ijg.org/

Download: See website and follow links.

Repository: N/A

Download the latest jpeg-xy sources on a Linux (or Unix) system, `cd` to `/path-to/jpeg-xy` and run:

```
$ ./configure
$ make [-jN]
```

This builds the library and should create the static library file `.libs/libjpeg.a`.

Execute the following command to extract the `libjpeg` symbol names used to build the `prefixed` libfltk_jpeg library:

```
$ nm --extern-only --defined-only .libs/libjpeg.a | awk '{print $3}' \
  | sed '/^$/d' | sort -u | awk '{print "#define $1 fltk_"$1}'} \
> fltk_jpeg_prefix.h
```

This creates the header file `fltk_jpeg_prefix.h` with the `# define` statements using the `fltk_` symbol prefix.

The following files need special handling:

- **CMakeLists.txt**: Keep FLTK version, update manually if necessary.
- **Makefile**: Same as CMakeLists.txt.
- **fltk_jpeg_prefix.h**: Generate on a Linux system and merge (see above).
- **jconfig.h**: keep changes flagged with
  ```
  /* FLTK */
  ```
- **makedepend**: Keep this file.

Run `make depend` in the jpeg folder on a Linux system after the upgrade to update this file.

25.2.7 nanosvg:

Website: https://github.com/memononen/nanosvg

Download: See website and follow links.

Repository: git clone https://github.com/memononen/nanosvg.git

FLTK Fork: git clone https://github.com/fltk/nanosvg.git

FLTK has its own GitHub fork of the original repository (see above).

The intention is to update this fork from time to time so the FLTK specific patches are up-to-date with the original library. Hopefully the FLTK patches will be accepted upstream at some time in the future so we no longer need our own patches. AlbrechtS, 04 Feb 2018.

Update (Feb 22, 2021): The upstream library is officially no longer maintained (see README.md) although updates appear from time to time.

Use this fork (branch 'fltk') to get the nanosvg library with FLTK specific patches:

```
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git checkout fltk
$ cd src
$ cp nanosvg.h nanosvgrast.h /path/to/fltk-1.4/nanosvg/
```

This library does not have its own build files since it is a header-only library. The headers are included in FLTK where necessary.

The following files need special handling:

- **nanosvg.h**: Merge or download from FLTK's fork (see above).
- **nanosvgrast.h**: Merge or download from FLTK's fork (see above).

Maintaining branch 'fltk' in FLTK's fork of nanosvg (fltk/nanosvg):

Only maintainers with write access on fltk/nanosvg can do this. Others can fork our fltk/nanosvg fork in their own GitHub account and either open a PR on fltk/nanosvg or tell us about their changes in fltk.development.

Use something similar to the following commands to update FLTK's fork of nanosvg to the latest version. Commands are only examples, you may need to change more or less, depending on the outstanding updates.

**Step 1**: Clone the fltk/nanosvg fork, set the remote 'upstream', and update the 'master' branch:

```
$ cd /to/your/dev/dir
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git remote add upstream https://github.com/memononen/nanosvg
$ git checkout master
$ git pull upstream master
```
Note: the 'master' branch must never be changed, i.e. it must always be the same as 'upstream/master'. Never commit your own (FLTK specific) changes to branch 'master'.

Step 2: rebase branch 'fltk' on the new master (upstream/master), fix potential conflicts, and tag the new branch.
It is important to keep the individual FLTK specific patches intact (one commit per patch) because this will preserve the history and the committer and make it easier to skip single patches when they are accepted upstream.

```bash
$ git checkout fltk
$ git rebase upstream/master
```

At this point you may need to fix conflicts! Do whatever is necessary to update the branch 'fltk'.

Now 'git tag' the 'fltk' branch for later reference.

Hint: use 'git show <any-older-tag-name>' to see its contents. I like to write a summary of commits in the tag comment.

```bash
$ git tag -a fltk_yyyy-mm-dd fltk
```

Replace 'yyyy-mm-dd' with the current date and add a comment when asked for it (your editor will open an empty file).

Step 3: at this point it is recommended to copy the changed header files to your working copy of the FLTK library and test the changes. If anything is wrong, go back, fix the bugs and change the git tag (delete and create a new one).

Step 4: push the new branch 'fltk' and the tag to the fltk/nanosvg repository:

```bash
$ git push -f origin fltk
$ git push origin fltk_yyyy-mm-dd
```

Step 5: copy the changed files to your working copy of the FLTK repository (if not done already), update this file accordingly, and commit/push the update to the fltk/fltk repository.

### 25.2.8 libdecor:

Website: [https://gitlab.freedesktop.org/libdecor/libdecor](https://gitlab.freedesktop.org/libdecor/libdecor)

Download: See website and follow links.

Repository: git clone [https://gitlab.freedesktop.org/libdecor/libdecor.git](https://gitlab.freedesktop.org/libdecor/libdecor.git)

libdecor is used by the Wayland/X11 hybrid platform to draw window titlebars when FLTK apps run as Wayland clients and the running Wayland compositor uses client-side decoration. In the future, when libdecor will have made its way into Linux packages, FLTK will use the system version of libdecor. libdecor will remain as an FLTK bundle to support Linux configurations where the libdecor package is not available or not installed.

FLTK uses libdecor source files without any modification. This part of the libdecor source tree is copied to directory libdecor/ of the FLTK source tree:

```
LICENSE
README.md
src/ ... and files below except meson.build files
```

Furthermore, directory libdecor/build/ of the FLTK source tree does not originate from the libdecor source tree but contains 3 FLTK-created files. File build/Makefile may need changes if a libdecor update adds or renames source files.

### 25.3 Developer Information

This chapter describes FLTK development and documentation.

#### Example

```c
/** \file
   Fl_Clock, Fl_Clock_Output widgets. */

/**
 * Fl_Clock_Output
 * \brief This widget can be used to display a program-supplied time.
 * The time shown on the clock is not updated. To display the current time, use Fl_Clock instead.
 *
 * \image html clock.png
 * \image latex clock.png /* width=10cm
 * \image html round_clock.png
 * \image latex clock.png /* width=10cm
```
Returns the displayed time.
Returns the time in seconds since the UNIX epoch (January 1, 1970).

```cpp
ulong value() const {return value_;
```

Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).

```cpp
void Fl_Clock_Output::value(ulong v) {
[...]
```

Create an Fl_Clock widget using the given position, size, and label string.
The default boxtype is \c FL_NO_BOX.

```cpp
Fl_Clock::Fl_Clock(int X, int Y, int W, int H, const char *L)
```

Create an Fl_Clock widget using the given boxtype, position, size, and
label string.

```cpp
Fl_Clock::Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L)
```

Note
From Duncan: (will be removed later, just for now as a reminder)
I've just added comments for the fl_color_chooser() functions, and in order to keep them and the general Function
Reference information for them together, I created a new doxygen group, and used \ingroup in the three comment
blocks. This creates a new Modules page (which may not be what we want) with links to it from the File Members
and Fl_Color_Chooser.H pages. It needs a bit more experimentation on my part unless someone already knows
how this should be handled. (Maybe we can add it to a functions.dox file that defines a functions group and do that
for all of the function documentation?)

**UPDATE:** the trick is not to create duplicate entries in a new group, but to move the function information into the
doxygen comments for the class, and use the navigation links provided. Simply using \relatesalso as the first
doxygen command in the function's comment puts it in the appropriate place. There is no need to have \defgroup
and \ingroup as well, and indeed they don't work. So, to summarize:

**Gizmo.H**
```cpp
/** 
 * class Gizmo
 * A gizmo that does everything
 */

```cpp
extern int popup_gizmo(...);
```

**Gizmo.cxx**
```cpp
/** 
 * relatesalso Gizmo
 * Pops up a gizmo dialog with a Gizmo in it
 */

```cpp
int popup_gizmo(...);
```
Comments Within Doxygen Comment Blocks

You can use HTML comment statements to embed comments in doxygen comment blocks. These comments will not be visible in the generated document.

The following text is a developer comment.

\!
\!--  ***  This *** is *** invisible *** -->

This will be visible again.

will be shown as:

The following text is a developer comment.

\!
\!--  ***  This *** is *** invisible *** -->

This will be visible again.

Different Headlines

You can use HTML tags \(<H1> \ldots \ <H4>\) for headlines with different sizes. As of doxygen 1.8.x there must not be more than three spaces at the beginning of the line for this to work. Currently (doxygen 1.8.6) there seems to be no difference in the font sizes of \(<H3>\) and \(<H4>\) in the pdf output, whereas the html output uses different font sizes.

\(<H1>Headline in big text \{H1\}\)</H1>
\(<H2>Headline in big text \{H2\}\)</H2>
\(<H3>Headline in big text \{H3\}\)</H3>
\(<H4>Headline in big text \{H4\}\)</H4>

25.3.1 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
Alternatively you can use UTF-8 encoding within Doxygen comments.

25.3.2 Document Structure

- \page creates a named page
- \section creates a named section within that page
- \subsection creates a named subsection within the current section
- \subsubsection creates a named subsubsection within the current subsection

All these statements take a "name" as their first argument, and a title as their second argument. The title can contain spaces.

The page, section, and subsection titles are formatted in blue color and a size like "\(<H1>\)\", "\(<H2>\)\", and "\(<H3>\)\", and "\(<H4>\)", respectively.

By FLTK documentation convention, a file like this one with a doxygen documentation chapter has the name "\<chapter>.dox". The \page statement at the top of the page is "\page \<chapter> This is the title". Sections within a documentation page must be called "\<chapter>_<section>\", where "\<chapter>" is the name part of the file, and "\<section>" is a unique section name within the page that can be referenced in links. The same for subsections and subsubsections.

Generated by Doxygen
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

### 25.3.3 Creating Links

Links to other documents and external links can be embedded with

- doxygen \ref links to other doxygen \page, \section, \subsection and \anchor locations
- HTML links without markup - doxygen creates "http://..." links automatically
- standard, non-Doxygen, HTML links
  
  - see chapter \ref unicode creates a link to the named chapter unicode that has been created with a \page statement.
  
  - For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
  
  - see <a href="https://www.fltk.org/">FLTK Library</a> creates a standard HTML link

appears as:

- see chapter Unicode and UTF-8 Support creates a link to the named chapter unicode that has been created with a \page statement.
- For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
- see FLTK Library creates a standard HTML link

### 25.3.4 Paragraph Layout

There is no real need to use HTML \texttt{<P>} and \texttt{</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, \texttt{commands} in the text it knows that it has reached the start or end of a paragraph. Doxygen also offers the \par command for special paragraph handling. It can be used to provide a paragraph title and also to indent a paragraph. Unfortunately \par won't do what you expect if you want to have doxygen links and sometimes html tags don't work either.

\par Normal Paragraph with title

This paragraph will have a title, but because there is a blank line between the \par and the text, it will have the normal layout.

\par Indented Paragraph with title

This paragraph will also have a title, but because there is no blank line between the \par and the text, it will be indented.

\par

It is also possible to have an indented paragraph without title. This is how you indent subsequent paragraphs.

\par No link to Fl_Widget::draw()

Note that the paragraph title is treated as plain text. Doxygen type links will not work. HTML characters and tags may or may not work.

Fl_Widget::draw() links and "html" tags work

\par

Use a single line ending with <br> for complicated paragraph titles.

The above code produces the following paragraphs:
Normal Paragraph with title

This paragraph will have a title, but because there is a blank line between the `par` and the text, it will have the normal layout.

Indented Paragraph with title

This paragraph will also have a title, but because there is no blank line between the `par` and the text, it will be indented.

It is also possible to have an indented paragraph without title. This is how you indent subsequent paragraphs.

No link to Fl_Widget::draw()

Note that the paragraph title is treated as plain text. Doxygen type links will not work. HTML characters and tags may or may not work.

Fl_Widget::draw() links and "html" tags work

Use a single line ending with `<br>` for complicated paragraph titles.

### 25.3.5 Navigation Elements

Each introduction (tutorial) page ends with navigation elements. These elements must only be included in the html documentation, therefore they must be separated with `htmlonly` and `endhtmlonly`.

The following code gives the navigation bar at the bottom of this page:

```html
\htmlonly
<br>
<table summary="navigation bar" width="100%" border="0">
<tr>
  <td width="45%" align="LEFT">
    <a class="el" href="migration_1_4.html">
      [Prev]
      Migrating Code from FLTK 1.3 to 1.4
    </a>
  </td>
  <td width="10%" align="CENTER">
    <a class="el" href="index.html">
      [Index]
    </a>
  </td>
  <td width="45%" align="RIGHT">
    <a class="el" href="license.html">
      Software License
      [Next]
    </a>
  </td>
</tr>
\endhtmlonly
```
Chapter 26

Todo List

Page Adding and Extending Widgets

Clarify `Fl_Window::damage(uchar)` handling - seems confused/wrong? ORing value doesn't match setting behavior in `Fl_Widget.H`!

Member `Fl::now()`

`Fl::system_driver()-gt-gettime()` was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses `gettimeofday()` with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: winmm.lib (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

Member `Fl_Browser_::scrollbar_width()` const

This method should eventually be removed in 1.4+

Member `Fl_Browser_::scrollbar_width(int width)`

This method should eventually be removed in 1.4+

Class `Fl_Chart`

Refactor `Fl_Chart::type()` information.

Member `Fl_File_Input::errorcolor()` const

Remove `Fl_File_Input::errorcolor()` in FLTK 1.5.0 or higher.

Member `Fl_File_Input::errorcolor(Fl_Color c)`

Remove `Fl_File_Input::errorcolor(Fl_Color)` in FLTK 1.5.0 or higher.

Member `fl_filename_list(const char *d, struct dirent ***l, 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_child(int)` in more FLTK subclasses. This is not yet complete.

Member `Fl_height(int font, int size)`

In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

Member `Fl_Help_View::find(const char *s, int p=0)`

complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and do not (yet) match correctly ("<" matches "<" but "€" doesn't match "€", and "ü" doesn't match "ü")

Member `Fl_Input_::handle_mouse(int e, 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_Int_Vector`

• Add other std::vector methods like erase(), etc.
  • Make memory blocking size flexible, and add related methods like capacity(), reserve(), shrink_to_fit(), etc.
• Add non-std methods that are nevertheless needed, e.g. insert(index,val), delete(index), delete(start, end), swap(a_idx,b_idx)

Member FL_Int_Vector::back () const
  Internals should maybe assert(size_ != 0)

Member FL_Int_Vector::pop_back ()
  Internals should maybe assert(size_ != 0)

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/newsgroups.php?gfltk.coredev+v:10086 and results of STR#2344

Member FL_Shortcut
  Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on all programs being coded with symbolic names and not hard coded bit values.

Member FL_Text_Display::extend_range_for_styles (int *start, int *end)
  Unicode?

Member FL_Text_Display::handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const
  we need to handle hidden hyphens and tabs here!
  we handle all styles and selections
  we must provide code to get pixel positions of the middle of a character as well

Member FL_Text_Display::overstrike (const char *)
  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::shortcut (int s)
  FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member FL_Text_Display::shortcut () const
  FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member FL_Text_Display::wrapped_column (int row, int column) const
  What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)
  Unicode?

Member FL_Text_Display::wrapped_row (int row) const
  What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)
Member `Fl_Tiled_Image::Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)`

Fix `Fl_Tiled_Image` as background image for widgets and windows and fix the implementation of `Fl::scheme(const char *)`.

Member `Fl_Tree::handle (int e) FL_OVERRIDE`

add `Fl_Widget_Tracker` (see `Fl_Browser__.cxx::handle()`)

Member `Fl_Tree::is_scrollbar (Fl_Widget *w)`

should be const

Member `Fl_Tree::show_self ()`

should be const

Member `Fl_Window::show () FL_OVERRIDE`

Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

Page FLTK Basics

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

Page Handling Events

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

Page Unicode and UTF-8 Support

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

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 27

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_::position () const**
"in 1.4.0 - use vposition() instead"

**Member Fl_Browser_::position (int pos)**
"in 1.4.0 - use vposition(pos) instead"

**Member Fl_Browser_::scrollbar_width () const**
Use scrollbar_size() instead.

**Member Fl_Browser_::scrollbar_width (int width)**
Use scrollbar_size() instead.

**Member fl_clip (int x, int y, int w, int h)**
Please use fl_push_clip(int x, int y, int w, int h) instead. fl_clip(int, int, int, int) will be removed in FLTK 1.5.

**Member Fl_File_Input::errorcolor () const**
Will be removed in FLTK 1.5.0 or higher.

**Member Fl_File_Input::errorcolor (Fl_Color c)**
Will be removed in FLTK 1.5.0 or higher.

**Member fl_find (Window xid)**
Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wid_window*), fl_win32_find(HWND) or fl_mac_find(FLWindow*) with FLTK 1.4.0 and above.

**Member Fl_GIF_Image::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. You should use W->take_focus() instead.

**Member Fl_Group::sizes ()**
Deprecates 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_Widget *w)**

   Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Member **Fl::Image::label (Fl_Menu_Item *m)**

   Please use `Fl_Menu_Item::image()` 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::Input_::position (int p)**

   "in 1.4.0 - use insert_position(p) instead"

Member **Fl::Input_::position (int p, int m)**

   "in 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

Member **Fl::Input_::position () const**

   "in 1.4.0 - use insert_position() instead"

Member **Fl::Menu_Item::check ()**

   Please use `Fl::Menu_Item::set()` instead. This method will be removed in FLTK 1.5.0 or later.

Member **Fl::Menu_Item::checked () const**

   Please use `Fl::Menu_Item::value()` instead. This method will be removed in FLTK 1.5.0 or later.

Member **Fl::Menu_Item::uncheck ()**

   Please use `Fl::Menu_Item::clear()` instead. This method will be removed in FLTK 1.5.0 or later.

Member **fl_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::Text_Selection::position (int *startpos, int *endpos) const**

   "in 1.4.0 - use selected(startpos, endpos) instead"

Member **Fl::Tile::position (int oldx, int oldy, int newx, int newy)**

   "in 1.4.0 - use move_intersection(p) instead"

Member **Fl::Tree::first_visible ()**

   in 1.3.3 ABI – use `first_visible_item()` instead.

Member **Fl::Tree::item_clicked (Fl_Tree_Item *val)**

   in 1.3.3 ABI – use `callback_item()` instead.

Member **Fl::Tree::item_clicked ()**

   in 1.3.3 ABI – use `callback_item()` instead.

Member **Fl::Tree::last_visible ()**

   in 1.3.3 – use `last_visible_item()` instead.
Member Fl_Tree_Item::Fl_Tree_Item (const Fl_Tree_Prefs &prefs)  
in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree+) for proper horizontal scrollbar behavior.

Member Fl_Tree_Item::next_displayed (Fl_Tree_Prefs &prefs)  
in 1.3.3 for confusing name, use next_visible() instead

Member Fl_Tree_Item::prev_displayed (Fl_Tree_Prefs &prefs)  
in 1.3.3 for confusing name, use prev_visible()

Member FL_VERSION  
This double version number is retained for compatibility with existing program code. New code should use int FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

Member Fl_Widget::color2 (unsigned a)  
Use selection_color(unsigned) instead.

Member Fl_Widget::color2 () const  
Use selection_color() instead.

Member Fl_Window::free_position ()  
please use force_position(0) instead

Member Fl_Window::icon () const  
in 1.3.3

Member Fl_Window::icon (const void *ic)  
in 1.3.3 in favor of platform-independent methods Fl_Window::icon(const Fl_RGB_Image *icon) and Fl_Window::icons(const Fl_RGB_Image *icons[], int count).

Page Operating System Issues  
Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window *) with versions 1.4 and above.

Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.
## Module Index

### 28.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callback Function Typedefs</td>
<td>271</td>
</tr>
<tr>
<td>Windows handling functions</td>
<td>272</td>
</tr>
<tr>
<td>Events handling functions</td>
<td>274</td>
</tr>
<tr>
<td>Selection &amp; Clipboard functions</td>
<td>289</td>
</tr>
<tr>
<td>Screen functions</td>
<td>293</td>
</tr>
<tr>
<td>Color &amp; Font functions</td>
<td>299</td>
</tr>
<tr>
<td>Drawing functions</td>
<td>314</td>
</tr>
<tr>
<td>Multithreading support functions</td>
<td>346</td>
</tr>
<tr>
<td>Safe widget deletion support functions</td>
<td>348</td>
</tr>
<tr>
<td>Cairo Support Functions and Classes</td>
<td>351</td>
</tr>
<tr>
<td>Unicode and UTF-8 functions</td>
<td>353</td>
</tr>
<tr>
<td>Mac OS X-specific symbols</td>
<td>371</td>
</tr>
<tr>
<td>Common Dialog Classes and Functions</td>
<td>372</td>
</tr>
<tr>
<td>File names and URI utility functions</td>
<td>388</td>
</tr>
<tr>
<td>Fl_string</td>
<td>395</td>
</tr>
</tbody>
</table>
Chapter 29
Hierarchical Index

29.1 Class Hierarchy

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

- Fl_GIF_Image::GIF_FRAME::CPAL ................................................. 397
- Fl_Preferences::Entry ............................................................... 397
- Fl .......................................................... 397
- Fl_Cairo_State ................................................................. 493
- FL_CHART_ENTRY ............................................................... 503
- Fl_End ................................................................. 539
- Fl_File_Chooser ............................................................... 545
- Fl_File_Icon .............................................................. 552
- Fl_Gl_Choice .............................................................. 581
- Fl_Glut_Bitmap_Font ....................................................... 590
- Fl_Glut_StrokeChar ......................................................... 590
- Fl_Glut_StrokeFont .......................................................... 591
- Fl_Glut_StrokeStrip .......................................................... 591
- Fl_Glut_StrokeVertex .......................................................... 591
- Fl_Help_Block ............................................................ 607
- Fl_Help_Dialog ........................................................... 607
- Fl_Help_Font_Stack ......................................................... 609
- Fl_Help_Font_Style .......................................................... 610
- Fl_Help_Link ............................................................... 610
- Fl_Help_Target ............................................................ 611
- Fl_Image ............................................................. 622
  - Fl_Bitmap ........................................................... 442
    - Fl_XBM_Image ......................................................... 1160
  - Fl_Pixmap ........................................................... 733
    - Fl_GIF_Image ......................................................... 578
    - Fl_Anim_GIF_Image .................................................. 429
    - Fl_XPM_Image ........................................................ 1161
  - Fl_RGB_Image .......................................................... 789
  - Fl_BMP_Image .......................................................... 446
    - Fl_ICO_Image .......................................................... 621
  - Fl_JPEG_Image .......................................................... 668
  - Fl_PNG_Image .......................................................... 739
  - Fl_PNM_Image .......................................................... 740
  - Fl_SVG_Image .......................................................... 865
  - Fl_Shared_Image .......................................................... 816
  - Fl_Tiled_Image .......................................................... 982
  - Fl_Image_Reader .......................................................... 632
  - Fl_Int_Vector ............................................................ 666
  - Fl_Label .............................................................. 671
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Mac_App_Menu</td>
<td>675</td>
</tr>
<tr>
<td>Fl_Menu_Item</td>
<td>697</td>
</tr>
<tr>
<td>Fl_Multi_Label</td>
<td>711</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser</td>
<td>715</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser_FLTK_Driver</td>
<td></td>
</tr>
<tr>
<td>Fl_Kdialog_Native_File_Chooser_Driver</td>
<td>.671</td>
</tr>
<tr>
<td>Fl_Zenity_Native_File_Chooser_Driver</td>
<td>1162</td>
</tr>
<tr>
<td>Fl_Plugin</td>
<td>737</td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
<td>533</td>
</tr>
<tr>
<td>Fl_Preferences</td>
<td>751</td>
</tr>
<tr>
<td>Fl_Plugin_Manager</td>
<td>738</td>
</tr>
<tr>
<td>Fl_Rect</td>
<td>783</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_LRTB</td>
<td>785</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Region_XYWH</td>
<td>786</td>
</tr>
<tr>
<td>Fl_Scheme</td>
<td>799</td>
</tr>
<tr>
<td>Fl_Scroll::Fl_Scrollbar_Data</td>
<td>813</td>
</tr>
<tr>
<td>Fl_String</td>
<td>848</td>
</tr>
<tr>
<td>Fl_Surface_Device</td>
<td>860</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td></td>
</tr>
<tr>
<td>Fl_Widget_Surface</td>
<td>1132</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>526</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>540</td>
</tr>
<tr>
<td>Fl_Image_Surface</td>
<td>633</td>
</tr>
<tr>
<td>Fl_Paged_Device</td>
<td>728</td>
</tr>
<tr>
<td>Fl_PostScript_File_Device</td>
<td>744</td>
</tr>
<tr>
<td>Fl_Printer</td>
<td>772</td>
</tr>
<tr>
<td>Fl_SVG_File_Surface</td>
<td>862</td>
</tr>
<tr>
<td>Fl_Text_Buffer</td>
<td>909</td>
</tr>
<tr>
<td>Fl_Text_Selection</td>
<td>975</td>
</tr>
<tr>
<td>Fl_Timeout</td>
<td>985</td>
</tr>
<tr>
<td>Fl_Tooltip</td>
<td>996</td>
</tr>
<tr>
<td>Fl_Tree_Item</td>
<td>1043</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array</td>
<td>1065</td>
</tr>
<tr>
<td>Fl_Tree_Prefs</td>
<td>1068</td>
</tr>
<tr>
<td>Fl_Widget</td>
<td>1090</td>
</tr>
<tr>
<td>Fl_Box</td>
<td>448</td>
</tr>
<tr>
<td>Fl_Button</td>
<td>489</td>
</tr>
<tr>
<td>Fl_Light_Button</td>
<td>673</td>
</tr>
<tr>
<td>Fl_Check_Button</td>
<td>509</td>
</tr>
<tr>
<td>Fl_Radio_Light_Button</td>
<td>781</td>
</tr>
<tr>
<td>Fl_Round_Button</td>
<td>797</td>
</tr>
<tr>
<td>Fl_Radio_Round_Button</td>
<td>782</td>
</tr>
<tr>
<td>Fl_Radio_Button</td>
<td>781</td>
</tr>
<tr>
<td>Fl_Repeat_Button</td>
<td>786</td>
</tr>
<tr>
<td>Fl_Return_Button</td>
<td>787</td>
</tr>
<tr>
<td>Fl_Shortcut_Button</td>
<td>824</td>
</tr>
<tr>
<td>Fl_Toggle_Button</td>
<td>995</td>
</tr>
<tr>
<td>Fl_Chart</td>
<td>496</td>
</tr>
<tr>
<td>Fl_Clock_Output</td>
<td></td>
</tr>
<tr>
<td>Fl_Clock</td>
<td>516</td>
</tr>
<tr>
<td>Fl_Round_Clock</td>
<td>798</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>572</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>573</td>
</tr>
<tr>
<td>Fl_FormsText</td>
<td>575</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>576</td>
</tr>
<tr>
<td>Fl_Group</td>
<td>593</td>
</tr>
</tbody>
</table>

Generated by Doxygen
<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Line_Dial</td>
<td>675</td>
</tr>
<tr>
<td>Fl_Roller</td>
<td>795</td>
</tr>
<tr>
<td>Fl_Slider</td>
<td>839</td>
</tr>
<tr>
<td>Fl_Fill_Slider</td>
<td>561</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider</td>
<td>619</td>
</tr>
<tr>
<td>Fl_Hor_Nice_Slider</td>
<td>619</td>
</tr>
<tr>
<td>Fl_Hor_Slider</td>
<td>620</td>
</tr>
<tr>
<td>Fl_Nice_Slider</td>
<td>721</td>
</tr>
<tr>
<td>Fl_Scrollbar</td>
<td>810</td>
</tr>
<tr>
<td>Fl_Value_Slider</td>
<td>1085</td>
</tr>
<tr>
<td>Fl_Hor_Value_Slider</td>
<td>621</td>
</tr>
<tr>
<td>Fl_Value_Input</td>
<td>1078</td>
</tr>
<tr>
<td>Fl_Value_Output</td>
<td>1082</td>
</tr>
<tr>
<td>Fl_Widget_Tracker</td>
<td>1135</td>
</tr>
<tr>
<td>Fl_XColor</td>
<td>1161</td>
</tr>
<tr>
<td>Fl_GIF_Image::GIF_FRAME</td>
<td>1163</td>
</tr>
<tr>
<td>Fl_ICO_Image::IconDirEntry</td>
<td>1163</td>
</tr>
<tr>
<td>Fl_Text_Editor::Key_Binding</td>
<td>1164</td>
</tr>
<tr>
<td>Fl_Preferences::Name</td>
<td>1164</td>
</tr>
<tr>
<td>Fl_Preferences::Node</td>
<td>1165</td>
</tr>
<tr>
<td>Fl_Paged_Device::page_format</td>
<td>1166</td>
</tr>
<tr>
<td>Fl_Preferences::RootNode</td>
<td>1166</td>
</tr>
<tr>
<td>Fl_Scroll::ScrollInfo</td>
<td>1166</td>
</tr>
<tr>
<td>Fl_Text_Display::Style_Table_Entry</td>
<td>1167</td>
</tr>
</tbody>
</table>
Chapter 30

Class Index

30.1 Class List

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

- Fl_GIF_Image::GIF_FRAME::CPAL
- Fl_Preferences::Entry
- Fl
  
The Fl is the FLTK global (static) class containing state information and global methods for the current application
- Fl_Adjuster
  
  Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range
- Fl_Anim_GIF_Image
  
  Supports loading, caching, and drawing of animated Compuserve GIF images
- Fl_Bitmap
  
  Supports caching and drawing of mono-color (bitmap) images
- Fl_BMP_Image
  
  Supports loading, caching, and drawing of Windows Bitmap (BMP) image files
- Fl_Box
  
  This widget simply draws its box, and possibly its label
- Fl_Browser
  
  Displays a scrolling list of text lines, and manages all the storage for the text
- Fl_Browser
  
  This is the base class for browsers
- Fl_Button
  
  Buttons generate callbacks when they are clicked by the user
- Fl_Cairo_State
  
  Contains all the necessary info on the current cairo context
- Fl_Cairo_Window
  
  This defines an FLTK window with Cairo support
- Fl_Chart
  
  Fl_Chart displays simple charts
- FL_CHART_ENTRY
  
  For internal use only
- Fl_Check_Browser
  
  Displays a scrolling list of text lines that may be selected and/or checked by the user
- Fl_Check_Button
  
  A button with a "checkmark" to show its status
- Fl_Choice
  
  A button that is used to pop up a menu
- Fl_Clock
  
  This widget provides a round analog clock display
- Fl_Clock_Output
  
  This widget can be used to display a program-supplied time
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Color_Chooser</td>
<td>Standard RGB color chooser</td>
<td>521</td>
</tr>
<tr>
<td>Fl_Copy_Surface</td>
<td>Supports copying of graphical data to the clipboard</td>
<td>526</td>
</tr>
<tr>
<td>Fl_Counter</td>
<td>Controls a single floating point value with button (or keyboard) arrows</td>
<td>529</td>
</tr>
<tr>
<td>Fl_Device_Plugin</td>
<td>This plugin socket allows the integration of new device drivers for special window or screen types</td>
<td>533</td>
</tr>
<tr>
<td>Fl_Dial</td>
<td>Circular dial to control a single floating point value</td>
<td>534</td>
</tr>
<tr>
<td>Fl_Display_Device</td>
<td>A display to which the computer can draw</td>
<td>536</td>
</tr>
<tr>
<td>Fl_Double_Window</td>
<td>The Fl_Double_Window provides a double-buffered window</td>
<td>537</td>
</tr>
<tr>
<td>Fl_End</td>
<td>This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:</td>
<td>539</td>
</tr>
<tr>
<td>Fl_EPS_File_Surface</td>
<td>Encapsulated PostScript drawing surface</td>
<td>540</td>
</tr>
<tr>
<td>Fl_File_Browser</td>
<td>Displays a list of filenames, optionally with file-specific icons</td>
<td>543</td>
</tr>
<tr>
<td>Fl_File_Chooser</td>
<td>Displays a standard file selection dialog that supports various selection modes</td>
<td>545</td>
</tr>
<tr>
<td>Fl_File_Icon</td>
<td>Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget</td>
<td>552</td>
</tr>
<tr>
<td>Fl_File_Input</td>
<td>This widget displays a pathname in a text input field</td>
<td>558</td>
</tr>
<tr>
<td>Fl_Fill_Dial</td>
<td>Draws a dial with a filled arc</td>
<td>561</td>
</tr>
<tr>
<td>Fl_Fill_Slider</td>
<td>Widget that draws a filled horizontal slider, useful as a progress or value meter</td>
<td>561</td>
</tr>
<tr>
<td>Fl_Flex</td>
<td>Fl_Flex is a container (layout) widget for one row or one column of widgets</td>
<td>562</td>
</tr>
<tr>
<td>Fl_Float_Input</td>
<td>Subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits)</td>
<td>571</td>
</tr>
<tr>
<td>Fl_FormsBitmap</td>
<td>Forms compatibility Bitmap Image Widget</td>
<td>572</td>
</tr>
<tr>
<td>Fl_FormsPixmap</td>
<td>Forms pixmap drawing routines</td>
<td>573</td>
</tr>
<tr>
<td>Fl_FormsText</td>
<td></td>
<td>575</td>
</tr>
<tr>
<td>Fl_Free</td>
<td>Emulation of the Forms &quot;free&quot; widget</td>
<td>576</td>
</tr>
<tr>
<td>Fl_GIF_Image</td>
<td>Supports loading, caching, and drawing of Compuserve GIF images</td>
<td>578</td>
</tr>
<tr>
<td>Fl_Gl_Choice</td>
<td></td>
<td>581</td>
</tr>
<tr>
<td>Fl_Gl_Window</td>
<td>Sets things up so OpenGL works</td>
<td>581</td>
</tr>
<tr>
<td>Fl_Glut_Bitmap_Font</td>
<td>Fltk glut font/size attributes used in the glutXXX functions</td>
<td>590</td>
</tr>
<tr>
<td>Fl_Glut_StrokeChar</td>
<td></td>
<td>590</td>
</tr>
<tr>
<td>Fl_Glut_StrokeFont</td>
<td></td>
<td>591</td>
</tr>
<tr>
<td>Fl_Glut_StrokeStrip</td>
<td></td>
<td>591</td>
</tr>
<tr>
<td>Fl_Glut_StrokeVertex</td>
<td></td>
<td>591</td>
</tr>
<tr>
<td>Fl_Glut_Window</td>
<td>GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibilitycxx):</td>
<td>591</td>
</tr>
</tbody>
</table>
30.1 Class List

Fl_Group
FLTK container widget .......................................................... 593

Fl_Help_Block

Fl_Help_Dialog
Displays a standard help dialog window using the Fl_Help_View widget ........ 607

Fl_Help_Font_Sock

Fl_Help_Font_Style
Fl_Help_View font stack element definition .................................. 610

Fl_Help_Link
Definition of a link for the html viewer ...................................... 610

Fl_Help_Target
Fl_Help_Target structure ......................................................... 611

Fl_Help_View
Displays HTML text ................................................................. 611

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 .......................... 618

Fl_Hor_Fill_Slider

Fl_Hor_Nice_Slider
Single thumb tab slider .......................................................... 619

Fl_Hor_Slider
Horizontal Slider class .............................................................. 620

Fl_Hor_Value_Slider

Fl_ICO_Image
Supports loading, caching, and drawing of Windows icon (.ico) files ........ 621

Fl_Image
Base class for image caching, scaling and drawing .......................... 622

Fl_Image_Reader

Fl_Image_Surface
Directs all graphics requests to an Fl_Image .................................. 633

Fl_Input
This is the FLTK text input widget ............................................ 637

Fl_Input_
This class provides a low-overhead text input field .......................... 640

Fl_Input_Choice
A combination of the input widget and a menu button .......................... 660

Fl_Int_Input
Subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaef) ........................................... 665

Fl_Int_Vector
An STL-ish vector without templates .......................................... 666

Fl_JPEG_Image
Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images .......................... 668

Fl_Kdialog_Native_File_Chooser_Driver ....................................... 671

Fl_Label
This struct stores all information for a text or mixed graphics label ....... 671

Fl_Light_Button
This subclass displays the "on" state by turning on a light, rather than drawing pushed in ............................................. 673

Fl_Line_Dial

Fl_Mac_App_Menu
Base class of all widgets that have a menu in FLTK .......................... 676

Fl_Menu
This widget provides a standard menubar interface ............................ 691

Fl_Menu_Bar
This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects ............................. 693
The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

Window type used for menus.

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

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.

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

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

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

This widget displays a piece of text.

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

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

Represents page-structured drawing surfaces.

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

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

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

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

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

This class is provided for Forms compatibility.

To send graphical output to a PostScript file.

Fi_Preferences store user settings between application starts.

OS-independent print support.

Displays a progress bar for the user.

A local struct to manage a region defined by left/right/top/bottom.

A local struct to manage a region defined by xywh.

The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down.
**Fl_Return_Button**

The **Fl_Return_Button** is a subclass of **Fl_Button** that generates a callback when it is pressed or when the user presses the Enter key.

**Fl_RGB_Image**

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

**Fl_Roller**

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

**Fl_Round_Button**

Buttons generate callbacks when they are clicked by the user.

**Fl_Round_Clock**

A clock widget of type `FL_ROUND_CLOCK`.

**Fl_Scheme**

**Fl_Scheme_Choice**

**Fl_Scroll**

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

**Fl_Scrollbar**

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

**Fl_Scrollbar::Fl_Scrollbar_Data**

A local struct to manage a scrollbar's xwh region and tab values.

**Fl_Secret_Input**

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

**Fl_Select_Browser**

The class is a subclass of **Fl_Browser** which lets the user select a single item, or no items by clicking on the empty space.

**Fl_Shared_Image**

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

**Fl_Shortcut_Button**

A button that allows the user to type a key combination to create shortcuts.

**Fl_Simple_Counter**

This widget creates a counter with only two arrow buttons.

**Fl_Simple_Terminal**

This is a continuous text scroll widget for logging and debugging output, much like a terminal.

**Fl_Single_Window**

This is the same as **Fl_Window**.

**Fl_Slider**

A sliding knob inside a box.

**Fl_Spinner**

This widget is a combination of a numerical input widget and repeat buttons.

**Fl_Spinner::Fl_Spinner_Input**

**Fl_String**

**Fl_String** is the basic string class for FLTK.

**Fl_Surface_Device**

A drawing surface that’s susceptible to receive graphical output.

**Fl_SVG_File_Surface**

A drawing surface producing a Scalable Vector Graphics (SVG) file.

**Fl_SVG_Image**

Supports loading, caching and drawing of scalable vector graphics (SVG) images.

**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.

**Fl_Table**

A table of widgets or other content.

**Fl_Table_Row**

A table with row selection capabilities.

**Fl_Tabs**

Container widget that displays a set of tabs, with each tab representing a different child widget.

**Fl_Text_Buffer**

This class manages Unicode text displayed in one or more **Fl_Text_Display** widgets.
FL_Text_Display
Rich text display widget ................................................................. 927
FL_Text_Editor
This is the FLTK text editor widget .................................................. 967
FL_Text_Selection
This is an internal class for FL_Text_Buffer to manage text selections .... 975
FL_Tile
Lets you resize its children by dragging the border between them .......... 979
FL_Tiled_Image
This class supports tiling of images over a specified area .................... 982
FL_Timeout
The internal class Fl_Timeout handles all timeout related functions ......... 985
FL_Timer
This is provided only to emulate the Forms Timer widget .................... 992
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 .................................................. 995
FL_Tooltip
Tooltip support for all FLTK widgets ............................................. 996
FL_Tree
Tree widget .................................................................................. 1001
FL_Tree_Item
Tree widget item ........................................................................... 1043
FL_Tree_Item_Array
Manages an array of Fl_Tree_Item pointers ....................................... 1065
FL_Tree_Prefs
Tree widget's preferences ................................................................ 1068
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 ................................................................. 1073
FL_Value_Input
Displays a numeric value .................................................................. 1078
FL_Value_Output
Displays a floating point value .......................................................... 1082
FL_Value_Slider
FL_Slider widget with a box displaying the current value .................... 1085
FL_Widget
FL_Widget is the base class for all widgets in FLTK .............................. 1090
FL_Widget_Surface
A surface on which any FLTK widget can be drawn .............................. 1132
FL_Widget_Tracker
This class should be used to control safe widget deletion .................... 1135
FL_Window
This widget produces an actual window ............................................. 1137
FL_Wizard
This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control ................................................................. 1159
FL_XBM_Image
Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files .... 1160
FL_XColor .......................................................... 1161
FL_XPM_Image
Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency .......................... 1161
FL_Zenity_Native_File_Chooser_Driver ................................................ 1162
Fl_GIF_Image::GIF_FRAME .......................................................... 1163
Fl_ICO_Image::IconDirEntry
Windows ICONDIRENTRY structure .................................................. 1163
30.1 Class List

**Fl_Text_Editor::Key_Binding**
Simple linked list item associating a key/state to a function .......................... 1164

**Fl_Preferences::Name**
'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly ......................................................... 1164

**Fl_Preferences::Node** ................................................................. 1165

**Fl_Paged_Device::page_format**
Width, height and name of a page format .................................................... 1166

**Fl_Preferences::RootNode** ............................................................ 1166

**Fl_Scroll::ScrollInfo**
Structure to manage scrollbar and widget interior sizes ............................... 1166

**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 ................................................................. 1167
Chapter 31

File Index

31.1  File List

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

- **Enumerations.H**
  This file contains type definitions and general enumerations .................................. 1169
- **filename.H**
  File names and URI utility functions ............................................................................. 1202
- **Fl.H**
  Fl static class .................................................................................................................. 1204
- **Fl_Adjuster.H**............................................................................................................. 1212
- **Fl_Anim_GIF_Image.H** ................................................................................................. 1213
- **fl_ask.H**
  API for common dialogs ................................................................................................. 1214
- **fl_attr.h**
  This file defines compiler-specific macros .................................................................... 1218
- **Fl_Bitmap.H**................................................................................................................. 1221
- **Fl_BMP_Image.H**.......................................................................................................... 1221
- **Fl_Box.H**
  Fl_Box widget ................................................................................................................ 1222
- **Fl_Browser.H**.............................................................................................................. 1223
- **Fl_Browser_.H**............................................................................................................. 1224
- **Fl_Button.H**............................................................................................................... 1226
- **Fl_Cairo.H**
  Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland) ................................................................. 1227
- **Fl_Cairo_Window.H**
  Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback ..................... 1228
- **fl_casts.H**................................................................................................................... 1230
- **Fl_Chart.H**
  Fl_Chart widget .............................................................................................................. 1230
- **Fl_Check_Browser.H**.................................................................................................. 1232
- **Fl_Check_Button.H**..................................................................................................... 1233
- **Fl_Choice.H**............................................................................................................... 1234
- **Fl_Clock.H**................................................................................................................ 1234
- **Fl_Color_Chooser.H**
  Fl_Color_Chooser widget ............................................................................................... 1235
- **fl_config.h**................................................................................................................. 1237
- **Fl_Copy_Surface.H**.................................................................................................... 1238
- **Fl_Counter.H**............................................................................................................ 1239
- **Fl_Device.H**
  Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin ........... 1240
- **Fl.Dial.H**................................................................................................................... 1241
- **Fl_Double_Window.H**................................................................................................ 1241
<table>
<thead>
<tr>
<th>Header Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_draw.H</td>
<td>1242</td>
</tr>
<tr>
<td>Fl_Export.H</td>
<td>1253</td>
</tr>
<tr>
<td>Fl_File_Browser.H</td>
<td>1254</td>
</tr>
<tr>
<td>Fl_File_Chooser.H</td>
<td>1255</td>
</tr>
<tr>
<td>Fl_File_Icon.H</td>
<td>1257</td>
</tr>
<tr>
<td>Fl_File_Input.H</td>
<td>1258</td>
</tr>
<tr>
<td>Fl_Fill_Dial.H</td>
<td>1259</td>
</tr>
<tr>
<td>Fl_Fill_Slider.H</td>
<td>1259</td>
</tr>
<tr>
<td>Fl_Flex.H</td>
<td>1260</td>
</tr>
<tr>
<td>Fl_Float_Input.H</td>
<td>1261</td>
</tr>
<tr>
<td>Fl_FormsBitmap.H</td>
<td>1262</td>
</tr>
<tr>
<td>Fl_FormsPixmap.H</td>
<td>1262</td>
</tr>
<tr>
<td>Fl_Free.H</td>
<td>1263</td>
</tr>
<tr>
<td>Fl_GIF_Image.H</td>
<td>1263</td>
</tr>
<tr>
<td>Fl_GL_Window.H</td>
<td>1264</td>
</tr>
<tr>
<td>Fl_Graphics_Driver.H</td>
<td>1266</td>
</tr>
<tr>
<td>Fl_Group.H</td>
<td>1271</td>
</tr>
<tr>
<td>Fl_Group_H</td>
<td>1272</td>
</tr>
<tr>
<td>Fl_Hold_Browser.H</td>
<td>1273</td>
</tr>
<tr>
<td>Fl_Hor_Fill_Slider.H</td>
<td>1276</td>
</tr>
<tr>
<td>Fl_Hor_Nice_Slider.H</td>
<td>1277</td>
</tr>
<tr>
<td>Fl_Hor_Slider.H</td>
<td>1277</td>
</tr>
<tr>
<td>Fl_Hor_Value_Slider.H</td>
<td>1278</td>
</tr>
<tr>
<td>Fl_ICO_Image.H</td>
<td>1278</td>
</tr>
<tr>
<td>Fl_Image.H</td>
<td>1278</td>
</tr>
<tr>
<td>Fl_Image_Surface.H</td>
<td>1282</td>
</tr>
<tr>
<td>Fl_Input.H</td>
<td>1282</td>
</tr>
<tr>
<td>Fl_Input_H</td>
<td>1283</td>
</tr>
<tr>
<td>Fl_Input_H</td>
<td>1286</td>
</tr>
<tr>
<td>Fl_Int_Input.H</td>
<td>1288</td>
</tr>
<tr>
<td>Fl_Int_Vector.H</td>
<td>1289</td>
</tr>
<tr>
<td>Fl_JPEG_Image.H</td>
<td>1290</td>
</tr>
<tr>
<td>Fl_Light_Button.H</td>
<td>1290</td>
</tr>
<tr>
<td>Fl_Line_Dial.H</td>
<td>1291</td>
</tr>
<tr>
<td>Fl_Menu.H</td>
<td>1291</td>
</tr>
<tr>
<td>Fl_Menu_H</td>
<td>1291</td>
</tr>
<tr>
<td>Fl_Menu_Bar.H</td>
<td>1293</td>
</tr>
<tr>
<td>Fl_Menu_Button.H</td>
<td>1293</td>
</tr>
<tr>
<td>Fl_Menu_Item.H</td>
<td>1294</td>
</tr>
<tr>
<td>Fl_Menu_Window.H</td>
<td>1297</td>
</tr>
<tr>
<td>Fl_message.H</td>
<td>1297</td>
</tr>
<tr>
<td>Fl_Multi_Browser.H</td>
<td>1298</td>
</tr>
<tr>
<td>Fl_Multi_Label.H</td>
<td>1298</td>
</tr>
<tr>
<td>Fl_MultiLine_Input.H</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_MultiLine_Output.H</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser.H</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser_H</td>
<td>1299</td>
</tr>
<tr>
<td>Fl_Nice_Slider.H</td>
<td>1302</td>
</tr>
<tr>
<td>Fl_Object.H</td>
<td>1302</td>
</tr>
<tr>
<td>Fl_Output.H</td>
<td>1303</td>
</tr>
<tr>
<td>Fl_Overlay_Window.H</td>
<td>1303</td>
</tr>
<tr>
<td>Fl_Pack.H</td>
<td>1304</td>
</tr>
</tbody>
</table>

Generated by Doxygen
31.1 File List

**FL_Paged_Device.H**
Declaration of class **FL_Paged_Device** .......................................................... 1304

**FL_Pixmap.H** .................................................................................................. 1306
**FL_Plugin.H** ................................................................................................. 1307
**FL_PNG_ImaGe.H** ......................................................................................... 1307
**FL_PNM_ImaGe.H** ......................................................................................... 1308
**FL_Positioner.H** ......................................................................................... 1308
**FL_PostScript.H**
Declaration of classes **FL_PostScript_File_Device** and **FL_EPS_File_Surface** .......... 1309

**FL_Preferences.H** ......................................................................................... 1311
**FL_Printer.H**
Declaration of class **FL_Printer** ................................................................. 1314

**FL_Progress.H** ............................................................................................ 1315
**FL_Radio_Button.H** ................................................................................... 1316
**FL_Radio_Light_Button.H** ......................................................................... 1316
**FL_Radio_Round_Button.H** ....................................................................... 1317

**FL_Rect.H** ................................................................................................... 1317
**FL_Repeat_Button.H** ................................................................................ 1318
**FL_Return_Button.H** ................................................................................ 1318
**FL_RGB_ImaGe.H** ..................................................................................... 1319
**FL_Roller.H** ............................................................................................... 1319
**FL_Round_Button.H** ................................................................................ 1320
**FL_Round_Clock.H** .................................................................................. 1320
**FL_Scheme.H** ............................................................................................ 1320
**FL_Scheme_Choice.H** ............................................................................... 1321

**FL_Scroll.H** ............................................................................................. 1321
**FL_Scrollbar.H** ....................................................................................... 1322
**FL_Secret_Input.H** .................................................................................. 1323
**FL_Select_Browser.H** .............................................................................. 1324
**FL_Shared_ImaGe.H**
**FL_Shared_ImaGe class** ............................................................................ 1325

**FL_Show_Colormap.H**
The **fl_show_colormap()** function hides the implementation classes used to provide the popup window and color selection mechanism .................................................. 1328

**fl_show_input.H** .......................................................................................... 1328
**FL_Simple_Counter.H** ............................................................................... 1329
**FL_Simple_Terminal.H** ............................................................................. 1329
**FL_Single_Window.H** ................................................................................. 1331
**FL_Slider.H** ............................................................................................... 1331
**FL_Spinner.H** ........................................................................................... 1332
**FL_String.H**
Basic **FL_Sring** class for FLTK ................................................................ 1334

**fl_string_functions.h**
Public header for FLTK's own platform agnostic string handling ...................... 1336

**FL_SVG_File_Surface.H** ........................................................................... 1337
**FL_SVG_ImaGe.H** ........................................................................................ 1338
**FL_Sys_Menu_Bar.H**
Definition of class **FL_SYS_Menu_Bar** ....................................................... 1338

**FL_Table.H** ............................................................................................... 1340
**FL_Table_Row.H** ....................................................................................... 1345
**FL_Tabs.H** .................................................................................................. 1347
**FL_Text_Buffer.H** .................................................................................... 1348
**FL_Text_Display.H** ................................................................................ 1352
**FL_Text_Editor.H** .................................................................................... 1357
**FL_Tile.H** ................................................................................................... 1358
**FL_Tiled_ImaGe.H** .................................................................................... 1359
**FL_Timer.H** ............................................................................................... 1359
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Toggle_Button.H</td>
<td>This file contains the definitions of the Fl_Tree class</td>
<td>1360</td>
</tr>
<tr>
<td>Fl_Toggle_Light_Button.H</td>
<td></td>
<td>1360</td>
</tr>
<tr>
<td>Fl_Toggle_Round_Button.H</td>
<td></td>
<td>1361</td>
</tr>
<tr>
<td>Fl_Tooltip.H</td>
<td></td>
<td>1361</td>
</tr>
<tr>
<td>Fl_Tree.H</td>
<td></td>
<td>1362</td>
</tr>
<tr>
<td>Fl_Tree_Item.H</td>
<td>This file contains the definitions for Fl_Tree_Item</td>
<td>1366</td>
</tr>
<tr>
<td>Fl_Tree_Item_Array.H</td>
<td>This file defines a class that manages an array of Fl_Tree_Item pointers</td>
<td>1370</td>
</tr>
<tr>
<td>Fl_Tree_Prefs.H</td>
<td>This file contains the definitions for Fl_Tree's preferences</td>
<td>1371</td>
</tr>
<tr>
<td>fl_types.h</td>
<td>This file contains simple &quot;C&quot;-style type definitions</td>
<td>1377</td>
</tr>
<tr>
<td>fl_utf8.h</td>
<td>Header for Unicode and UTF-8 character handling</td>
<td>1378</td>
</tr>
<tr>
<td>Fl_Valuator.H</td>
<td></td>
<td>1382</td>
</tr>
<tr>
<td>Fl_Value_Input.H</td>
<td></td>
<td>1383</td>
</tr>
<tr>
<td>Fl_Value_Output.H</td>
<td></td>
<td>1384</td>
</tr>
<tr>
<td>Fl_Value_Slider.H</td>
<td></td>
<td>1385</td>
</tr>
<tr>
<td>Fl_Widget.H</td>
<td>Fl_Widget and Fl_Label classes</td>
<td>1386</td>
</tr>
<tr>
<td>Fl_Widget_Surface.H</td>
<td></td>
<td>1391</td>
</tr>
<tr>
<td>Fl_Window.H</td>
<td>Fl_Window widget</td>
<td>1391</td>
</tr>
<tr>
<td>Fl_Wizard.H</td>
<td></td>
<td>1394</td>
</tr>
<tr>
<td>Fl_XBM_Image.H</td>
<td></td>
<td>1395</td>
</tr>
<tr>
<td>Fl_XPM_Image.H</td>
<td></td>
<td>1395</td>
</tr>
<tr>
<td>forms.H</td>
<td></td>
<td>1396</td>
</tr>
<tr>
<td>gl.h</td>
<td>This file defines wrapper functions for OpenGL in FLTK</td>
<td>1405</td>
</tr>
<tr>
<td>gl2opengl.h</td>
<td></td>
<td>1410</td>
</tr>
<tr>
<td>gl_draw.H</td>
<td></td>
<td>1411</td>
</tr>
<tr>
<td>glu.h</td>
<td></td>
<td>1411</td>
</tr>
<tr>
<td>glut.H</td>
<td></td>
<td>1412</td>
</tr>
<tr>
<td>mac.h</td>
<td>Mac OS X-specific symbols</td>
<td>1417</td>
</tr>
<tr>
<td>math.h</td>
<td></td>
<td>1420</td>
</tr>
<tr>
<td>names.h</td>
<td>This file defines arrays of human readable names for FLTK symbolic constants</td>
<td>1420</td>
</tr>
<tr>
<td>platform.H</td>
<td></td>
<td>1422</td>
</tr>
<tr>
<td>platform_types.h</td>
<td>Definitions of platform-dependent types</td>
<td>1423</td>
</tr>
<tr>
<td>wayland.H</td>
<td>Definitions of functions specific to the Wayland platform</td>
<td>1426</td>
</tr>
<tr>
<td>win32.H</td>
<td>Definitions of functions specific to the Windows platform</td>
<td>1427</td>
</tr>
<tr>
<td>x.H</td>
<td>Definitions of functions specific to the X11 platform</td>
<td>1429</td>
</tr>
<tr>
<td>x11.H</td>
<td>Definitions of functions specific to the X11 platform</td>
<td>1429</td>
</tr>
<tr>
<td>cgdebug.h</td>
<td></td>
<td>1431</td>
</tr>
<tr>
<td>fastarrow.h</td>
<td></td>
<td>1434</td>
</tr>
<tr>
<td>Fl.cxx</td>
<td>Implementation of the member functions of class Fl</td>
<td>1434</td>
</tr>
<tr>
<td>fl_arc.cxx</td>
<td>Utility functions for drawing arcs and circles</td>
<td>1436</td>
</tr>
<tr>
<td>fl_ask.cxx</td>
<td>Utility functions for common dialogs</td>
<td>1436</td>
</tr>
</tbody>
</table>
31.1 File List

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_boxtype.cxx</td>
<td>Drawing code for common box types</td>
<td>1438</td>
</tr>
<tr>
<td>fl_cmap.h</td>
<td></td>
<td>1439</td>
</tr>
<tr>
<td>fl_color.cxx</td>
<td>Color handling</td>
<td>1443</td>
</tr>
<tr>
<td>Fl_compose.cxx</td>
<td>Utility functions to support text input</td>
<td>1443</td>
</tr>
<tr>
<td>fl_contrast.cxx</td>
<td>Color contrast handling</td>
<td>1443</td>
</tr>
<tr>
<td>fl_curve.cxx</td>
<td>Utility for drawing Bézier curves, adding the points to the</td>
<td>1444</td>
</tr>
<tr>
<td></td>
<td>current fl_begin/fl_vertex/fl_end path</td>
<td></td>
</tr>
<tr>
<td>Fl_Double_Window.cxx</td>
<td>Fl_Double_Window implementation</td>
<td>1444</td>
</tr>
<tr>
<td>Fl_GL_Choice.H</td>
<td></td>
<td>1445</td>
</tr>
<tr>
<td>Fl_GL_Window_Driver.H</td>
<td></td>
<td>1445</td>
</tr>
<tr>
<td>Fl_Image_Reader.h</td>
<td></td>
<td>1447</td>
</tr>
<tr>
<td>Fl_Message.h</td>
<td></td>
<td>1448</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser_Kdialog.H</td>
<td></td>
<td>1449</td>
</tr>
<tr>
<td>Fl_Native_File_Chooser_Zenity.H</td>
<td></td>
<td>1450</td>
</tr>
<tr>
<td>fl_oxy.h</td>
<td></td>
<td>1451</td>
</tr>
<tr>
<td>Fl_Paged_Device.cxx</td>
<td>Implementation of class Fl_Paged_Device</td>
<td>1451</td>
</tr>
<tr>
<td>fl_rect.cxx</td>
<td>Drawing and clipping routines for rectangles</td>
<td>1451</td>
</tr>
<tr>
<td>Fl_Screen_Driver.H</td>
<td></td>
<td>1451</td>
</tr>
<tr>
<td>Fl_String.cxx</td>
<td>Basic Fl_String class for FLTK</td>
<td>1454</td>
</tr>
<tr>
<td>Fl_Sys_Menu_Bar_Driver.H</td>
<td></td>
<td>1456</td>
</tr>
<tr>
<td>Fl_System_Driver.H</td>
<td></td>
<td>1456</td>
</tr>
<tr>
<td>Fl_Timeout.cxx</td>
<td>Fl_Timeout handling</td>
<td>1459</td>
</tr>
<tr>
<td>Fl_Timeout.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fl_vertex.cxx</td>
<td>Portable drawing code for drawing arbitrary shapes with simple</td>
<td>1461</td>
</tr>
<tr>
<td></td>
<td>2D transformations</td>
<td></td>
</tr>
<tr>
<td>Fl_Window_Driver.H</td>
<td></td>
<td>1461</td>
</tr>
<tr>
<td>fl_write_png.cxx</td>
<td>PNG image support functions</td>
<td>1463</td>
</tr>
<tr>
<td>Fl_XColor.H</td>
<td></td>
<td>1465</td>
</tr>
<tr>
<td>fstrftime.h</td>
<td></td>
<td>1466</td>
</tr>
<tr>
<td>freeglut_teatpot_data.h</td>
<td></td>
<td>1467</td>
</tr>
<tr>
<td>mediumarrow.h</td>
<td></td>
<td>1469</td>
</tr>
<tr>
<td>numericsort.c</td>
<td></td>
<td>1469</td>
</tr>
<tr>
<td>print_button.h</td>
<td></td>
<td>1470</td>
</tr>
<tr>
<td>print_panel.h</td>
<td></td>
<td>1471</td>
</tr>
<tr>
<td>slowarrow.h</td>
<td></td>
<td>1471</td>
</tr>
<tr>
<td>utf8_internal.h</td>
<td></td>
<td>1471</td>
</tr>
<tr>
<td>vsnprintf.c</td>
<td>Portable vsnprintf() implementation</td>
<td>1472</td>
</tr>
<tr>
<td>Xuf8.h</td>
<td></td>
<td>1473</td>
</tr>
<tr>
<td>case.h</td>
<td></td>
<td>1475</td>
</tr>
<tr>
<td>dingbats_.h</td>
<td></td>
<td>1495</td>
</tr>
<tr>
<td>spacing.h</td>
<td></td>
<td>1502</td>
</tr>
<tr>
<td>symbol_.h</td>
<td></td>
<td>1525</td>
</tr>
<tr>
<td>armscii_8.h</td>
<td></td>
<td>1538</td>
</tr>
<tr>
<td>ascii.h</td>
<td></td>
<td>1539</td>
</tr>
<tr>
<td>big5.h</td>
<td></td>
<td>1539</td>
</tr>
<tr>
<td>big5_emacs.h</td>
<td></td>
<td>1587</td>
</tr>
<tr>
<td>cp1133.h</td>
<td></td>
<td>1589</td>
</tr>
</tbody>
</table>
Chapter 32

Module Documentation

32.1 Callback Function Typedefs

Typedefs defined in `<FL/Fl.H>` for callback or handler functions passed as function parameters.

**Typedefs**

- typedef void(* Fl_Abort_Handler)(const char *format,...)
  
  Signature of set_abort functions passed as parameters.

- typedef int(* Fl_Args_Handler)(int argc, char **argv, int &i)
  
  Signature of args functions passed as parameters.

- typedef void(* Fl_Atclose_Handler)(Fl_Window *window, void *data)
  
  Signature of set_atclose functions passed as parameters.

- typedef void(* Fl_Awake_Handler)(void *data)
  
  Signature of some wakeup callback functions passed as parameters.

- typedef void(Fl_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color)
  
  Signature of some box drawing functions passed as parameters.

- typedef void(* Fl_Clipboard_Notify_Handler)(int source, void *data)
  
  Signature of add_clipboard_notify functions passed as parameters.

- typedef int(* Fl_Event_Dispatch)(int event, Fl_Window *w)
  
  Signature of event_dispatch functions passed as parameters.

- typedef int(* Fl_Event_Handler)(int event)
  
  Signature of add_handler functions passed as parameters.

- typedef void(* Fl_FD_Handler)(FL_SOCKET fd, void *data)
  
  Signature of add_fd functions passed as parameters.

- typedef void(* Fl_Idle_Handler)(void *data)
  
  Signature of add_idle callback functions passed as parameters.

- typedef void(Fl_Label_Draw_F)(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align)
  
  Signature of some label drawing functions passed as parameters.

- typedef void(Fl_Label_Measure_F)(const Fl_Label *label, int &width, int &height)
  
  Signature of some label measurement functions passed as parameters.

- typedef void(* Fl_Old_Idle_Handler)()
  
  Signature of set_idle callback functions passed as parameters.

- typedef int(* Fl_System_Handler)(void *event, void *data)
  
  Signature of add_system_handler functions passed as parameters.

- typedef void(* Fl_Timeout_Handler)(void *data)
  
  Signature of timeout callback functions passed as parameters.
32.1.1 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

32.1.2 Typedef Documentation

32.1.2.1 Fl_Event_Dispatch

typedef int(*)(Fl_Event_Dispatch) (int event, Fl_Window *)
Signature of event_dispatch functions passed as parameters.

See also

Fl::event_dispatch(Fl_Event_Dispatch)

32.1.2.2 Fl_Timeout_Handler

typedef void(*)(Fl_Timeout_Handler) (void *data)
Signature of timeout callback functions passed as parameters.

Please see Fl::add_timeout() for details.

32.2 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.
32.2 Windows handling functions

Variables

- static void(* Fl::atclose)(Fl_Window *, void *)

    Back compatibility: default window callback handler.

32.2.1 Detailed Description

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

32.2.2 Function Documentation

32.2.2.1 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.

32.2.2.2 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.

32.2.2.3 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.

32.2.2.4 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.

32.2.2.5 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.

### 32.2.2.6 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).

### 32.2.2.7 next_window()

```c
Fl_Window * Fl::next_window (const Fl_Window * window ) [static]
```

Returns the next top-level window in the list of shown() windows.
You can use this call to iterate through all the windows that are shown().
Parameters

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

### 32.2.2.8 set_atclose()

```c
static void Fl::set_atclose (Fl_Atclose_Handler f ) [inline], [static]
```

For back compatibility, sets the Fl::atclose handler callback.
You can now simply change the callback for the window instead.

See also

Fl_Window::callback(Fl_Callback *)

### 32.2.3 Variable Documentation

#### 32.2.3.1 atclose

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

Back compatibility: default window callback handler.

See also

Fl::set_atclose()

### 32.3 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)
32.3 Events handling functions

Install a function to intercept system events.

- static Fl_Widget * Fl::belowmouse ()
  Gets the widget that is below the mouse.
- static void Fl::belowmouse (Fl_Widget *)
  Sets the widget that is below the mouse.
- static Fl_Callback_Reason Fl::callback_reason ()
  Give the reason for calling a callback.
- static int Fl::compose (int &del)
  Any text editing widget should call this for each FL_KEYBOARD event.
- static void Fl::compose_reset ()
  If the user moves the cursor, be sure to call Fl::compose_reset().
- static void Fl::disable_im ()
  Disables the system input methods facilities.
- static void Fl::enable_im ()
  Enables the system input methods facilities.
- static int Fl::event ()
  Returns the last event that was processed.
- static int Fl::event_alt ()
  Returns non-zero if the Alt key is pressed.
- static int Fl::event_button ()
  Gets which particular mouse button caused the current event.
- static int Fl::event_button1 ()
  Returns non-zero if mouse button 1 is currently held down.
- static int Fl::event_button2 ()
  Returns non-zero if button 2 is currently held down.
- static int Fl::event_button3 ()
  Returns non-zero if button 3 is currently held down.
- static int Fl::event_buttons ()
  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- static int Fl::event_clicks ()
  Returns non zero if we had a double click event.
- static void Fl::event_clicks (int i)
  Manually sets the number returned by Fl::event_clicks().
- static void * Fl::event_clipboard ()
  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- static const char * Fl::event_clipboard_type ()
  Returns the type of the pasted data during an FL_PASTE event.
- static int Fl::event_command ()
  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- static int Fl::event_ctrl ()
  Returns non-zero if the Control key is pressed.
- static Fl_Event_Dispatch Fl::event_dispatch ()
  Return the current event dispatch function.
- static void Fl::event_dispatch (Fl_Event_Dispatch d)
  Set a new event dispatch function.
- static int Fl::event_dx ()
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int Fl::event_dy ()
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int Fl::event_inside (const Fl_Widget *)
  Returns whether or not the mouse event is inside a given child widget.
- static int Fl::event_inside (int, int, int, int)
  Returns whether or not the mouse event is inside the given rectangle.
- static int Fl::event_is_click ()
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH
  or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- static void Fl::event_is_click (int i)
  Clears the value returned by Fl::event_is_click().
- static int Fl::event_key ()
  Gets which key on the keyboard was last pushed.
- static int Fl::event_key (int key)
  Returns true if the given key was held down (or pressed) during the last event.
- static int Fl::event_length ()
  Returns the length of the text in Fl::event_text().
- static int Fl::event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.
- static int Fl::event_shift ()
  Returns non-zero if the Shift key is pressed.
- static int Fl::event_state ()
  Returns the keyboard and mouse button states of the last event.
- static int Fl::event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.
- static const char * Fl::event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- static int Fl::event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- static int Fl::event_x_root ()
  Returns the mouse position on the screen of the event.
- static int Fl::event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- static int Fl::event_y_root ()
  Returns the mouse position on the screen of the event.
- static Fl_Widget * Fl::focus ()
  Gets the current Fl::focus() widget.
- static void Fl::focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.
- static int Fl::get_key (int key)
  Returns true if the given key is held down now.
- static void Fl::get_mouse (int &, int &)
  Returns 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 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).
32.3 Events handling functions

Variables

- const char *const fl_callback_reason_names []
  This is an array of callback reason names you can use to convert font numbers into names.
- const char *const fl_eventnames []
  This is an array of event names you can use to convert event numbers into names.
- const char *const fl_fontnames []
  This is an array of font names you can use to convert font numbers into names.

32.3.1 Detailed Description

Fl class events handling API declared in `<FL/Fl.H`

32.3.2 Function Documentation

32.3.2.1 add_handler()

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 *)

32.3.2.2 add_system_handler()

void Fl::add_system_handler ( 
  Fl_System_Handler ha,
  void * data ) [static]

Install a function to intercept system events.
FLTK calls each of these functions as soon as a new system event is received. The processing will stop at the first
function to return non-zero. If all functions return zero then the event is passed on for normal handling by FLTK.
Each function will be called with a pointer to the system event as the first argument and data as the second
argument. The system event pointer will always be void *, but will point to different objects depending on the
platform:

- X11: XEvent
- Windows: MSG
- OS X: NSEvent
Parameters

<table>
<thead>
<tr>
<th>ha</th>
<th>The event handler function to register</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>User data to include on each call</td>
</tr>
</tbody>
</table>

See also

`Fl::remove_system_handler(Fl_System_Handler)`

### 32.3.2.3 belowmouse() [1/2]

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

Gets the widget that is below the mouse.

See also

`belowmouse(Fl_Widget*)`

### 32.3.2.4 belowmouse() [2/2]

```c
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()).

### 32.3.2.5 callback_reason()

```c
Fl_Callback_Reason Fl::callback_reason () [static]
```

Give the reason for calling a callback.

Returns

the reason for the current callback

See also

`Fl_Widget::when(), Fl_Widget::do_callback(), Fl_Widget::callback()`

### 32.3.2.6 compose()

```c
int Fl::compose ( int & del ) [static]
```

Any text editing widget should call this for each FL_KEYBOARD event. Use of this function is very simple. If true is returned, then it has modified the Fl::event_text() and Fl::event_length() to a set of bytes to insert (it may be of zero length!). It will also set the "del" parameter to the number of bytes to the left of the cursor to delete, this is used to delete the results of the previous call to Fl::compose(). If false is returned, the keys should be treated as function keys, and del is set to zero. You could insert the text anyways, if you don't know what else to do. 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.
32.3 Events handling functions

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.

32.3.2.7 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.

32.3.2.8 disable_im()

void Fl::disable_im ( ) [static]

Disables the system input methods facilities.

See also

   enable_im()

32.3.2.9 enable_im()

void Fl::enable_im ( ) [static]

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

See also

   disable_im()

32.3.2.10 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.

32.3.2.11 event_button()

static int Fl::event_button ( ) [inline], [static]

Gets which particular mouse button caused the current event.
This returns garbage if the most recent event was not a FL_PUSH or FL_RELEASE event.
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_LEFT_MOUSE</td>
<td></td>
</tr>
<tr>
<td>FL_MIDDLE_MOUSE</td>
<td></td>
</tr>
<tr>
<td>FL_RIGHT_MOUSE</td>
<td></td>
</tr>
</tbody>
</table>

See also

Fl::event_buttons()

32.3.2.12 event_button1()

static int Fl::event_button1 ( ) [inline], [static]

Returns non-zero if mouse button 1 is currently held down.
For more details, see Fl::event_buttons().

32.3.2.13 event_button2()

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

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

32.3.2.14 event_button3()

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

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

32.3.2.15 event_buttons()

static int Fl::event_buttons ( ) [inline], [static]

Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
This function returns the button state at the time of the event. During an FL_RELEASE event, the state of the
released button will be 0. To find out, which button caused an FL_RELEASE event, you can use Fl::event_button()
instead.

Returns

a bit mask value like { [FL_BUTTON1] | [FL_BUTTON2] | [FL_BUTTON3] }

32.3.2.16 event_clicks() [1/2]

static int Fl::event_clicks ( ) [inline], [static]

Returns non zero if we had a double click event.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero</td>
<td>if the most recent FL_PUSH or FL_KEYBOARD was a &quot;double click&quot;.</td>
</tr>
<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>

32.3.2.17 event_clicks() [2/2]

static void Fl::event_clicks (  
    int i ) [inline], [static]
Manually sets the number returned by `Fl::event_clicks()`.
This can be used to set it to zero so that later code does not think an item was double-clicked.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>corresponds to no double-click if 0, i+1 mouse clicks otherwise</td>
</tr>
</tbody>
</table>

**See also**

- `int event_clicks()`

### 32.3.2.18 event_clipboard()

```c
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`.

### 32.3.2.19 event_clipboard_type()

```c
static const char * Fl::event_clipboard_type ( ) [inline], [static]
```

Returns the type of the pasted data during an FL_PASTE event.
This type can be `Fl::clipboard_plain_text` or `Fl::clipboard_image`.

### 32.3.2.20 event_dispatch()

```c
void Fl::event_dispatch ( Fl_Event_Dispatch d ) [static]
```

Set a new event dispatch function.
The event dispatch function is called after native events are converted to FLTK events, but before they are handled by FLTK. If the dispatch function `Fl_Event_Dispatch d` is set, it is up to the dispatch function to call `Fl::handle_(int, Fl_Window *)` or to ignore the event.
The dispatch function itself must return 0 if it ignored the event, or non-zero if it used the event. If you call `Fl::handle_()`, then this will return the correct value.
The event dispatch can be used to handle exceptions in FLTK events and callbacks before they reach the native event handler:

```c
int myHandler(int e, Fl_Window *w) {
    try {
        return Fl::handle_(e, w);
    } catch (...) {
        ...
    }
}
main() {
    Fl::event_dispatch(myHandler);
    ...
    Fl::run();
}
```

**Parameters**

| d | new dispatch function, or NULL |

**See also**

- `Fl::add_handler(Fl_Event_Handler)`
- `Fl::handle(int, Fl_Window *)`
- `Fl::handle_(int, Fl_Window *)`
32.3.2.21 event_dx()

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

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

32.3.2.22 event_dy()

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

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

32.3.2.23 event_inside() [1/2]

int Fl::event_inside ( const Fl_Widget * o ) [static]

Returns whether or not the mouse event is inside a given child widget.
Returns non-zero if the current Fl::event_x() and Fl::event_y() put it inside the given child widget's bounding box.
This method can only be used to check whether the mouse event is inside a child widget of the window that handles the event, and there must not be an intermediate subwindow (i.e. the widget must not be inside a subwindow of the current window). However, it is valid if the widget is inside a nested Fl_Group.
You must not use it with the window itself as the o argument in a window's handle() method.

Note
The mentioned restrictions are necessary, because this method does not transform coordinates of child widgets, and thus the given widget o must be within the same window that is handling the current event. Otherwise the results are undefined.

You should always call this rather than doing your own comparison so you are consistent about edge effects.

See also
Fl::event_inside(int, int, int, int)

Parameters

| in | o  | child widget to be tested |

Returns

non-zero, if mouse event is inside the widget

32.3.2.24 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

| in | xx,yy,ww,hh | bounding box |

Generated by Doxygen
Returns
non-zero, if mouse event is inside

32.3.2.25  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.

32.3.2.26  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.

32.3.2.27  event_key() [1/2]
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&).

32.3.2.28  event_key() [2/2]
int Fl::event_key ( int key ) [static]
Returns true if the given key was held down (or pressed) during the last event.
This is constant until the next event is read from the server.
Fl::get_key(int) returns true if the given key is held down now. Under X this requires a round-trip to the server and is
much slower than Fl::event_key(int).
Keys are identified by the unshifted values. FLTK defines a set of symbols that should work on most modern
machines for every key on the keyboard:

- All keys on the main keyboard producing a printable ASCII character use the value of that ASCII character
  (as though shift, ctrl, and caps lock were not on). The space bar is 32.
- All keys on the numeric keypad producing a printable ASCII character use the value of that ASCII character
  plus FL_KP (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.

32.3.2.29  

32.3.2.30  

32.3.2.31  

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.

32.3.2.32  

Use mask to pass the event states you’re interested in. The legal event state bits are defined in Fl::event_state().
32.3 Events handling functions

32.3.2.33  event_text()

static const char * Fl::event_text ( ) [inline], [static]
Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
This can be used in response to FL_KEYUP, FL_KEYDOWN, FL_PASTE, and FL_DND_RELEASE.
When responding to FL_KEYUP/FL_KEYDOWN, use this function instead of Fl::event_key() to get the text equivalent of keystrokes suitable for inserting into strings and text widgets.
The returned string is guaranteed to be NULL terminated. However, see Fl::event_length() for the actual length of the string, in case the string itself contains NULLs that are part of the text data.

Returns

A NULL terminated text string equivalent of the last keystroke.

32.3.2.34  event_x_root()

static int Fl::event_x_root ( ) [inline], [static]
Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and event_x(),event_y().

32.3.2.35  event_y_root()

static int Fl::event_y_root ( ) [inline], [static]
Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and event_x(),event_y().

32.3.2.36  focus() [1/2]

static Fl_Widget * Fl::focus ( ) [inline], [static]
Gets the current Fl::focus() widget.

See also

Fl::focus(Fl_Widget*)

32.3.2.37  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_focus
UNFOCUS events. Changing the focus does not send FL_FOCUS to this or any widget, because sending FL_focus
FOCUS is supposed to test if the widget wants the focus (by it returning non-zero from handler()).
Widgets can set the NEEDS_KEYBOARD flag to indicate that a keyboard is essential for the widget to function.
Touchscreen devices will be sent a request to show an on-screen keyboard if no hardware keyboard is connected.

See also

Fl_Widget::take_focus()
Fl_Widget::needs_keyboard() const
Fl_Widget::needs_keyboard(bool)
32.3.2.38  get_key()

int Fl::get_key (  
    int key  ) [static]

Returns true if the given key is held down now.  
Under X this requires a round-trip to the server and is much slower than Fl::event_key(int).

See also

    event_key(int)

32.3.2.39  get_mouse()

void Fl::get_mouse ( 
    int & x,  
    int & y  ) [static]

Return where the mouse is on the screen by doing a round-trip query to the server.  
You should use Fl::event_x_root() and Fl::event_y_root() if possible, but this is necessary if you are not sure if a 
mouse event has been processed recently (such as to position your first window). If the display is not open, this will 
open it.

32.3.2.40  handle()

int Fl::handle (  
    int e,  
    Fl_Window * window  ) [static]

Handle events from the window system.  
This is called from the native event dispatch after native events have been converted to FLTK notation. This function 
calls Fl::handle_(int, Fl_Window+) unless the user sets a dispatch function. If a user dispatch function is set, the 
user must make sure that Fl::handle_() is called, or the event will be ignored.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>the event type (Fl::event_number() is not yet set)</td>
</tr>
<tr>
<td>window</td>
<td>the window that caused this event</td>
</tr>
</tbody>
</table>

Returns

0 if the event was not handled

See also

    Fl::add_handler(Fl_Event_Handler)  
    Fl::event_dispatch(Fl_Event_Dispatch)

32.3.2.41  handle_()

int Fl::handle_ (  
    int e,  
    Fl_Window * window  ) [static]

Handle events from the window system.  
This function is called from the native event dispatch, unless the user sets another dispatch function. In that case, 
the user dispatch function must decide when to call Fl::handle_(int, Fl_Window+)  
Callbacks can set FL_REASON_CLOSED and FL_REASON_CANCELLED.
32.3 Events handling functions

### Parameters

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

### Returns

0 if the event was not handled

See also

`Fl::event_dispatch(Fl_Event_Dispatch)`

#### 32.3.2.42 pushed() [1/2]

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

Gets the widget that is being pushed.

See also

`void pushed(Fl_Widget*)`

#### 32.3.2.43 pushed() [2/2]

void Fl::pushed (Fl_Widget * o) [static]

Sets the widget that is being pushed.

FL_DRAG or FL_RELEASE (and any more FL_PUSH) events will be sent to this widget.

If you change the pushed widget, the previous one and all parents (that don’t contain the new widget) are sent
FL_RELEASE events. Changing this does not send FL_PUSH to this or any widget, because sending FL_PUSH is
supposed to test if the widget wants the mouse (by it returning non-zero from handle()).

#### 32.3.2.44 remove_handler()

void Fl::remove_handler (Fl_Event_Handler ha) [static]

Removes a previously added event handler.

See also

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

#### 32.3.2.45 remove_system_handler()

void Fl::remove_system_handler (Fl_System_Handler ha) [static]

Removes a previously added system event handler.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ha</code></td>
<td>The event handler function to remove</td>
</tr>
</tbody>
</table>
32.3.2.46  testShortcut()

```c
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.

32.3.3  Variable Documentation

32.3.3.1  fl_callback_reason_names

```c
const char* const fl_callback_reason_names[]
```

Initial value:

```c
= 
    ["FL_REASON_UNKNOWN",
     "FL_REASON_SELECTED",
     "FL_REASON_DESELECTED",
     "FL_REASON_REUSESELECTED",
     "FL_REASON_OPENED",
     "FL_REASON_CLOSED",
     "FL_REASON_DRAGGED",
     "FL_REASON_CANCELLED",
     "FL_REASON_CHANGED",
     "FL_REASON_GOT_FOCUS",
     "FL_REASON_LOST_FOCUS",
     "FL_REASON_RELEASED",
    ]
```

This is an array of callback reason names you can use to convert font numbers into names.
The array gets defined inline wherever your `#include <FL/names.h>` appears.

32.3.3.2  fl_eventnames

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

32.3.3.3  fl_fontnames

```c
const char* const fl_fontnames[]
```

Initial value:

```c
= ["FL_HELVETICA",
    "FL_HELVETICA_BOLD",
    "FL_HELVETICA_ITALIC",
    "FL_HELVETICA_BOLD_ITALIC",
    "FL_COURIER",
    "FL_COURIER_BOLD",
    "FL_COURIER_ITALIC",
```

See also

Fl::add_system_handler(Fl_System_Handler)
"FL_COURIER_BOLD_ITALIC",
"FL_TIMES",
"FL_TIMES_BOLD",
"FL_TIMES_ITALIC",
"FL_TIMES_BOLD_ITALIC",
"FL_SYMBOL",
"FL_SCREEN",
"FL_SCREEN_BOLD",
"FL_ZAPF_DINGBATS",
}

This is an array of font names you can use to convert font numbers into names. The array gets defined inline wherever your '#include <FL/names.h>' appears.

Example:

```c
#include <FL/names.h> // array will be defined here
int MyClass::my_callback(Fl_Widget *w, void*) {
  int fnum = w->labelfont();
  // Resulting output might be e.g. "Label's font is FL_HELVETICA (0)"
  printf("Label's font is %s (%d)\n", fl_fontnames[fnum], fnum);
  // ...resulting output might be e.g. "Label's font is FL_HELVETICA (0)"...
  [..]
}
```

### 32.4 Selection & Clipboard functions

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

#### Functions

- **static void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data=0)**
  
  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.

- **static int Fl::clipboard_contains (const char *type)**
  
  Returns non 0 if the clipboard contains data matching `type`.

- **static void Fl::copy (const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text)**
  
  Copies the data pointed to by `stuff` to the selection buffer (`destination` is 0), the clipboard (`destination` is 1), or both (`destination` is 2).

- **static int Fl::dnd ()**
  
  Initiate a Drag And Drop operation.

- **static void Fl::paste (Fl_Widget &receiver)**
  
  Backward compatibility only.

- **static void Fl::paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)**
  
  Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.

- **static void Fl::remove_clipboard_notify (Fl_Clipboard_Notify_Handler h)**
  
  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.

- **static void Fl::selection (Fl_Widget &owner, const char *, int len)**
  
  Changes the current selection.

- **static Fl_Widget * Fl::selection_owner ()**
  
  Back-compatibility only: Gets the widget owning the current selection

- **static void Fl::selection_owner (Fl_Widget *)**
  
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.

- **static int Fl::selection_to_clipboard ()**
  
  Copies the current selection_to_clipboard mode.

- **static void Fl::selection_to_clipboard (int mode)**
  
  Copies selections on X11 directly to the clipboard if enabled.

#### Variables

- **static char const *const Fl::clipboard_image = "image"**
  
  Denotes image data.

- **static char const *const Fl::clipboard_plain_text = "text/plain"**
  
  Denotes plain textual data.
32.4.1 Detailed Description

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

32.4.2 Function Documentation

32.4.2.1 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");  
}  
```

```c
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.

32.4.2.2 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`.

32.4.2.3 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. Copies data to the selection buffer, the clipboard, or both.

The `destination` can be:

- 0: selection buffer (see note below)
- 1: clipboard
The selection buffer exists only on the X11 platform and is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations. On all other platforms the selection buffer (\texttt{destination = 0}) is mapped to the clipboard, i.e. on platforms other than X11 all \texttt{destinations} are equivalent and the data is always copied to the clipboard.

\textbf{Note}

Please see \texttt{Fl::section_to_clipboard()} to enable duplication of the selection buffer to the clipboard on X11, i.e. if \texttt{destination = 0} (selection buffer) and \texttt{Fl::selection_to_clipboard()} is enabled, then the data is copied to both the selection buffer and the clipboard. This makes the X11 behavior similar to other platforms but keeps the selection buffer for X11 specific inter process communication.

\texttt{type} should always be \texttt{Fl::clipboard_plain_text} which is the default. Other values are ignored and reserved for future extensions.

\textbf{Note}

This function is, at present, intended only to copy UTF-8 encoded textual data. To copy graphical data, use the \texttt{Fl_Copy_Surface} class. The \texttt{type} argument may allow to copy other kinds of data in the future.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{in \ stuff}</td>
<td>text data to be copied</td>
</tr>
<tr>
<td>\texttt{in \ len}</td>
<td>the number of relevant bytes in \texttt{stuff}</td>
</tr>
<tr>
<td>\texttt{in \ destination}</td>
<td>0 = selection, 1 = clipboard, 2 = both (see description)</td>
</tr>
<tr>
<td>\texttt{in \ type}</td>
<td>usually plain text (see description)</td>
</tr>
</tbody>
</table>

#### 32.4.2.4 \texttt{dnd()}

\begin{verbatim}
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 \texttt{text}.
Create a selection first using: \texttt{Fl::copy(const char \*stuff, int len, 0)}
\end{verbatim}

#### 32.4.2.5 \texttt{paste()} [1/2]

\begin{verbatim}
void Fl::paste (Fl_Widget & receiver) [static]
Backward compatibility only.
This calls \texttt{Fl::paste(receiver, 0)};
See also
\texttt{Fl::paste(Fl_Widget &receiver, int clipboard, const char* type)}
\end{verbatim}

#### 32.4.2.6 \texttt{paste()} [2/2]

\begin{verbatim}
void Fl::paste (Fl_Widget & receiver,
int source,
const char * type = Fl::clipboard_plain_text ) [static]
Pastes the data from the selection buffer (\texttt{source} is 0) or the clipboard (\texttt{source} is 1) into \texttt{receiver}.
The selection buffer (\texttt{source} is 0) is used for middle-mouse pastes and for drag-and-drop selections. The clipboard (\texttt{source} is 1) is used for copy/cut/paste operations.
\end{verbatim}
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. If 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.

### 32.4.2.7 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.

### 32.4.2.8 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*)
```

### 32.4.2.9 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.
32.4.2.10  selection_to_clipboard() [1/2]

static int Fl::selection_to_clipboard ( ) [inline], [static]
Returns the current selection_to_clipboard mode.

See also
void selection_to_clipboard(int)

32.4.2.11  selection_to_clipboard() [2/2]

static void Fl::selection_to_clipboard ( int mode ) [inline], [static]
Copies selections on X11 directly to the clipboard if enabled.
This method can be called on all platforms. Other platforms than X11 are not affected by this feature.
If this is switched on (mode = 1), Fl::copy() copies all data to the clipboard regardless of its destination argument. If the destination is 0 (selection buffer) data is copied to both the selection buffer and the clipboard.
Drag and drop is also affected since drag-and-drop data is copied to the selection buffer.
You can use this to make the experience of data selection and copying more like that on other platforms (Windows, macOS, and even Wayland).
The default operation mode is the standard X11 behavior (disabled).

Note
This feature is experimental and enabling it may have unexpected side effects. It is your own responsibility if you enable it.

Since
1.4.0

Parameters

| in | mode | 1 = enable selection_to_clipboard, 0 = disable selection_to_clipboard |

See also
copy(const char *, int, int, const char *)

32.5  Screen functions

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

Functions

- static int Fl::h ()
  Returns the height in pixels of the main screen work area.
- static void Fl::keyboard_screen_scaling (int value)
  Controls the possibility to scale all windows by ctrl/+-/0/ or cmd/+-/0/.
- static int Fl::screen_count ()
  Gets the total count of available screens.
- static void Fl::screen_dpi (float &h, float &v, int n=0)
  Gets the screen resolution in dots-per-inch for the given screen.
- static int Fl::screen_num (int x, int y)
  Gets the screen number of a screen that contains the specified screen position x, y.
- static int Fl::screen_num (int x, int y, int w, int h)
Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.

• static float Fl::screen_scale (int n)
  Current value of the GUI scaling factor for screen number n (n [0, Fl::screen_count()-1])
• static void Fl::screen_scale (int n, float factor)
  Sets the value of the GUI scaling factor for screen number n (n [0, Fl::screen_count()-1]).
• static int Fl::screen_scaling_supported()
  See if scaling factors are supported by this platform.

• static void Fl::screen_work_area (int &X, int &Y, int &W, int &H)
  Gets the bounding box of the work area of the screen that contains the mouse pointer.
• static void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)
  Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.
• static void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int n)
  Gets the bounding box of the work area of the given screen.

• static void Fl::screen_xywh (int &X, int &Y, int &W, int &H)
  Gets the bounding box of a screen that contains the mouse pointer.
• static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my)
  Gets the bounding box of a screen that contains the specified screen position mx, my.
• static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)
  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by mx, my, mw, mh.
• static void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int n)
  Gets the screen bounding rect for the given screen.

• static int Fl::w()
  Returns the width in pixels of the main screen work area.
• static int Fl::x()
  Returns the leftmost x coordinate of the main screen work area.
• static int Fl::y()
  Returns the topmost y coordinate of the main screen work area.

32.5.1 Detailed Description

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

FLTK supports high-DPI screens using a screen scaling factor. The scaling factor is initialized by the library to a value based on information obtained from the OS. If this initial value is not satisfactory, the FLTK_SCALING_FACTOR environment variable can be set to a value FLTK will multiply to the OS-given value. The 2 variants of functions Fl::screen_scale() allow to programmatically get and set scaling factor values. The scaling factor value can be further changed at runtime by typing ctrl/+/-/0/ (cmd/+/-/0/ under macOS). FLTK sends the FL_ZOOM_EVENT when the factor value is changed, to which a callback can be associated with Fl::add_handler(). By default, FLTK displays the new scaling factor value in a yellow, transient window. This can be changed with option Fl::OPTION_SHOW_SCALING.

32.5.2 Function Documentation

32.5.2.1 keyboard_screen_scaling()

void Fl::keyboard_screen_scaling (int value) [static]

Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
This function should be called before fl_open_display() runs. If it is not called, the default is to handle these keys for window scaling.

Note

This function can currently only be used to switch the internal handler off, i.e. value must be 0 (zero) - all other values result in undefined behavior and are reserved for future extension.
32.5 Screen functions

Parameters

| value | 0 to stop recognition of ctrl/+/-/0/ (or cmd/+/-/0/ under macOS) keys as window scaling. |

32.5.2.2 screen_count()

```c
int Fl::screen_count ( ) [static]
```

Gets the total count of available screens.

**Note**

Screen numbers range from 0 to `Fl::screen_count()-1` in the FLTK API.

32.5.2.3 screen_dpi()

```c
void Fl::screen_dpi ( 
    float & h, 
    float & v, 
    int n = 0 ) [static]
```

Gets the screen resolution in dots-per-inch for the given screen.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>h,v</th>
<th>horizontal and vertical resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the screen number (0 to <code>Fl::screen_count() - 1</code>)</td>
</tr>
</tbody>
</table>

**See also**

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

32.5.2.4 screen_num() [1/2]

```c
int Fl::screen_num ( 
    int x, 
    int y ) [static]
```

Gets the screen number of a screen that contains the specified screen position `x`, `y`.

**Parameters**

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

**Returns**

a screen number `[0 , Fl::screen_count()-1]`

**Attention**

When the running system contains screens with different scaling factor values, this API may become ambiguous because a given value pair `(x, y)` may belong to distinct screens. In that situation, other APIs should be preferred, e.g., `Fl_Window::screen_num()` and `Fl::screen_scale(int)`.

Generated by Doxygen
32.5.2.5  **screen_num()**  [2/2]

```c
int Fl::screen_num (  
    int x,  
    int y,  
    int w,  
    int h ) [static]
```

Gets the screen number for the screen which intersects the most with the rectangle defined by \( x, y, w, h \).

**Parameters**

- \( x, y, w, h \) the rectangle to search for intersection with

**Returns**

- a screen number \([0, \text{Fl::screen_count}()-1]\)

32.5.2.6  **screen_scale()**

```c
void Fl::screen_scale (  
    int n,  
    float factor ) [static]
```

Sets the value of the GUI scaling factor for screen number \( n \) \([0, \text{Fl::screen_count}()-1]\).

Also sets the scale factor value of all windows mapped to screen number \( n \), if any.

32.5.2.7  **screen_scaling_supported()**

```c
int Fl::screen_scaling_supported ( ) [static]
```

See if scaling factors are supported by this platform.

**Returns**

- 0 if scaling factors are not supported by this platform, 1 if a single scaling factor value is shared by all screens, 2 if each screen can have its own scaling factor value.

**See also**

Fl::screen_scale(int)

32.5.2.8  **screen_work_area()**  [1/3]

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

Gets the bounding box of the work area of the screen that contains the mouse pointer.

**Parameters**

- \( X, Y, W, H \) the work area bounding box

**See also**

void  

screen_work_area(int &x, int &y, int &w, int &h, int mx, int my)
32.5 Screen functions

32.5.2.9  screen_work_area() [2/3]

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

Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
</tr>
<tr>
<td>in</td>
<td>mx, my</td>
</tr>
</tbody>
</table>

32.5.2.10  screen_work_area() [3/3]

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
</tbody>
</table>

See also

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

Note

Like all quantities accessible via public APIs of FLTK, values of X, Y, W, H are given in FLTK units, that is, in drawing units divided by the scaling factor of screen n.

32.5.2.11  screen_xywh() [1/4]

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

Gets the bounding box of a screen that contains the mouse pointer.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>X, Y, W, H</td>
</tr>
</tbody>
</table>
See also

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

### 32.5.2.12 screen_xywh() [2/4]

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

Gets the bounding box of a screen that contains the specified screen position `mx, my`.

**Parameters**

<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</td>
<td>the absolute screen position</td>
</tr>
</tbody>
</table>

### 32.5.2.13 screen_xywh() [3/4]

```c
void Fl::screen_xywh (  
    int & X,  
    int & Y,  
    int & W,  
    int & H,  
    int mx,  
    int my,  
    int mw,  
    int mh )  [static]
```

Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by `mx, my, mw, mh`.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>X,Y,W,H</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mx,my,mw,mh</td>
<td>the rectangle to search for intersection with</td>
</tr>
</tbody>
</table>

See also

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

### 32.5.2.14 screen_xywh() [4/4]

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

Gets the screen bounding rect for the given screen.

Under Windows, Mac OS X, and X11 + the Gnome desktop, screen #0 contains the menubar/taskbar.
### Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>(X, Y, W, H)</th>
<th>the corresponding screen bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>(n)</td>
<td>the screen number (0 to Fl::screen_count() - 1)</td>
</tr>
</tbody>
</table>

**Note**

Like all quantities accessible via public APIs of FLTK, values of \(X, Y, W, H\) are given in FLTK units, that is, in drawing units divided by the scaling factor of screen \(n\).

See also

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

### 32.6 Color & Font functions

**fl global color, font functions.**

#### Functions

- **Fl_Color fl_color ()**
  
  Return the last `fl_color()` that was set.

- **void fl_color (Fl_Color c)**
  
  Set the color for all subsequent drawing operations.

- **void fl_color (int c)**
  
  For back compatibility - use `fl_color(Fl_Color c)` instead

- **void fl_color (uchar r, uchar g, uchar b)**
  
  Set the color for all subsequent drawing operations.

- **Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)**
  
  Returns the weighted average color between the two given colors.

- **Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context)**
  
  Returns a color that contrasts with the background color.

- **void fl_contrast_function (Fl_Contrast_Function *f)**
  
  Register a custom contrast function.

- **int fl_contrast_level ()**
  
  Get the contrast level (sensitivity) of the `fl_contrast()` method.

- **void fl_contrast_level (int level)**
  
  Set the contrast level (sensitivity) of the `fl_contrast()` method.

- **int fl_contrast_mode ()**
  
  Return the current contrast algorithm (mode).

- **void fl_contrast_mode (int mode)**
  
  Set the contrast algorithm (mode).

- **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()`.

- **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.

- **int fl_height (int font, int 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.
- **const char * fl_latin1_to_local (const char *t, int n=-1)**
  - Convert text from Windows/X11 latin1 character set to local encoding.
- **double fl_lightness (FL_Color color)**
  - Return the perceived lightness of a color.
- **const char * fl_local_to_latin1 (const char *t, int n=-1)**
  - Convert text from local encoding to Windows/X11 latin1 character set.
- **const char * fl_local_to_mac_roman (const char *t, int n=-1)**
  - Convert text from local encoding to Mac Roman character set.
- **double fl_luminance (FL_Color color)**
  - Return the raw / physical luminance of a color.
- **const char * fl_mac_roman_to_local (const char *t, int n=-1)**
  - Convert text from Mac Roman character set to local encoding.
- **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().
- **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 font.
- **void fl_text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h)**
  - Determine the minimum pixel dimensions of a sequence of n characters (bytes) using the current fl_font().
- **double fl_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 (Fl_Font)**
  - Gets the string for this face.
- **static const char * Fl::get_font_name (Fl_Font, int *attributes=0)**
  - Get a human-readable string describing the family of this face.
- **static int Fl::get_font_sizes (Fl_Font, int *&sizep)**
  - Return an array of sizes in sizep.
- **static void Fl::set_color (Fl_Color i, 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.
32.6 Color & Font functions

- 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.

32.6.1 Detailed Description

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

32.6.2 Function Documentation

32.6.2.1 fl_color() [1/3]

```c
Fl_Color fl_color () [inline]
```

Return the last fl_color() that was set.
This can be used for state save/restore.

32.6.2.2 fl_color() [2/3]

```c
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

32.6.2.3 fl_color() [3/3]

```c
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

32.6.2.4 fl_color_average()

```c
Fl_Color fl_color_average ( Fl_Color color1,
                           Fl_Color color2,
                           float weight )
```

Generated by Doxygen
Returns the weighted average color between the two given colors.

The red, green and blue values are averages using the following formula:

\[
\text{color} = \text{color1} \times \text{weight} + \text{color2} \times (1 - \text{weight})
\]

Thus, a \text{weight} value of 1.0 will return the first color, while a value of 0.0 will return the second color.

Parameters

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

### 32.6.2.5 \text{fl\_contrast()}

\text{Fl\_Color} \text{fl\_contrast} (\newline \quad \text{Fl\_Color} fg, \newline \quad \text{Fl\_Color} bg, \newline \quad \text{Fl\_Fontsize} fs, \newline \quad \text{int} context )

Returns a color that contrasts with the background color. This will be the foreground color if it contrasts sufficiently with the background color. Otherwise, returns \text{FL\_WHITE} or \text{FL\_BLACK} depending on which color provides the best contrast.

FLTK 1.4.0 uses a different default contrast function than earlier releases (1.3.x) but you can use the old "legacy" contrast function by calling

\text{fl\_contrast\_mode} (\text{FL\_CONTRAST\_LEGACY});

early in your main program.

Note

It is a known issue that static initialization using \text{fl\_contrast()} may already have been executed before you call this function in \text{main()}. You should be aware of this and, if necessary, write your own (static) contrast initialization function. This should rarely be necessary.

You can change the behavior of \text{fl\_contrast()} in several ways:

- Change the "level" (sensitivity) for contrast calculation, see \text{fl\_contrast\_level()}. Valid levels are 0 - 100, the default "medium" value is 50. If you raise the level above 50 the overall contrast will generally be higher, i.e. the required contrast to return the foreground color is raised and therefore the calculated color will switch "earlier" to either black or white. In other words, using the following values:

  - 0 will always use the foreground color
  - 50 will use the default, unmodified algorithm
  - 100 will always use black or white

- values larger than 50 may yield slightly better results. Changing the level is particularly useful and intended for the "legacy mode" to improve the results partially. Values slightly above 50 (50 - 70) will likely return the best results (50 is the default, as used in FLTK 1.3.x).

Note

Different contrast modes (algorithms) can use their own values and defaults of \text{fl\_contrast\_level()}.\n
- Change the used contrast calculation function. You can either use the old (FLTK 1.3.x) function or use the better but slower function based on the CIETLAB (L+\text{a}+\text{b}+) color model, or you can define your own custom contrast function if you need even better contrast results.

The following contrast functions are available:

- \text{FL\_CONTRAST\_LEGACY}, the old FLTK 1.3.x compatible function. This is the fastest function (using integer arithmetic) but it provides worse results in border cases. You may want to use this on embedded or otherwise CPU constrained systems or if you need strict backwards compatibility. For slightly better results you may utilize the new \text{fl\_contrast\_level(int)} function (since 1.4.0) to increase the contrast sensitivity. This will provide slightly better results than FLTK 1.3.x and earlier but we recommend to use the new default function:
• **FL_CONTRAST_CIELAB**, based on the CIELAB (L*a*b*) color model. This function is superior regarding the visual contrast perception but may be slower. **This is the default since FLTK 1.4.0.**

• **FL_CONTRAST_CUSTOM**, your own contrast calculation function.

In the future we may provide even more (and superior) contrast algorithms. The new parameters *fs* and *context* (since 1.4.0) are defined for future extensions and are currently not used. Default values are 0.

**Note**

These new optional parameters must be provided in the custom contrast function which is the reason why they are added now. In the future we may use the fontsize to adjust the calculated contrast, and users defining their own contrast functions may use them in their functions.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fg</th>
<th>foreground (text) color</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bg</td>
<td>background color</td>
</tr>
<tr>
<td>in</td>
<td>fs</td>
<td>font size (optional, default = 0 == undefined)</td>
</tr>
<tr>
<td>in</td>
<td>context</td>
<td>graphical context (optional, default = 0 == text)</td>
</tr>
</tbody>
</table>

**Returns**

contrasting color: *fg*, FL_BLACK, or FL_WHITE

See also

- `fl_contrast_level(int)`
- `fl_contrast_mode(int)`
- `fl_contrast_function()`

### 32.6.2.6 fl_contrast_function()

```c
void fl_contrast_function ( Fl_Contrast_Function * f )
```

Register a custom contrast function.

Your custom contrast function will be called when `fl_contrast()` is called if and only if you previously registered your function and called `fl_contrast_mode(FL_CONTRAST_CUSTOM)`.

Your custom contrast function must provide the signature

`Fl_Color my_contrast_function(Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context)`

The arguments are the same as for the full `fl_contrast()` function since FLTK 1.4. You can use the supplied fontsize *fs* to modify the result. Depending on the caller the *fs* parameter can be 0 (default) or a valid fontsize. The *context* parameter is not yet used and will always be 0 unless included in a call to `fl_contrast()`. The value 0 should be interpreted as text. In the future the *context* argument will be used to supply a different context than text (small icons, large icons, etc.). The exact usage is not yet specified.

Your function may also use `fl_contrast_level()` to modify the result accordingly.

Since

1.4.0

See also

- `fl_contrast_mode(int)`
- `fl_contrast_level(int)`
- `fl_contrast()`
### 32.6.2.7 `fl_contrast_level()` [1/2]

```c
int fl_contrast_level();
```

Get the contrast level (sensitivity) of the `fl_contrast()` method. This returns the level of the currently selected contrast mode.

**Returns**

- The current contrast level.

**See also**

- `fl_contrast_level(int level)`
- `fl_contrast_mode(int mode)`

**Since**

1.4.0

### 32.6.2.8 `fl_contrast_level()` [2/2]

```c
void fl_contrast_level(int level);
```

Set the contrast level (sensitivity) of the `fl_contrast()` method. This can be used to tune the legacy `fl_contrast()` function to achieve slightly better results. The default value is defined per contrast mode (see below). Values between 50 and 70 are recommended but you can raise it up to 100. Lower values than 50 are probably not useful.

The contrast level affects not only the legacy (1.3.x) `fl_contrast()` function but also the new CIELAB contrast mode which is the default since FLTK 1.4.0.

Other contrast modes are currently not affected by the contrast level. You may use the contrast level if you define your own custom contrast function in mode `FL_CONTRAST_CUSTOM`.

**Note**

All contrast modes store their own contrast level because the behavior is slightly different. You must change the contrast mode `fl_contrast_mode()` before you set or get the contrast level.

The default contrast level is:

- 50 in mode `FL_CONTRAST_LEGACY` (compatible with FLTK 1.3.x)
- 55 in mode `FL_CONTRAST_CIELAB`
- 0 (undefined) for all other modes

See the description of `fl_contrast_mode(int mode)` for more information about the contrast level per mode.

**Example:**

```c
fl_contrast_mode(FL_CONTRAST_LEGACY);
fl_contrast_level(65);
```

A level greater than 50 (probably best in the range 50 to 70) may achieve better results of the legacy `fl_contrast()` function in some border cases of low contrast between foreground and background colors but we recommend to use the new default algorithm `FL_CONTRAST_CIELAB` unless you need strict backwards compatibility or use a CPU constrained embedded system.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td><code>level</code></td>
<td>valid range is 0 to 100</td>
</tr>
</tbody>
</table>
32.6 Color & Font functions

Since

1.4.0

32.6.2.9  fl_contrast_mode() [1/2]

int fl_contrast_mode ( )
Return the current contrast algorithm (mode).

Returns

Contrast algorithm (mode).

Since

1.4.0

See also

fl_contrast_mode(int)

32.6.2.10  fl_contrast_mode() [2/2]

void fl_contrast_mode ( int mode )
Set the contrast algorithm (mode).

You can use one of

• FL_CONTRAST_NONE (not recommended: returns the foreground color)
• FL_CONTRAST_LEGACY (same as in FLTK 1.3.x)
• FL_CONTRAST_CIELAB (default since FLTK 1.4.0)
• FL_CONTRAST_CUSTOM (you must define your own contrast algorithm)

If you set FL_CONTRAST_CUSTOM you must also register your custom contrast function by calling fl_contrast_function().

You may set the contrast level fl_contrast_level(int) after setting the contrast mode. This affects the contrast algorithm as described below:

• FL_CONTRAST_LEGACY: default level is 50 which is compatible with FLTK 1.3.x and older. This mode is no longer the default and is not recommended because it doesn't take human contrast perception into account and doesn't properly handle sRGB color values. You may get better contrasts if you set the level higher than 50. Values in the range 50 to 70 may be useful. Higher values result in higher contrast, i.e. the algorithm switches "earlier" to black or white mode.

• FL_CONTRAST_CIELAB: default level is 55 which appears to be a good value. The higher the level is, the more contrast is to be expected. Values in the range below 55 accept lower contrast and values above 55 switch "earlier" to black or white. Values between 45 and 65 may yield usable contrast experience.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mode</th>
<th>if invalid, FL_CONTRAST_CIELAB will be selected</th>
</tr>
</thead>
</table>
Since 1.4.0

See also

fl_contrast_function(Fl_Contrast_Function *)
fl_contrast_level(int)

32.6.2.11 fl_font() [1/2]

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.

32.6.2.12 fl_font() [2/2]

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.

32.6.2.13 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().

32.6.2.14 fl_height() [2/2]

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.

32.6.2.15 fl_latin1_to_local()

const char * fl_latin1_to_local ( const char * t, int n = -1 )
Convert text from Windows/X11 latin1 character set to local encoding.
32.6 Color & Font functions

### Parameters

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

**Returns**

pointer to internal buffer containing converted characters

#### 32.6.2.16 fl_lightness()

```c
double fl_lightness ( Fl_Color color )
```

Return the perceived lightness of a color. This function calculates the perceived lightness of Fl_Color color. The returned lightness value $L_{star}$ according to the CIELAB ($L^*a^*b^*$) color model is almost linear with respect to human perception. It is in the range 0 (black) to 100 (white). The result values of two colors can be compared directly and the difference is their perceived contrast.

**Parameters**

| in | color | Fl_Color value |

**Returns**

perceived lightness (0 .. 100)

Since 1.4.0

#### 32.6.2.17 fl_local_to_latin1()

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

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

**Parameters**

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

**Returns**

pointer to internal buffer containing converted characters

#### 32.6.2.18 fl_local_to_mac_roman()

```c
const char * fl_local_to_mac_roman ( const char * t, int n = -1 )
```
Convert text from local encoding to Mac Roman character set.
32.6 Color & Font functions

Parameters

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

Returns

pointer to internal buffer containing converted characters

32.6.2.19 fl_luminance()

double fl_luminance ( Fl_Color color )

Return the raw / physical luminance of a color.
This function calculates the physical luminance of Fl_Color color.
The returned luminance value (aka $Y$) is the physical luminance of the Fl_Color color.
The result is in the range 0.0 (black) to 1.0 (white).

Note

This is probably not what you want if you are interested in perceived contrast or lightness calculation because the luminance $Y$ is not linear with respect to human perception.

See fl_lightness(Fl_Color) for a function that returns the perceived lightness of a color which can be used directly for contrast calculation.

Parameters

| in  | color | FI_Color value |

Returns

Raw (physical) luminance (0.0 .. 1.0)

Since

1.4.0

See also

fl_lightness(Fl_Color)

32.6.2.20 fl_mac_roman_to_local()

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

Convert text from Mac Roman character set to local encoding.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>character string (Mac Roman encoding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>optional number of characters to convert (default is all)</td>
</tr>
</tbody>
</table>
Returns
pointer to internal buffer containing converted characters

32.6.2.21  fl_show_colormap()

\texttt{Fl\_Color fl\_show\_colormap ( Fl\_Color oldcol )}

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

![Figure 32.1 fl_show_colormap](image)

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
\textit{in} & \textit{oldcol} & color to be highlighted when grid is shown. \\
\hline
\end{tabular}

\textbf{Return values}

\begin{tabular}{|c|c|}
\hline
\textit{Fl\_Color} & value of the chosen colormap entry. \\
\hline
\end{tabular}

\textbf{See also}

\texttt{Fl\_Color\_Chooser}

32.6.2.22  fl_size()

\texttt{Fl\_Fontsize fl\_size ( ) [inline]}

Return the size set by the most recent call to \texttt{fl\_font()}.
This can be used to save/restore the font.
32.6.2.23 `fl_text_extents()` [1/2]

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

Determine the minimum pixel dimensions of a null-terminated string using the current `fl_font()`.

Usage: given a string "txt" drawn using `fl_draw(txt, x, y)` you would determine its pixel extents on the display using `fl_text_extents(txt, dx, dy, wo, ho)` such that a bounding box that exactly fits around the text could be drawn with `fl_rect(x+dx, y+dy, wo, ho)`. Note the dx, dy values hold the offset of the first "colored in" pixel of the string, from the draw origin.

Note the desired font and font size must be set with `fl_font()` before calling this function. This differs slightly from `fl_measure()` in that the dx/dy values are also returned.

No FLTK symbol expansion will be performed.

Example use:

```c
int dx, dy, W, H;
fl_font(FR_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);
```

32.6.2.24 `fl_text_extents()` [2/2]

```c
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

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

32.6.2.25 `fl_width()`

```c
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.

32.6.2.26 `free_color()`

```c
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.

---

Generated by Doxygen
32.6.27  get_color() [1/3]

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.

32.6.28  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)

32.6.29  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)

32.6.30  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.

32.6.31  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.
32.6.2.32  get_font_sizes()

```c
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.

32.6.2.33  set_color() [1/3]

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

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

32.6.2.34  set_color() [2/3]

```c
void Fl::set_color (  
    Fl_Color i,  
    uchar red,  
    uchar green,  
    uchar blue ) [static]
```

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

32.6.2.35  set_color() [3/3]

```c
void Fl::set_color (  
    Fl_Color i,  
    uchar red,  
    uchar green,  
    uchar blue,  
    uchar alpha ) [static]
```

Sets an entry in the `fl_color` index table.
You can set it to any 8-bit RGBA color.

**Note**

The color transparency is effective under the Wayland, hybrid Wayland/X11 and macOS platforms, whereas it has no effect under the X11 and Windows platforms. It's also effective for widgets added to an `Fl_Gl_Window`.

**Version**

1.4

32.6.2.36  set_font()

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

Changes a face.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fnum</code></td>
<td>The font number to be assigned a new face</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Name of the font to assign. The string pointer is simply stored, the string is not copied, so the string must be in static memory. The exact name to be used depends on the platform:</td>
</tr>
</tbody>
</table>

- Windows, X11, Xft: use the family name prefixed by one character to indicate the desired font variant. Characters ' ', 'I', 'B', 'P' denote plain, italic, bold, and bold-italic variants, respectively. For example, string "IGabriola" is to be used to denote the "Gabriola italic" font. The "Oblique" suffix, in whatever case, is to be treated as "italic", that is, prefix the family name with 'I'.

- Other platforms, i.e., X11 + Pango, Wayland, macOS: use the full font name as returned by function `Fl::get_font_name()` or as listed by applications test/fonts or test/utf8. No prefix is to be added.

### 32.6.2.37  `set_fonts()`

```c
Fl_Font Fl::set_fonts (const char * xstarname = 0) [static]
```

FLTK will open the display, and add every fonts on the server to the face table. It will attempt to put “families” of faces together, so that the normal one is first, followed by bold, italic, and bold italic. The only argument to this function is somewhat obsolete since FLTK and most underlying operating systems move to support Unicode. For completeness, following is the original documentation and a few updates:

On X11, the optional argument is a string to describe the set of fonts to add. Passing NULL will select only fonts that have the ISO8859-1 character set (and are thus usable by normal text). Passing "-*" will select all fonts with any encoding as long as they have normal X font names with dashes in them. Passing "*" will list every font that exists (on X this may produce some strange output). Other values may be useful but are system dependent.

With the Xft option on Linux, this parameter is ignored.

With Windows, NULL selects fonts with ANSI_CHARSET encoding and non-NULL selects all fonts.

On macOS, this parameter is ignored.

The return value is how many faces are in the table after this is done.

### 32.7  Drawing functions

FLTK global graphics and GUI drawing functions.

#### Enumerations

```c
enum {
    FL_SOLID = 0 , FL_DASH = 1 , FL_DOT = 2 , FL_DASHDOT = 3 ,
    FL_DASHDOTDOT = 4 , FL_CAP_FLAT = 0x100 , FL_CAP_ROUND = 0x200 , FL_CAP_SQUARE = 0x300 ,
    FL_JOIN_MITER = 0x1000 , FL_JOIN_ROUND = 0x2000 , FL_JOIN_BEVEL = 0x3000
}
```

#### Functions

- `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.
32.7 Drawing functions

- void fl_begin_complex_polygon ()
  Start drawing a complex filled polygon.

- void fl_begin_line ()
  Start drawing a list of lines.

- void fl_begin_loop ()
  Start drawing a closed sequence of lines.

- void fl_begin_offscreen (Fl_Offscreen ctx)
  Send all subsequent drawing commands to this offscreen buffer.

- void fl_begin_points ()
  Start drawing a list of points.

- void fl_begin_polygon ()
  Start drawing a convex filled polygon.

- char fl_can_do_alpha_blending ()
  Check whether platform supports true alpha blending for RGBA images.

- Fl_RGB_Image * fl_capture_window (Fl_Window * win, int x, int y, int w, int h)
  Captures the content of a rectangular zone of a mapped window.

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

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

- void fl_clip (int x, int y, int w, int h)
  Intersect the current clip region with a rectangle and push this new region onto the stack (deprecated).

- int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)
  Intersect a rectangle with the current clip region and return the bounding box of the result.

- Fl_Region fl_clip_region ()
  Return the current clipping region.

- void fl_clip_region (Fl_Region r)
  Replace the top of the clipping stack with a clipping region of any shape.

- void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)
  Copy a rectangular area of the given offscreen buffer into the current drawing destination.

- Fl_Offscreen fl_create_offscreen (int w, int h)
  Creation of an offscreen graphics buffer.

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

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

- void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)
  Add a series of points on a Bézier curve to the path.

- void fl_delete_offscreen (Fl_Offscreen ctx)
  Deletion of an offscreen graphics buffer.

- void fl_draw (const char *str, int n, int x, int y)
  Draws starting at the given x, y location a UTF-8 string of length n bytes.

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

- void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)
  Fancy string drawing function which is used to draw all the labels.

- void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, void(*callthis)(const char *, int, int), Fl_Image *img=0, int draw_symbols=1)
  The same as fl_draw(const char*,int,int,Fl_Align,Fl_Fmage*,int) with the addition of the callthis parameter, which is a pointer to a text drawing function such as fl_draw(const char*, int, int, int, Fl_Image *img=0, int draw_symbols=1) to do the real work.

- void fl_draw (int angle, const char *str, int n, int x, int y)
  Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.
void fl_draw (int angle, const char *str, int x, int y)
  Draw a null-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)
  Draw an "arrow like" GUI element for the selected scheme.

void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)
  Draws a box using given type, position, size and color.

void fl_draw_check (Fl_Rect bb, Fl_Color col)
  Draw a check mark inside the given bounding box.

void fl_draw_circle (int x0, int y0, int d, Fl_Color color)
  Draw a potentially small, filled circle as a GUI element.

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.

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

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

void fl_end_complex_polygon ()
  End complex filled polygon, and draw.

void fl_end_line ()
  End list of lines, and draw.

void fl_end_loop ()
  End closed sequence of lines, and draw.

void fl_end_offscreen ()
  Quit sending drawing commands to the current offscreen buffer.

void fl_end_points ()
  End list of points, and draw.

void fl_end_polygon ()
  End convex filled polygon, and draw.

const char * fl_expand_text (const char *from, char *buf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)
  Copy from to buf, replacing control characters with ^X.

void fl_focus_rect (int x, int y, int w, int h)
  Draw a dotted rectangle, used to indicate keyboard focus on a widget.

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

void fl_frame2 (const char *s, int x, int y, int w, int h)
  Draws a series of line segments around the given box.

void fl_gap ()
  Separate loops of the path.

void fl_line (int x, int y, int x1, int y1)
  Draw a line from (x,y) to (x1,y1)

void fl_line (int x, int y, int x1, int y1, int x2, int y2)
  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)
32.7 Drawing functions

- void **fl_line_style** (int style, int width=0, char *dashes=0)  
  Set how to draw lines (the "pen").

- void **fl_load_identity** ()  
  Set the transformation matrix to identity.

- void **fl_load_matrix** (double a, double b, double c, double d, double x, double y)  
  Set the current transformation matrix.

- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2)  
  Outline a 3-sided polygon with lines.

- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)  
  Outline a 4-sided polygon with lines.

- void **fl_measure** (const char *str, int &x, int &y, int draw_symbols=1)  
  Measure how wide and tall the string will be when printed by the **fl_draw()** function with **align** parameter.

- int **fl_measure_pixmap** (char *const *data, int &w, int &h)  
  Get the dimensions of a pixmap.

- int **fl_measure_pixmap** (const char *const *cdata, int &w, int &h)  
  Get the dimensions of a pixmap.

- void **fl_mult_matrix** (double a, double b, double c, double d, double x, double y)  
  Concatenate another transformation onto the current one.

- int **fl_not_clipped** (int x, int y, int w, int h)  
  Does the rectangle intersect the current clip region?

- unsigned int **fl_old_shortcut** (const char *s)  
  Emulation of XForms named shortcuts.

- void **fl_overlay_clear** ()  
  Erase a selection rectangle without drawing a new one.

- void **fl_overlay_rect** (int x, int y, int w, int h)  
  Draw a transient dotted selection rectangle.

- float **fl_override_scale** ()  
  Removes any GUI scaling factor in subsequent drawing operations.

- void **fl_pie** (int x, int y, int w, int h, double a1, double a2)  
  Draw filled ellipse sections using integer coordinates.

- void **fl_point** (int x, int y)  
  Draw a single pixel at the given coordinates.

- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2)  
  Fill a 3-sided polygon.

- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)  
  Fill a 4-sided polygon.

- void **fl_pop_clip** ()  
  Restore the previous clip region.

- void **fl_pop_matrix** ()  
  Restore the current transformation matrix from the stack.

- void **fl_push_clip** (int x, int y, int w, int h)  
  Intersect the current clip region with a rectangle and push this new region onto the stack.

- void **fl_push_matrix** ()  
  Save the current transformation matrix on the stack.

- void **fl_push_no_clip** ()  
  Push an empty clip region onto the stack so nothing will be clipped.

- uchar * **fl_read_image** (uchar *p, int X, int Y, int W, int H, int alpha=0)  
  Reads an RGB(A) image from the current window or off-screen buffer.

- void **fl_rect** (Fl_Rect r)  
  Draw a 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.

- **void** `fl_rescale_offscreen (Fl_Offscreen &ctx)`
  - Adapts an offscreen buffer to a changed value of the scale factor.

- **void** `fl_restore_clip ()`
  - Undo any clobbering of the clip region done by your program.

- **void** `fl_restore_scale (float s)`
  - Restores the GUI scaling factor and the clipping region in subsequent drawing operations.

- **void** `fl_rotate (double d)`
  - Concatenate rotation transformation onto the current one.

- **void** `fl_rounded_rect (int x, int y, int w, int h, int r)`
  - Draw a 1-pixel rounded border inside the given bounding box.

- **void** `fl_rounded_rectf (int x, int y, int w, int h, int r)`
  - Color with current color a rounded rectangle that exactly fills the given bounding box.

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

- **void** `fl_scale (double x)`
  - Concatenate scaling transformation onto the current one.

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

- **void** `fl_scroll (int X, int Y, int W, int H, int dx, int dy, void (*draw_area)(void *, int, int, int), void *data)`
  - Scroll a rectangle and draw the newly exposed portions.

- **void** `fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)`
  - Inform text input methods about the current text insertion cursor.

- **void** `fl_set_status (int X, int Y, int W, int H)`
  - Related to text input methods under X11.

- **const char * flShortcutLabel (unsigned int shortcut)**
  - Get a human-readable string from a shortcut value.

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

- **double flTransform Dx (double x, double y)**
  - Transform distance using current transformation matrix.

- **double flTransform Dy (double x, double y)**
  - Transform distance using current transformation matrix.

- **double flTransform X (double x, double y)**
  - Transform coordinate using the current transformation matrix.
32.7 Drawing functions

- 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).

32.7.1 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.

32.7.2 Enumeration Type Documentation

32.7.2.1 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>
### 32.7.3 Function Documentation

#### 32.7.3.1 fl_add_symbol()

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

Adds a symbol to the system.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>name</code> name of symbol (without the &quot;@&quot;)</td>
</tr>
<tr>
<td>in</td>
<td><code>drawit</code> function to draw symbol</td>
</tr>
<tr>
<td>in</td>
<td><code>scalable</code> set to 1 if <code>drawit</code> uses scalable vector drawing</td>
</tr>
</tbody>
</table>

**Returns**

1 on success, 0 on failure

#### 32.7.3.2 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.

#### 32.7.3.3 fl_arc() [1/2]

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

Add a series of points to the current path on the arc of a circle. You can get elliptical paths by using scale and rotate before calling `fl_arc()`.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>x, y, r</code> center and radius of circular arc</td>
</tr>
<tr>
<td>in</td>
<td><code>start, end</code> angles of start and end of arc measured in degrees counter-clockwise from 3 o’clock. If <code>end</code> is less than <code>start</code> then it draws the arc in a clockwise direction.</td>
</tr>
</tbody>
</table>
32.7 Drawing functions

Examples:

```c
// Draw an arc of points
fl_begin_points();
fl_arc(100.0, 100.0, 50.0, 0.0, 180.0);
fl_end_points();
// Draw arc with a line
fl_begin_line();
fl_arc(200.0, 100.0, 50.0, 0.0, 180.0);
fl_end_line();
// Draw filled arc
fl_begin_polygon();
fl_arc(300.0, 100.0, 50.0, 0.0, 180.0);
fl_end_polygon();
```

32.7.3.4 `fl_arc()` [2/2]

```c
void fl_arc ( int x,
             int y,
             int w,
             int h,
             double a1,
             double a2 ) [inline]
```

Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and 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, `a2` must be greater or equal to `a1`.

`fl_arc()` draws a series of lines to approximate the arc. Notice that the integer version of `fl_arc()` has a different number of arguments than the double version `fl_arc(double x, double y, double r, double start, double end)`

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>bounding box of complete circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>a1,a2</td>
<td>start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. <code>a2</code> must be greater than or equal to <code>a1</code>.</td>
</tr>
</tbody>
</table>
32.7.3.5 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 `fl_gap()` to separate loops of the path. To outline the polygon, use `fl_begin_loop()` and replace each `fl_gap()` with `fl_end_loop();fl_begin_loop()` pairs.

**Note**

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

32.7.3.6 fl_begin_offscreen()

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

Send all subsequent drawing commands to this offscreen buffer.

**Parameters**

- `ctx` the offscreen buffer.

**Note**

The `ctx` argument must have been created by `fl_create_offscreen()`.

32.7.3.7 fl_begin_points()

```c
void fl_begin_points ( ) \[inline\]
```

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

32.7.3.8 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
32.7 Drawing functions

32.7.3.9 fl_capture_window()

```c
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><code>win</code></th>
<th>a mapped Fl_Window (derived types including Fl_Gl_Window are also possible)</th>
</tr>
</thead>
<tbody>
<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

32.7.3.10 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**

| `x,y,r` | center and radius of circle |

**Note**

fl_circle() is best used as part of the Drawing Complex Shapes API, that is, flanked by fl_begin_XXX() and fl_end_XXX() calls where XXX can be ‘loop’ or ‘polygon’ to draw, respectively a circle or a disk. Transformation functions (e.g., fl_scale(double, double)) can be also used for fl_circle() to draw empty of filled ellipses. It must be the only thing in the path: if you want a circle as part of a complex polygon you must use fl_arc(). Nevertheless, fl_circle() can also be used by itself to draw circles.

32.7.3.11 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.

### 32.7.3.12 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 |

Returns

Non-zero if the resulting rectangle is different to the original.

See also

`fl_not_clipped()`

### 32.7.3.13 fl_clip_region() [1/2]

```c
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.

### 32.7.3.14 fl_clip_region() [2/2]

```c
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**

| in | r | clipping region |

### 32.7.3.15 fl_copy_offscreen()

```c
void fl_copy_offscreen ( int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy ) [inline]
```
Copy a rectangular area of the given offscreen buffer into the current drawing destination.

**Parameters**

| x,y | position where to draw the copied rectangle |
| w,h | size of the copied rectangle |
| pixmap | offscreen buffer containing the rectangle to copy |
| srcx,srcy | origin in offscreen buffer of rectangle to copy |

### 32.7.3.16 fl_create_offscreen()

```c
Fl_Offscreen fl_create_offscreen ( int w, int h )
```
Creation of an offscreen graphics buffer.

**Parameters**

| w,h | width and height in FLTK units of the buffer. |
Returns

the created graphics buffer.

The pixel size of the created graphics buffer is equal to the number of pixels in an area of the screen containing the current window sized at \( w, h \) FLTK units. This pixel size varies with the value of the scale factor of this screen.

Note

Work with the \texttt{fl	extunderscore XXX	extunderscore offscreen()} functions is equivalent to work with an \texttt{Fl	extunderscore Image	extunderscore Surface} object, as follows:

<table>
<thead>
<tr>
<th>Fl	extunderscore Offscreen-based approach</th>
<th>Fl	extunderscore Image	extunderscore Surface-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Fl	extunderscore Offscreen off = fl	extunderscore create	extunderscore offscreen(w, h)}</td>
<td>\texttt{Fl	extunderscore Image	extunderscore Surface \ast surface = new Fl	extunderscore Image	extunderscore Surface(w, h, 1)}</td>
</tr>
<tr>
<td>\texttt{fl	extunderscore begin	extunderscore offscreen(off)}</td>
<td>\texttt{Fl	extunderscore Surface	extunderscore Device::push	extunderscore current(surface)}</td>
</tr>
<tr>
<td>\texttt{fl	extunderscore end	extunderscore offscreen()}</td>
<td>\texttt{Fl	extunderscore Surface	extunderscore Device::pop	extunderscore current()}</td>
</tr>
<tr>
<td>\texttt{fl	extunderscore copy	extunderscore offscreen(x,y,w,h, off, sx,sy)}</td>
<td>\texttt{fl	extunderscore copy	extunderscore offscreen(x,y,w,h, surface\rightarrow offscreen(), sx,sy)}</td>
</tr>
<tr>
<td>\texttt{fl	extunderscore rescale	extunderscore offscreen(off)}</td>
<td>\texttt{surface\rightarrow rescale()}</td>
</tr>
<tr>
<td>\texttt{fl	extunderscore delete	extunderscore offscreen(off)}</td>
<td>\texttt{delete surface}</td>
</tr>
</tbody>
</table>

32.7.3.17 \texttt{fl	extunderscore cursor()}

\begin{verbatim}
void fl_cursor ( 
    Fl_Cursor c )
\end{verbatim}

Sets the cursor for the current window to the specified shape and colors.

The cursors are defined in the \texttt{<FL/Enumerations.H>} header file.

32.7.3.18 \texttt{fl	extunderscore curve()}

\begin{verbatim}
void fl_curve ( 
    double X0, 
    double Y0, 
    double X1, 
    double Y1, 
    double X2, 
    double Y2, 
    double X3, 
    double Y3 ) [inline]
\end{verbatim}

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

| \texttt{in \ast X0,Y0} | curve start point |
| \texttt{in \ast X1,Y1} | curve control point |
| \texttt{in \ast X2,Y2} | curve control point |
| \texttt{in \ast X3,Y3} | curve end point |

32.7.3.19 \texttt{fl	extunderscore delete	extunderscore offscreen()}

\begin{verbatim}
void fl_delete_offscreen ( 
    Fl	extunderscore Offscreen ctx )
\end{verbatim}

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().

32.7.3.20 fl_draw() [1/4]

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

Draw a nul-terminated UTF-8 string starting at the given x, y location. Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract fl_descent() from y. To align to the top, subtract fl_descent() and add fl_height(). This version of fl_draw provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

32.7.3.21 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.

32.7.3.22 fl_draw() [3/4]

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

Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

Note

When using X11 (Unix, Linux, Cygwin et al.) this needs Xft to work. Under plain X11 (w/o Xft) rotated text is not supported by FLTK. A warning will be issued to stderr at runtime (only once) if you use this method with an angle other than 0.

32.7.3.23 fl_draw() [4/4]

```c
void fl_draw (int angle,
```

Generated by Doxygen
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 counterclockwise.
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.

32.7.3.24 fl_draw_arrow()

void fl_draw_arrow (  
    Fl_Rect r,
    Fl_Arrow_Type t,
    Fl_Orientation o,
    Fl_Color col )

Draw an "arrow like" GUI element for the selected scheme.
In the future this function should be integrated in Fl_Scheme as a virtual method, i.e. it would call a method like ...
Fl_Scheme::current()->draw_arrow(r, t, o, col);

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>r</th>
<th>bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
<td>arrow type</td>
</tr>
<tr>
<td>in</td>
<td>o</td>
<td>orientation</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>arrow color</td>
</tr>
</tbody>
</table>

32.7.3.25 fl_draw_box()

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>

32.7.3.26 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 |
|    | col       | Fl_Color to draw the check mark         |

Since

1.4.0

32.7.3.27  fl_draw_circle()

```c
void fl_draw_circle (  
  int x0,  
  int y0,  
  int d,  
  Fl_Color color )
```

Draw a potentially small, filled circle as a GUI element.

This method draws a filled circle, using fl_pie() if the given diameter \(d\) is larger than 6 pixels (aka FLTK units).

If \(d\) is 6 or smaller it approximates a filled circle by drawing several filled rectangles, depending on the size because fl_pie() might not draw well on many systems for smaller sizes.

Parameters

| in | x0,y0   | coordinates of top left of the bounding box |
|    | d       | diameter == width and height of the bounding box |
| in | color   | drawing color                           |

32.7.3.28  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

| in | buf  | points at the "r" data of the top-left pixel. Color data must be in r, g, b order. Luminance data is only one gray byte. |
|    | X,Y  | position where to put top-left corner of image |
| in | W,H  | size of the image |
| in | D    | 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 |
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.

### 32.7.3.29 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

```c
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.

### 32.7.3.30 fl_draw_image_mono() [1/2]

```c
void fl_draw_image_mono ( 
```
32.7 Drawing functions

const uchar * buf,
int X,
int Y,
int W,
int H,
int D = 1,
int L = 0) [inline]

Draw a gray-scale (1 channel) image.

See also

fl_draw_image(const uchar * buf, int X, int Y, int W, int H, int D, int L)

32.7.3.31 fl_draw_image_mono() [2/2]

void fl_draw_image_mono(
   Fl_Draw_Image_Cb cb,
   void * data,
   int X,
   int Y,
   int W,
   int H,
   int D = 1) [inline]

Draw a gray-scale image using a callback function to generate image data.

See also

fl_draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D)

32.7.3.32 fl_draw_pixmap() [1/2]

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

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

32.7.3.33 fl_draw_pixmap() [2/2]

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

Draw XPM image data, with the top-left corner at the given position.
The image is dithered on 8-bit displays so you won’t lose color space for programs displaying both images and pixmaps.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>data</th>
<th>pointer to XPM image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y</td>
<td>position of top-left corner</td>
</tr>
<tr>
<td>in</td>
<td>bg</td>
<td>background color</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

0 if there was any error decoding the XPM data.

32.7.3.34  fl_draw_symbol()

int fl_draw_symbol (  
    const char * label,  
    int x,  
    int y,  
    int w,  
    int h,  
    Fl_Color col )

Draw the named symbol in the given rectangle using the given color.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>label</th>
<th>name of symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x, y</td>
<td>position of symbol</td>
</tr>
<tr>
<td>in</td>
<td>w, h</td>
<td>size of symbol</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>color of symbol</td>
</tr>
</tbody>
</table>

Returns

1 on success, 0 on failure

32.7.3.35  fl_expand_text()

const char * fl_expand_text (  
    const char * from,  
    char * buf,  
    int maxbuf,  
    double maxw,  
    int & n,  
    double & width,  
    int wrap,  
    int draw_symbols )

Copy from to buf, replacing control characters with ^X. Stop at a newline or if maxbuf characters written to buffer. Also word-wrap if width exceeds maxw. Returns a pointer to the start of the next line of characters. Sets n to the number of characters put into the buffer. Sets width to the width of the string in the current font.

32.7.3.36  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.

32.7.3.37  fl_frame()

void fl_frame (  

const char *s,
    int x,
    int y,
    int w,
    int h )

Draws a series of line segments around the given box.
The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: top, left, bottom, right. The result of calling `fl_frame()` with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and `fl_frame2()` is the order of the line segments.

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>

32.7.3.38 fl_frame2()

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

Draws a series of line segments around the given box.
The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: bottom, right, top, left. The result of calling `fl_frame2()` with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and `fl_frame()` is the order of the line segments.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>sets of 4 grayscale values in bottom, right, top, left order</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>position and size</td>
</tr>
</tbody>
</table>

32.7.3.39 fl_gap()

```c
void fl_gap ( ) [inline]
```

Separate loops of the path.
It is unnecessary but harmless to call `fl_gap()` before the first vertex, after the last vertex, or several times in a row.

32.7.3.40 fl_line_style()

```c
void fl_line_style ( 
    int style,
    int width = 0,
    char *dashes = 0 ) [inline]
```

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

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

Note

Because of how line styles are implemented on Win32 systems, you must set the line style after setting the drawing color. If you set the color after the line style you will lose the line style settings.

The dashes array does not work under the (unsupported!) operating systems Windows 95, 98 or Me, since those operating systems do not support complex line styles.

32.7.3.41 fl_load_matrix()

```c
void fl_load_matrix ( double a, double b, double c, double d, double x, double y ) [inline]
```

Set the current transformation matrix.

Parameters

| in  | a,b,c,d,x,y   | transformation matrix elements |

32.7.3.42 fl_measure()

```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() function with align parameter. If the incoming w is non-zero it will wrap to that width.

The current font is used to do the width/height calculations, so unless its value is known at the time fl_measure() is called, it is advised to first set the current font with fl_font(). With event-driven GUI programming you can never be sure which widget was exposed and redrawn last, nor which font it used. If you have not called fl_font() explicitly in your own code, the width and height may be set to unexpected values, even zero!

**Note:** In the general use case, it’s a common error to forget to set w to 0 before calling fl_measure() when wrap behavior isn’t needed.

Parameters

| in  | str          | nul-terminated string |
| out | w,h          | width and height of string in current font |
Parameters

| in | draw_symbols | non-zero to enable @symbol handling [default=1] |

// Example: Common use case for fl_measure()
const char *s = "This is a test";
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

32.7.3.43  fl_measure_pixmap() [1/2]

int fl_measure_pixmap (
    char const * data,
    int & w,
    int & h )
Get the dimensions of a pixmap.
An XPM image contains the dimensions in its data. This function returns the width and height.

Parameters

| in   | data | pointer to XPM image data. |
| out  | w,h  | width and height of image |

Returns

non-zero if the dimensions were parsed OK
0 if there were any problems

32.7.3.44  fl_measure_pixmap() [2/2]

int fl_measure_pixmap (  
    const char *const * cdata,
    int & w,
    int & h )
Get the dimensions of a pixmap.

See also

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

32.7.3.45  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 |

Generated by Doxygen
### 32.7.3.46 fl_not_clipped()

```c
int fl_not_clipped (  
    int x,  
    int y,  
    int w,  
    int h )  [inline]
```

Does the rectangle intersect the current clip region?

#### Parameters

- **in** x,y,w,h position and size of rectangle

#### Returns

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

#### 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()

---

### 32.7.3.47 fl_old_shortcut()

```c
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_ALT|FL_CTRL|FL_META|‘c’
- "%c" -- Same as FL_COMMAND|‘c’ (see FL_COMMAND for platform specific behavior)
- "0x63" -- Same as "c" (hex 63==‘c’)
- "99" -- Same as "c" (dec 99==‘c’)
- "0143" -- Same as "c" (octal 0143==‘c’)
- "^0x63" -- Same as (FL_CTRL|‘c’), or (FL_CTRL|0x63)
- "^99" -- Same as (FL_CTRL|‘c’), or (FL_CTRL|99)
- "^0143" -- Same as (FL_CTRL|‘c’), or (FL_CTRL|0143)
Remarks

Due to XForms legacy, there are some odd things to consider when using the modifier characters.

(1) You can use the special modifier keys for chords only if the modifiers are provided in this order: #, +, ^, !, @. Other ordering can yield undefined results.

So for instance, Ctrl-Alt-c must be specified as "^c" (and not "c"), due to the above ordering rule.

(2) If you want to make a shortcut that uses one of the special modifier characters (as the character being modified), then to avoid confusion, specify the numeric equivalent, e.g.

If you want... Then use...

' #' as the shortcut... "0x23" (instead of just ").
' +' as the shortcut... "0x2b" (instead of just "+").
'^' as the shortcut... "0x5e" (instead of "^").
Alt-+ as the shortcut... "0x2b" (instead of "+").
Alt-^ as the shortcut... "0x5e" (instead of ".").
..etc..

As a general rule that's easy to remember, unless the shortcut key to be modified is a single alpha-numeric character [A-Z,a-z,0-9), it's probably best to use the numeric equivalents.

Don't fix these silly legacy issues in a future release. Nobody is using this anymore.

32.7.3.48 fl_overlay_clear()

void fl_overlay_clear ( )
Erase a selection rectangle without drawing a new one.

See also

fl_overlay_rect(int x, int y, int w, int h)

32.7.3.49 fl_overlay_rect()

void fl_overlay_rect ( int x, int y, int w, int h )
Draw a transient dotted selection rectangle.
This function saves the current screen content and then draws a dotted selection rectangle into the front screen buffer. If another selection rectangle was drawn earlier, the previous screen graphics are restored first.
To clear the selection rectangle, call fl_overlay_clear().

The typical (and only) use for this function is to draw a selection rectangle during a mouse drag event sequence without having to redraw the entire content of the widget.
Your event handle should look similar to this (also see test/mandelbrot.cxx):

```c
int MyWidget::handle(int event) { 
  static int ix, iy;
  switch (event) {
    case FL_PUSH: 
      ix = Fl::event_x(); iy = Fl::event_y();
      return 1;
    case FL_DRAG: 
      this->make_current();
      fl_overlay_rect(ix, iy, ix-Fl::event_x(), iy-Fl::event_y());
      return 1;
    case FL_RELEASE: 
      this->make_current();
      fl_overlay_clear();
      // select the element under the rectangle 
      return 1;
  }
  return MySuperWidget::handle(event);
}
```
Note

Between drawing an overlay rect and clearing it, the content of the widget must not change. `fl_overlay_rect()` and `fl_overlay_clear()` should be called when the actual event occurs, and not within `Widget::draw()`. `fl_overlay_rect()` and `fl_overlay_clear()` should not be mixed with `Fl_Overlay_Window`. `Fl_Overlay_Window` provides an entirely different way of drawing selection outlines and is not limited to rectangles.

Parameters

| x, y, w, h | position and size of the overlay rectangle. |

See also

`fl_overlay_clear()`

32.7.3.50  `fl_override_scale()`

```c
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.

32.7.3.51  `fl_pie()`

```c
void fl_pie ( int x, int y, int w, int h, double a1, double a2 ) [inline]
```

Draw filled ellipse sections using integer coordinates. Like `fl_arc()`, but `fl_pie()` draws a filled-in pie slice. This slice may extend outside the line drawn by `fl_arc()`; to avoid this use `w - 1` and `h - 1`.

Parameters

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

| in   | a1, a2     | start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. a2 must be greater than or equal to a1. |

Generated by Doxygen
32.7 Drawing functions

32.7.3.52 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.

32.7.3.53 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.

32.7.3.54 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.

32.7.3.55 fl_push_clip()

```c
void fl_push_clip (  
    int x,  
    int y,  
    int w,  
    int h  
) [inline]
```

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

Parameters

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

Figure 32.4 fl_pie() and fl_arc()
32.7.3.56  fl_push_matrix()

void fl_push_matrix ( ) [inline]
Save the current transformation matrix on the stack.
The maximum depth of the stack is 32.

32.7.3.57  fl_read_image()

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

| in | p       | pixel buffer, or NULL to allocate one |
|    | X, Y    | position of top-left of image to read |
|    | w, h    | width and height of image to read    |
|    | alpha   | alpha value for image (0 for none)  |

Returns

pointer to pixel buffer, or NULL if allocation failed.

The p argument points to a buffer that can hold the image and must be at least w*h*3 bytes when reading RGB images, or w*h*4 bytes when reading RGBA images. If NULL, fl_read_image() will create an array of the proper size which can be freed using delete[].
The alpha parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.

See also

fl_capture_window()

32.7.3.58  fl_rect() [1/2]

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.

32.7.3.59  fl_rect() [2/2]

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.
32.7.3.60  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)

32.7.3.61  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.

32.7.3.62  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

32.7.3.63  fl_reset_spot()

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.

32.7.3.64  fl_restore_scale()

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(). |

### 32.7.3.65 fl_rotate()

```c
void fl_rotate (double d) [inline]
```

Concatenate rotation transformation onto the current one.

Parameters

| in d | - rotation angle, counter-clockwise in degrees (not radians) |

### 32.7.3.66 fl_rounded_rect()

```c
void fl_rounded_rect (int x, int y, int w, int h, int r) [inline]
```

Draw a 1-pixel rounded border inside the given bounding box. The radius code is optimized for speed and works best for values between 5 and 15 units.

### 32.7.3.67 fl_rounded_rectf()

```c
void fl_rounded_rectf (int x, int y, int w, int h, int r) [inline]
```

Color with current color a rounded rectangle that exactly fills the given bounding box. The radius code is optimized for speed and works best for values between 5 and 15 units.

### 32.7.3.68 fl_scale() [1/2]

```c
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 |

### 32.7.3.69 fl_scale() [2/2]

```c
void fl_scale (double x, double y) [inline]
```

Concatenate scaling transformation onto the current one.
32.7 Drawing functions

Parameters

| in | $x,y$ | scale factors in x-direction and y-direction |

32.7.3.70 fl_scroll()

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

Scroll a rectangle and draw the newly exposed portions.

Parameters

| in | $X,Y$ | position of top-left of rectangle |
| in | $W,H$ | size of rectangle |
| in | $dx,dy$ | pixel offsets for shifting rectangle |
| in | draw_area | callback function to draw rectangular areas |
| in | data | 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. |

32.7.3.71 fl_set_spot()

```c
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. |

32.7.3.72 fl_set_status()

```c
void fl_set_status ( )
```
int X,
int Y,
int W,
int H)

Related to text input methods under X11.
This function is presently used only by the utf8 test application and only for the X11 platform. This function is
apparently not indispensable for text input to work correctly as suggested by other apps that don't use it (e.g.,
editor).

32.7.3.73 fl_shortcut_label() [1/2]

const char * fl_shortcut_label (  
  unsigned int shortcut  )

Get a human-readable string from a shortcut value.
Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This
only works if the shortcut is a character key or a numbered function key. If the shortcut is zero then an empty string
is returned. The return value points at a static buffer that is overwritten with each call.

Since
FLTK 1.3.4 modifier key names can be localized, but key names can not yet be localized. This may be added
to a future FLTK version.

Modifier key names (human-readable shortcut names) can be defined with the following global const char * pointer
variables:

• fl_local_ctrl => name of FL_CTRL
• fl_local_alt => name of FL_ALT
• fl_local_shift => name of FL_SHIFT
• fl_local_meta => name of FL_META

fl_local_ctrl = "Strg";       // German for "Ctrl"
fl_local_shift = "Umschalt"; // German for "Shift"

Note
Due to random static initialization order this should always be done from code in main() or called by main()
as opposed to static initialization since the default strings in the FLTK library are set by static initializers.
Otherwise this might result in the wrong order so FLTK's internal initialization overwrites your strings.

The shortcut name will be constructed by adding all modifier names in the order defined above plus the name of the
key. A '+' character is added to each modifier name unless it has a trailing '" or a trailing '+'.
Example:
Ctrl+Alt+Shift+Meta+F12

The default values for modifier key names are as given above for all platforms except macOS. macOS uses graphical
characters that represent the typical macOS modifier names in menus, e.g. cloverleaf, saucepan, etc. You may,
however, redefine macOS modifier names as well.

Parameters

| in | shortcut | the integer value containing the ASCII character or extended keystroke plus modifiers |

Returns

a pointer to a static buffer containing human readable text for the shortcut

32.7.3.74 fl_shortcut_label() [2/2]

const char * fl_shortcut_label (  

Generated by Doxygen
unsigned int shortcut,
const char ** eom )

Get a human-readable string from a shortcut value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in shortcut</td>
<td>the integer value containing the ASCII character or extended keystroke plus modifiers</td>
</tr>
<tr>
<td>in eom</td>
<td>if this pointer is set, it will receive a pointer to the end of the modifier text</td>
</tr>
</tbody>
</table>

Returns

a pointer to a static buffer containing human readable text for the shortcut

See also

fl_shortcut_label(unsigned int shortcut)

32.7.3.75 fl_transform_dx()

double fl_transform_dx (  
  double x, 
  double y )  [inline]

Transform distance using current transformation matrix.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x, y</td>
<td>coordinate</td>
</tr>
</tbody>
</table>

32.7.3.76 fl_transform_dy()

double fl_transform_dy (  
  double x, 
  double y )  [inline]

Transform distance using current transformation matrix.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x, y</td>
<td>coordinate</td>
</tr>
</tbody>
</table>

32.7.3.77 fl_transform_x()

double fl_transform_x (  
  double x, 
  double y )  [inline]

Transform coordinate using the current transformation matrix.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x, y</td>
<td>coordinate</td>
</tr>
</tbody>
</table>
32.7.3.78  fl_transform_y()

double fl_transform_y ( double x, double y ) [inline]
Transform coordinate using the current transformation matrix.

Parameters

$\textbf{in} \ x, y \ \text{coordinate}$

32.7.3.79  fl_transformed_vertex()

void fl_transformed_vertex ( double xf, double yf ) [inline]
Add coordinate pair to the vertex list without further transformations.

Parameters

$\textbf{in} \ xf, yf \ \text{transformed coordinate}$

32.7.3.80  fl_translate()

void fl_translate ( double x, double y ) [inline]
Concatenate translation transformation onto the current one.

Parameters

$\textbf{in} \ x, y \ \text{translation factor in x-direction and y-direction}$

32.7.3.81  fl_vertex()

void fl_vertex ( double x, double y ) [inline]
Add a single vertex to the current path.

Parameters

$\textbf{in} \ x, y \ \text{coordinate}$

32.8  Multithreading support functions

fl multithreading support functions declared in $\texttt{<FL/Fl.H>}$
32.8 Multithreading support functions

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.

32.8.1 Detailed Description

flt multithreading support functions declared in `<FL/Fl.H>`

32.8.2 Function Documentation

32.8.2.1 awake() [1/2]

```c
int Fl::awake ( Fl_Awake_Handler func,
                void * data = 0 ) [static]
```

See void awake(void *message=0).

Let the main thread know an update is pending and have it call a specific function.

Registers a function that will be called by the main thread during the next message handling cycle. Returns 0 if the callback function was registered, and -1 if registration failed. Over a thousand awake callbacks can be registered simultaneously.

See also

`Fl::awake(void *message=0)`

32.8.2.2 awake() [2/2]

```c
void Fl::awake ( void * msg = 0 ) [static]
```

Sends a message pointer to the main thread, causing any pending Fl::wait() call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

Multiple calls to Fl::awake() will queue multiple pointers for the main thread to process, up to a system-defined (typically several thousand) depth. The default message handler saves the last message which can be accessed using the Fl::thread_message() function.

In the context of a threaded application, a call to Fl::awake() with no argument will trigger event loop handling in the main thread. Since it is not possible to call Fl::flush() from a subsidiary thread, Fl::awake() is the best (and only, really) substitute.

It's not necessary to wrap calls to any form of Fl::awake() by Fl::lock() and Fl::unlock(). Nevertheless, the early, single call to Fl::lock() used to initialize threading support is necessary.

Function Fl::awake() in all its forms is typically called by worker threads, but it can be used safely by the main thread too, as a means to break the event loop.

See also

`Multithreading`
32.8.2.3 lock()

```cpp
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

32.8.2.4 thread_message()

```cpp
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

32.8.2.5 unlock()

```cpp
void Fl::unlock () [static]
```

The `unlock()` method releases the lock that was set using the `lock()` method. Child threads should call this method as soon as they are finished accessing FLTK.

See also: Multithreading

32.9 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.

32.9.1 Detailed Description

These functions, declared in `<FL/Fl.H>`, support deletion of widgets inside callbacks. `Fl::delete_widget()` should be called when deleting widgets or complete widget trees (Fl_Group, Fl_Window, ...) inside callbacks.

The other functions are intended for internal use. The preferred way to use them is by using the helper class `Fl_Widget_Tracker`.

The following is to show how it works ...

There are three groups of related methods:

1. scheduled widget deletion
32.9 Safe widget deletion support functions

- `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`.

### 32.9.2 Function Documentation

#### 32.9.2.1 clear_widget_pointer()

```cpp
def 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`

#### 32.9.2.2 delete_widget()

```cpp
def 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()

32.9.2.3 do_widget_deletion()

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)

32.9.2.4 release_widget_pointer()

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()

32.9.2.5 watch_widget_pointer()

void Fl::watch_widget_pointer ( Fl_Widget *& w ) [static]

Adds a widget pointer to the widget watch list.

Note

Internal use only, please use class Fl_Widget_Tracker instead.

This can be used, if it is possible that a widget might be deleted during a callback or similar function. The widget pointer must be added to the watch list before calling the callback. After the callback the widget pointer can be queried, if it is NULL. If it is NULL, then the widget has been deleted during the callback and must not be accessed anymore. If the widget pointer is not NULL, then the widget has not been deleted and can be accessed safely.

After accessing the widget, the widget pointer must be released from the watch list by calling Fl::release_widget_pointer().

Example for a button that is clicked (from its handle() method):

```c
Fl_Widget *wp = this; // save 'this' in a pointer variable
Fl::watch_widget_pointer(wp); // add the pointer to the watch list
set_changed(); // set the changed flag
do_callback(); // call the callback
if (!wp) { // the widget has been deleted
    // DO NOT ACCESS THE DELETED WIDGET !
} else {
    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

\texttt{Fl::release_widget_pointer()}
\texttt{Fl::clear_widget_pointer()}

An easier and more convenient method to control widget deletion during callbacks is to use the class \texttt{Fl_Widget_Tracker} with a local (automatic) variable.

See also

\texttt{class Fl_Widget_Tracker}

### 32.10 Cairo Support Functions and Classes

#### Classes

- \texttt{class Fl_Cairo_State}
  
  \textit{Contains all the necessary info on the current cairo context.}
- \texttt{class Fl_Cairo_Window}
  
  \textit{This defines an FLTK window with Cairo support.}

#### Functions

- static bool \texttt{Fl::cairo_autolink_context () [inline], [static]}
  
  \textit{Gets the current autolink mode for Cairo support.}
- static void \texttt{Fl::cairo_autolink_context (bool alink)}
  
  \textit{When \texttt{FLTK_HAVE_CAIRO} is defined and \texttt{cairo_autolink_context()} is true, any current window \texttt{dc} is linked to a current Cairo context.}
- static \texttt{cairo_t * Fl::cairo_cc ()}
  
  \textit{Gets the current Cairo context linked with a fltk window.}
- static void \texttt{Fl::cairo_cc (cairo_t \* c, bool own=false)}
  
  \textit{Sets the current Cairo context to \texttt{c}.}
- static void \texttt{Fl::cairo_flush (cairo_t \* c)}
  
  \textit{Flush Cairo drawings on Cairo context \texttt{c}.}
- static \texttt{cairo_t \* Fl::cairo_make_current (Fl_Window \* w)}
  
  \textit{Provides a Cairo context for window \texttt{w}.}

#### 32.10.1 Detailed Description

#### 32.10.2 Function Documentation

##### 32.10.2.1 cairo_autolink_context() [1/2]

\texttt{static bool Fl::cairo_autolink_context ( ) [inline], [static]}

\textit{Gets the current autolink mode for Cairo support.}

\textbf{Return values}

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{false}</td>
<td>if no Cairo context autolink is made for each window.</td>
</tr>
<tr>
<td>\texttt{true}</td>
<td>if any fltk window is attached a Cairo context when it is current.</td>
</tr>
</tbody>
</table>

See also

\texttt{void cairo_autolink_context(bool alink)}

Generated by Doxygen
32.10.2.2 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 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

32.10.2.3 cairo_cc()

static void Fl::cairo_cc (  
    cairo_t ∗c,  
    bool own = false ) [inline], [static]

Sets the current Cairo context to c. Set own to true if you want fltk to handle this cc deletion.

Note

Only available when configure has the --enable-Cairo option

32.10.2.4 cairo_flush()

static void Fl::cairo_flush (  
    cairo_t ∗c ) [inline], [static]

Flush Cairo drawings on Cairo context c. This is required on Windows if you use the Cairo context provided by the "Cairo autolink" option. Call this when all your drawings on the Cairo context are finished. This is maybe not necessary on other platforms than Windows but it does no harm if you call it always.

You don't need to use this if you use an Fl_Cairo_Window which does this automatically after the draw callback returns.

Code example for "Cairo autolink" mode:

In the overridden draw() method of your subclass of Fl_Window or any widget:

cairo_t ∗cc = Fl::cairo_cc(); // get the "autolink" Cairo context
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc); // flush Cairo drawings to the device

If you configure FLTK with --enable-cairo or CMake option 'OPTION_CAIRO' (i.e. without --enable-cairoext or CMake option 'OPTION_CAIROEXT') or if you don't enable the 'autolink' Cairo context you may do the equivalent to use Cairo drawings in an overridden draw() method of derived classes by using

// get the Cairo context for the \c window
cairo_t ∗cc = Fl::cairo_make_current(window);
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc); // flush Cairo drawings to the device
32.10.2.5 cairo_make_current()

cairo_t * Fl::cairo_make_current (Fl_Window * wi) [static]

Provides a Cairo context for window wi. This is needed in a draw() override if Fl::cairo_autolink_context() returns false, which is the default. The cairo_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 graphical context and the current window didn’t change between two calls, the previous gc is internally kept, thus optimizing the drawing performances. Also, after this call, Fl::cairo_cc() is adequately updated with this Cairo context.

Only available when configure has the --enable-cairo option

Returns

The valid cairo_t *cairo context associated to this window.

Return values

- **NULL** if wi is NULL or maybe with GL windows under Wayland

32.11 Unicode and UTF-8 functions

Fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

**Macros**

- **#define ERRORS_TO_CP1252 1**
  
  Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
- **#define ERRORS_TO_ISO8859_1 1**
  
  Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
- **#define NBC 0xFFFF + 1**
- **#define STRICT_RFC3629 0**
  
  A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.

**Functions**

- **int fl_access (const char *f, int mode)**
  
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- **int fl_chdir (const char *path)**
  
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- **int fl_chmod (const char *f, int mode)**
  
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- **int fl_close_fd (int fd)**

See also

Fl::cairo_autolink_context(bool)
Fl::cairo_make_current(Fl_Window*);
Cross-platform function to close a file descriptor.

- int \textbf{fl_execvp} (const char *file, char *const *argv)

  FILE * \textbf{fl_fopen} (const char *f, const char *mode)

  Cross-platform function to open files with a UTF-8 encoded name.

- char * \textbf{fl_getcwd} (char *buf, int len)

  Cross-platform function to get the current working directory as a UTF-8 encoded value.

- char * \textbf{fl_getenv} (const char *v)

  Cross-platform function to get environment variables with a UTF-8 encoded name or value.

- char * \textbf{fl_make_path} (const char *path)

  Cross-platform function to recursively create a path in the file system.

- void \textbf{fl_make_path_for_file} (const char *path)

  Cross-platform function to create a path for the file in the file system.

- int \textbf{fl_mkdir} (const char *f, int mode)

  Cross-platform function to create a directory with a UTF-8 encoded name.

- unsigned int \textbf{fl_nonspacing} (unsigned int ucs)

  \textbf{Returns true if the Unicode character} ucs \textbf{is non-spacing.}

- int \textbf{fl_open} (const char *fname, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.

- int \textbf{fl_open_ext} (const char *fname, int binary, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.

- int \textbf{fl_putenv} (const char *var)

  Cross-platform function to write environment variables with a UTF-8 encoded name or value.

- int \textbf{fl_rename} (const char *f, const char *n)

  Cross-platform function to rename a filesystem object using UTF-8 encoded names.

- int \textbf{fl_rmdir} (const char *f)

  Cross-platform function to remove a directory with a UTF-8 encoded name.

- int \textbf{fl_stat} (const char *f, struct stat *b)

  Cross-platform function to stat() a file using a UTF-8 encoded name or value.

- int \textbf{fl_system} (const char *cmd)

  Cross-platform function to run a system command with a UTF-8 encoded string.

- int \textbf{fl_tolower} (unsigned int ucs)

  \textbf{Returns the Unicode lower case value of} ucs.

- int \textbf{fl_toupper} (unsigned int ucs)

  \textbf{Returns the Unicode upper case value of} ucs.

- unsigned int \textbf{fl_ucs_to_Utf16} (const unsigned ucs, unsigned short *dst, const unsigned dstlen)

  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.

- int \textbf{fl_unlink} (const char *fname)

  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

- char * \textbf{fl_utf2mbcs} (const char *s)

  Converts UTF-8 string s to a local multi-byte character string.

- unsigned int \textbf{fl_utf8back} (const char *p, const char *start, const char *end)

  Move p backward until it points to the start of a UTF-8 character.

- int \textbf{fl_utf8bytes} (unsigned ucs)

  Return the number of bytes needed to encode the given UCS4 character in UTF-8.

- unsigned \textbf{fl_utf8decode} (const char *p, const char *end, int *len)

  Decode a single UTF-8 encoded character starting at p.

- int \textbf{fl_utf8encode} (unsigned ucs, char *buf)

  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.

- unsigned \textbf{fl_utf8from_mb} (char *dst, unsigned dstlen, const char *src, unsigned srclen)

  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.

- unsigned \textbf{fl_utf8froma} (char *dst, unsigned dstlen, const char *src, unsigned srclen)
32.11 Unicode and UTF-8 functions

Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

• unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)
  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.

• const char * fl_utf8fwd (const char *p, const char *start, const char *end)
  Move p forward until it points to the start of a UTF-8 character.

• int fl_utf8len (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.

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

• int fl_utf8strlen (const char *text, int len)
  Return the length in bytes of a UTF-8 string.

• int fl_utf8test (const char *src, unsigned srclen)
  Examines the first srclen bytes in src and returns a verdict on whether it is UTF-8 or not.

• unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

• unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.

• unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 16-bit characters.

• unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
  Converts a UTF-8 string into a wide character string.

• int fl_utf_nb_char (const unsigned char *buf, int len)
  Returns the number of Unicode chars in the UTF-8 string.

• int fl_utf_strcasecmp (const char *s1, const char *s2)
  UTF-8 aware strcasecmp - converts to Unicode and tests.

• int fl_utf_strncasecmp (const char *s1, const char *s2, int n)
  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

• int fl_utf_tolower (const unsigned char *str, int len, char *buf)
  Converts the string str to its lower case equivalent into buf.

• int fl_utf_toupper (const unsigned char *str, int len, char *buf)
  Converts the string str to its upper case equivalent into buf.

• int fl_wcwidth (const char *src)
  extended wrapper around fl_wcwidth_(unsigned int ucs) function.

• int fl_wcwidth_ (unsigned int ucs)
  Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

32.11.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in `<FL/fl_utf8.h>`

32.11.2 Macro Definition Documentation

32.11.2.1 ERRORS_TO_CP1252

#define ERRORS_TO_CP1252 1
Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
You should also set ERRORS_TO_ISO8859_1. With this a huge amount of more available text (such as all web pages) are correctly converted to Unicode.

Generated by Doxygen
32.11.2.2 ERRORS_TO_ISO8859_1

#define ERRORS_TO_ISO8859_1 1
Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
If this is zero they are instead turned into the Unicode REPLACEMENT CHARACTER, of value 0xfffd. If this is on
fl_utf8decode() will correctly map most (perhaps all) human-readable text that is in ISO-8859-1. This may allow you
to completely ignore character sets in your code because virtually everything is either ISO-8859-1 or UTF-8.

32.11.2.3 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.

32.11.3 Function Documentation

32.11.3.1 fl_access()

int fl_access (const char ∗ f,
    int mode )
Cross-platform function to test a files access() with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard access() function fails with UTF-8
encoded non-ASCII filenames.
Windows defines the mode values 0 for existence, 2 for writable, 4 for readable, and 6 of readable and writable. On
other systems, the modes X_OK, W_OK, and R_OK are usually defined as 1, 2, and 4.
Upon successful completion, the value 0 is returned on all platforms.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>the mode to test</td>
</tr>
</tbody>
</table>

Returns

the return value of _waccess() on Windows or access() on other platforms.

32.11.3.2 fl_chdir()

int fl_chdir (const char ∗ path )
Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
This function is especially useful on the Windows platform where the standard _wchdir() function needs a path in
UTF-16 encoding.
The path is converted to a system specific encoding if necessary and the system specific chdir(converted←
_path) function is called.
The function returns 0 on success and -1 on error. Depending on the platform, errno may be set if an error occurs.

Note

The possible errno values are platform specific. Refer to the documentation of the platform specific chdir() function.

If the function is not implemented on a particular platform the default implementation returns -1 and errno is not
set.
If the path is NULL the function returns -1, but errno is not changed. This is a convenience feature of fl_chdir() as opposed to chdir().
Parameters

| in | path | the target directory for chdir (may be NULL) |

Returns

0 if successful, -1 on error (errno may be set)

### 32.11.3.3 fl_chmod()

```c
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. 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.

### 32.11.3.4 fl_close_fd()

```c
int fl_close_fd (int fd)
```

Cross-platform function to close a file descriptor.

Returns

0 in case of success, or -1 in case of error.

### 32.11.3.5 fl_fopen()

```c
FILE * fl_fopen (const char *f, const char *mode)
```

Cross-platform function to open files with a UTF-8 encoded name. This function is especially useful on the Windows platform where the standard fopen() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

| 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().

32.11.3.6 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>buf</th>
<th>the buffer to populate (may be NULL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>the length of the buffer</td>
</tr>
</tbody>
</table>

Returns

the CWD encoded as UTF-8

32.11.3.7 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

<table>
<thead>
<tr>
<th>in</th>
<th>v</th>
<th>the UTF-8 encoded environment variable</th>
</tr>
</thead>
</table>
Returns

the environment variable in UTF-8 encoding, or NULL in case of error.

32.11.3.8 fl_make_path()

char fl_make_path (const char * path)

Cross-platform function to recursively create a path in the file system.
This function creates a path in the file system by recursively creating all directories.

Parameters

| in | path | a Unix style ("/" forward slashes) absolute or relative pathname |

Returns

1 if the path was created, 0 if creating the path failed at some point.

32.11.3.9 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.

32.11.3.10 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

| in | f | the UTF-8 encoded filename |
| in | mode | the mode of the directory |

Returns

the return value of _wmkdir() on Windows or mkdir() on other platforms.

32.11.3.11 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 (\textasciitilde{}), diaeresis (\textasciicircum{}), or other marks that are added to a base character, for instance ‘a’ (base character) + ‘\textasciitilde{}’ (combining mark) = ‘\textaccentumlaut{a}’ (German Umlaut).

- http://unicode.org/glossary/#base_character
32.11.3.12 fl_open()

```c
int fl_open (    
    const char * fname, 
    int oflags, 
    ... )
```

Cross-platform function to open files with a UTF-8 encoded name. This function is especially useful on the Windows platform where the standard open() function fails with UTF-8 encoded non-ASCII filenames.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>fname</code></th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>oflags</code></td>
<td>other arguments are as in the standard open() function</td>
</tr>
</tbody>
</table>

**Returns**

a file descriptor upon successful completion, or -1 in case of error.

**See also**

`fl_fopen(), fl_open_ext(), fl_close_fd(int fd)`.

32.11.3.13 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><code>fname</code></th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>binary</code></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><code>oflags,...</code></td>
<td>these arguments are as in the standard open() function. Setting <code>oflags</code> to zero opens the file for reading.</td>
</tr>
</tbody>
</table>

**Returns**

a file descriptor upon successful completion, or -1 in case of error.

32.11.3.14 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 \texttt{var} must be encoded in UTF-8 in the form "name=value". The \texttt{name} part must conform to platform dependent restrictions on environment variable names.

The string given in \texttt{var} is copied and optionally converted to the required encoding for the platform. On platforms other than Windows this function calls putenv directly.

The return value is zero on success and non-zero in case of error. The value in case of error is platform specific and returned as-is.

\textbf{Note}

The copied string is allocated on the heap and "lost" on some platforms, i.e. calling \texttt{fl_putenv()} to change environment variables frequently may cause memory leaks. There may be an option to avoid this in a future implementation.

This function is not thread-safe.

\textbf{Parameters}

\begin{verbatim}
in var the UTF-8 encoded environment variable 'name=value'
\end{verbatim}

\textbf{Returns}

0 on success, non-zero in case of error.

### 32.11.3.15 fl_rename()

\begin{verbatim}
int fl_rename ( 
    const char * f, 
    const char * n )
\end{verbatim}

Cross-platform function to rename a filesystem object using UTF-8 encoded names. This function is especially useful on the Windows platform where the standard \_wrename() function expects UTF-16 encoded non-ASCII filenames.

\textbf{Parameters}

\begin{verbatim}
in f the UTF-8 encoded filename to change 
in n the new UTF-8 encoded filename to set
\end{verbatim}

\textbf{Returns}

the return value of \_wrename() on Windows or rename() on other platforms.

### 32.11.3.16 fl_rmdir()

\begin{verbatim}
int fl_rmdir ( 
    const char * f )
\end{verbatim}

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.

\textbf{Parameters}

\begin{verbatim}
in f the UTF-8 encoded filename to remove
\end{verbatim}
32.11 Unicode and UTF-8 functions

Returns

the return value of _wrmdir() on Windows or rmdir() on other platforms.

32.11.3.17 fl_stat()

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

<table>
<thead>
<tr>
<th>in</th>
<th>f</th>
<th>the UTF-8 encoded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
<td>the stat struct to populate</td>
</tr>
</tbody>
</table>

Returns

the return value of _wstat() on Windows or stat() on other platforms.

32.11.3.18 fl_system()

int fl_system {
    const char * cmd
}

Cross-platform function to run a system command with a UTF-8 encoded string.
This function is especially useful on the Windows platform where non-ASCII program (file) names must be encoded as wide characters.
On platforms other than Windows this function calls system() directly.

Parameters

| in | cmd       | the UTF-8 encoded command string |

Returns

the return value of _wsystem() on Windows or system() on other platforms.

32.11.3.19 fl_ucs_to_Utf16()

unsigned fl_ucs_to_Utf16 {
    const unsigned ucs,
    unsigned short * dst,
    const unsigned dstlen
}

Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
These are used by some system calls, especially on Windows.

ucs is the value to convert.
dst points at an array to write, and dstlen is the number of locations in this array. At most dstlen words will be written, and a 0 terminating word will be added if dstlen is large enough. Thus this function will never overwrite the buffer and will attempt return a zero-terminated string if space permits. If dstlen is zero then dst can be set to NULL and no data is written, but the length is returned.
The return value is the number of 16-bit words that would be written to dst if it is large enough, not counting any terminating zero.
If the return value is greater than dstlen it indicates truncation, you should then allocate a new array of size return+1 and call this again.

Unicode characters in the range 0x10000 to 0x10ffff are converted to “surrogate pairs” which take two words each (in UTF-16 encoding). Typically, setting dstlen to 2 will ensure that any valid Unicode value can be converted, and setting dstlen to 3 or more will allow a NULL terminated sequence to be returned.

### 32.11.3.20 fl_unlink()

```c
int fl_unlink (const char * fname )
```

Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename. This function is especially useful on the Windows platform where the standard function expects UTF-16 encoded non-ASCII filenames.

**Parameters**

- `fname` - the filename to unlink

**Returns**

the return value of _wunlink() on Windows or unlink() on other platforms.

### 32.11.3.21 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.

### 32.11.3.22 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**

- `ucs` - UCS4 encoded character

**Returns**

number of bytes required

### 32.11.3.23 fl_utf8decode()

```c
unsigned fl_utf8decode (const char * p, const char * end, int * len )
```
Decode a single UTF-8 encoded character starting at \( p \).
The resulting Unicode value (in the range 0-0x10ffff) is returned, and \( len \) is set to the number of bytes in the UTF-8 encoding (adding \( len \) to \( p \) will point at the next character).

If \( p \) points at an illegal UTF-8 encoding, including one that would go past \( end \), or where a code uses more bytes than necessary, then \( *(\text{unsigned char}*)p \) is translated as though it is in the Microsoft CP1252 character set and \( len \) is set to 1. Treating errors this way allows this to decode almost any ISO-8859-1 or CP1252 text that has been mistakenly placed where UTF-8 is expected, and has proven very useful.

If you want errors to be converted to error characters (as the standards recommend), adding a test to see if the length is unexpectedly 1 will work:

```c
if (*p & 0x80) { // what should be a multibyte encoding
    code = fl_utf8decode(p,end,&len);
    if (len<2) code = 0xFFFD; // Turn errors into REPLACEMENT CHARACTER
    else { // handle the 1-byte UTF-8 encoding:
        code = *p;
        len = 1;
    }
}
```

Direct testing for the 1-byte case (as shown above) will also speed up the scanning of strings where the majority of characters are ASCII.

### 32.11.3.24 `fl_utf8encode()`

```c
int fl_utf8encode {
    unsigned ucs,
    char * buf }
```

Write the UTF-8 encoding of \( ucs \) into \( buf \) and return the number of bytes written.

Up to 4 bytes may be written. If you know that \( ucs \) is less than 0x10000 then at most 3 bytes will be written. If you wish to speed this up, remember that anything less than 0x80 is written as a single byte.

If \( ucs \) is greater than 0x10ffff this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER).

RFC 3629 also says many other values for \( ucs \) are illegal (in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff). However I encode these as though they are legal, so that utf8encode/fl_utf8decode will be the identity for all codes between 0 and 0x10fff.

### 32.11.3.25 `fl_utf8from_mb()`

```c
unsigned fl_utf8from_mb {
    char * dst,
    unsigned dstlen,
    const char * src,
    unsigned srclen )
```

Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.

Up to \( dstlen \) bytes are written to \( dst \), including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to \( dstlen \) then if you malloc a new array of size \( n+1 \) you will have the space needed for the entire string. If \( dstlen \) is zero then nothing is written and this call just measures the storage space needed.

On Unix or on Windows when a UTF-8 locale is in effect, this does not change the data. You may also want to check if \( fl_utf8test() \) returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.

### 32.11.3.26 `fl_utf8froma()`

```c
unsigned fl_utf8froma {
    char * dst,
    unsigned dstlen,
    const char * src,
    unsigned srclen )
```

Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

It is possible this should convert Microsoft's CP1252 to UTF-8 instead. This would translate the codes in the range 0x80-0x9f to different characters. Currently it does not do this.

Up to \( dstlen \) bytes are written to \( dst \), including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to \( dstlen \) then if you malloc a new array of
size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.
srclen is the number of bytes in src to convert.
If the return value equals srclen then this indicates that no conversion is necessary, as only ASCII characters are in the string.

32.11.3.27 fl_utf8fromwc()

unsigned fl_utf8fromwc (char ∗ dst, unsigned dstlen, const wchar_t ∗ src, unsigned srclen )

Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.
Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.
srclen is the number of words in src to convert. On Windows this is not necessarily the number of characters, due to there possibly being "surrogate pairs" in the UTF-16 encoding used. On Unix wchar_t is 32 bits and each location is a character.
On Unix if a src word is greater than 0x10ffff then this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER). Characters in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff are also illegal according to RFC 3629. However I encode these as though they are legal, so that fl_utf8towc will return the original data.
On Windows "surrogate pairs" are converted to a single character and UTF-8 encoded (as 4 bytes). Mismatched halves of surrogate pairs are converted as though they are individual characters.

32.11.3.28 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.

32.11.3.29 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

32.11.3.30 fl_utf8len1()

int fl_utf8len1 (char c )

Generated by Doxygen
32.11 Unicode and UTF-8 functions

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

\[ \text{fl_utf8len} \]

32.11.3.31  \textbf{fl_utf8locale()}

\begin{verbatim}
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.
\end{verbatim}

\textit{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.

32.11.3.32  \textbf{fl_utf8strlen()}

\begin{verbatim}
int fl_utf8strlen ( const char * text, int len )
Return the length in bytes of a UTF-8 string.
\end{verbatim}

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>text</th>
<th>encoded in UTF-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>number of Unicode characters, -1 to test until the end of text</td>
</tr>
</tbody>
</table>

Returns

number of bytes that make up the Unicode string

See also

fl_utf_nb_char(const unsigned char *buf, int len)

32.11.3.33  \textbf{fl_utf8test()}

\begin{verbatim}
int fl_utf8test ( const char * src, unsigned srclen )
Examine the first srclen bytes in src and returns a verdict on whether it is UTF-8 or not.
\end{verbatim}

- Returns 0 if there is any illegal UTF-8 sequences, using the same rules as \text{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 \( \text{srclen} \) 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.
32.11.3.34 fl_utf8to_mb()

unsigned fl_utf8to_mb (  
    const char * src,  
    unsigned srclen,  
    char * dst,  
    unsigned dstlen )

Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).
Unfortunately due to stupid design you will have to do this as needed for filenames. This is a bug on both Unix and Windows.
Up to dstlen bytes are written to dst, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to dstlen then if you malloc a new array of size n+1 you will have the space needed for the entire string. If dstlen is zero then nothing is written and this call just measures the storage space needed.
If fl_utf8locale() returns true then this does not change the data.

32.11.3.35 fl_utf8toa()

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

Convert a UTF-8 sequence into an array of 1-byte characters.
If the UTF-8 decodes to a character greater than 0xff then it is replaced with '?'.
Errors in the UTF-8 sequence are converted as individual bytes, same as fl_utf8decoder() 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.

32.11.3.36 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).

32.11.3.37 fl_utf8towc()

unsigned fl_utf8towc (  
    const char * src,  
    unsigned srclen,  
    char * dst,  
    unsigned dstlen )

Convert a UTF-8 sequence into a wide character.
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 a location to write, and dstlen is the number of locations in this string. At most dstlen-1 characters will be written there, plus a 0 terminating character. 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 characters 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.
32.11 Unicode and UTF-8 functions

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. \textasciitilde ucs4\textasciitilde as it were) except on Windows where it is equivalent to \textasciitilde 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 \textit{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, \texttt{-enable-cygwin}), either native (GDI) or X11.

32.11.3.38 fl_utf_nb_char()

int fl_utf_nb_char (  
  const unsigned char * buf,
  int len )
Returns the number of Unicode chars in the UTF-8 string.

See also

\texttt{fl_utf8strlen(const char *text, int len)}

32.11.3.39 fl_utf_strcasecmp()

int fl_utf_strcasecmp (  
  const char * s1,
  const char * s2 )
UTF-8 aware strcasecmp - converts to Unicode and tests.

Returns

\texttt{result of comparison}

Return values

\begin{tabular}{|c|c|}
\hline
0 & if the strings are equal \\
1 & if s1 is greater than s2 \\
-1 & if s1 is less than s2 \\
\hline
\end{tabular}

32.11.3.40 fl_utf_strncasecmp()

int fl_utf_strncasecmp (  
  const char * s1,
  const char * s2,
  int n )
UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

Generated by Doxygen
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s1,s2</strong></td>
<td>the UTF-8 strings to compare</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>the maximum number of UTF-8 characters to compare</td>
</tr>
</tbody>
</table>

Returns

result of comparison

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>if the strings are equal</td>
</tr>
<tr>
<td><strong>&gt;0</strong></td>
<td>if s1 is greater than s2</td>
</tr>
<tr>
<td><strong>&lt;0</strong></td>
<td>if s1 is less than s2</td>
</tr>
</tbody>
</table>

32.11.3.41 fl_utf_tolower()

```c
int fl_utf_tolower ( const unsigned char * str, int len, char * buf )
```

Converts the string `str` to its lower case equivalent into `buf`. Warning: to be safe buf length must be at least 3 * len [for 16-bit Unicode]

32.11.3.42 fl_utf_toupper()

```c
int fl_utf_toupper ( const unsigned char * str, int len, char * buf )
```

Converts the string `str` to its upper case equivalent into `buf`. Warning: to be safe buf length must be at least 3 * len [for 16-bit Unicode]

32.11.3.43 fl_wcwidth()

```c
int fl_wcwidth ( const char * src )
```

extended wrapper around `fl_wcwidth_(unsigned int ucs)` function.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>src</strong></td>
<td>pointer to start of UTF-8 byte sequence</td>
</tr>
</tbody>
</table>

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.

32.11.3.44 fl_wcwidth_()

```c
int fl_wcwidth_ ( unsigned int ucs )
```
32.12 Mac OS X-specific symbols

Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

Parameters

| in  | ucs | Unicode character value |

Returns

width of character in columns

See http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c for Markus Kuhn's original implementation of wcwidth() and wcswidth() (defined in IEEE Std 1002.1-2001) for Unicode.

WARNING: this function returns widths for "raw" Unicode characters. It does not even try to map C1 control characters (0x80 to 0x9F) to CP1252, and C0/C1 control characters and DEL will return -1. You are advised to use fl_width(const char* src) instead.

32.12 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 *)
  Returns the macOS-specific window reference corresponding to the given FL_Window object.
- void fl_open_callback (void(*)(const char*))
  Register a function called for each file dropped onto an application icon.

Variables

- int fl_mac_os_version
  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13).

32.12.1 Detailed Description

Mac OS X-specific symbols declared in <FL/platform.h>

See also

The Apple OS X Interface

32.12.2 Function Documentation
32.12.2.1 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).

32.12.2.2 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.

32.12.3 Variable Documentation

32.12.3.1 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.

32.13 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.

Classes

- class Fl_Color_Chooser
  
  The Fl_Color_Chooser widget provides a standard RGB color chooser.

- class Fl_File_Chooser
  
  The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Functions

- void fl_alert (const char ∗fmt,...)
  
  Shows an alert message dialog box.

- int fl_ask (const char ∗fmt,...)
  
  Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.

- void fl_beep (int type)
  
  Emits a system beep.
• int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style `fmt` message.

• int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)
  Shows a dialog displaying the printf style `fmt` message.

• int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

• int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)
  Pops up a window to let the user pick an arbitrary RGB color.

• char * fl_dir_chooser (const char *message, const char *fname, int relative)
  Shows a file chooser dialog and gets a directory.

• char * fl_file_chooser (const char *message, const char *fname, int relative)
  Shows a file chooser dialog and gets a filename.

• void fl_file_chooser_callback (void(cb)(const char *))
  Set the file chooser callback.

• void fl_file_chooser_ok_label (const char *l)
  Set the "OK" button label.

• const char * fl_input (const char *fmt, const char *defstr,...)
  Shows an input dialog displaying the `fmt` message with variable arguments.

• Fl_String fl_input_str (int &ret, int maxchar, 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 &ret, int maxchar, 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
  FLTKit calls Fl::error() to output a normal error message.
- static void(Fl::fatal)(const char *,...) = Fl_System_Driver::fatal
  FLTKit calls Fl::fatal() to output a fatal error message.
- const char * fl_cancel = "Cancel"
  string pointer used in common dialogs, you can change it to another language
- const char * fl_close = "Close"
  string pointer used in common dialogs, you can change it to another language
- Fl_Font fl_message_font_ = FL_HELVETICA
- Fl_Fontsize fl_message_size_ = -1
- const char * fl_no = "No"
  string pointer used in common dialogs, you can change it to another language
- const char * fl_ok = "OK"
  string pointer used in common dialogs, you can change it to another language
- const char * fl_yes = "Yes"
  string pointer used in common dialogs, you can change it to another language
- static void(Fl::warning)(const char *,...) = Fl_System_Driver::warning
  FLTKit calls Fl::warning() to output a warning message.

32.13.1 Detailed Description
Common dialog functions for file selection, message output, and more.

32.13.2 Function Documentation

32.13.2.1 fl_alert()

void fl_alert (const char * fmt, ...)  
Shows an alert message dialog box.
#include <FL/fl_ask.H>

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</td>
</tr>
</tbody>
</table>

32.13.2.2 fl_ask()

int fl_ask (const char * fmt, ...)  
Shows a dialog displaying the fmt message, this dialog features 2 yes/no buttons.
#include <FL/fl_ask.H>

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt</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 no button is selected</td>
</tr>
<tr>
<td>1</td>
<td>if yes is selected</td>
</tr>
</tbody>
</table>

**Deprecated** `fl_ask()` is deprecated since it uses "Yes" and "No" for the buttons which does not conform to the current FLTK Human Interface Guidelines. Use `fl_choice()` with the appropriate verbs instead.

### 32.13.2.3 fl_beep()

```c
void fl_beep (int type)
```

Emits a system beep.

This function is platform specific. Depending on the input `type` a different sound may be played or the system speaker may beep with a different volume.

On X the system speaker is used which may not work at all on newer systems that don't have a speaker. Since 1.4.0 `FL_BEEP_DEFAULT` and other types honor the system or user settings whereas `FL_BEEP_ERROR` uses 100% volume. This may be changed in a future version.

On Wayland an ASCII BEL (0x07) is output to stderr.

On Windows the `MessageBeep()` function is used to play different sounds depending on the `type` argument.

On macOS the system beep function `NSBeep()` is used for `FL_BEEP_DEFAULT` and `FL_BEEP_ERROR`. Other types are ignored.

On other platforms the behavior is undefined and may change in the future.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type</code></td>
<td>The beep type from the Fl_Beep enumeration (optional)</td>
</tr>
</tbody>
</table>

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

### 32.13.2.4 fl_choice()

```c
int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2, ...)
```

Shows a dialog displaying the printf style `fmt` message.

This dialog features up to 3 customizable choice buttons which are specified in order of right-to-left in the dialog, e.g.

![Figure 32.5 fl_choice() button ordering](image)

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

Three choices with printf() style formatting:
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>0</th>
<th>if the button with b0 text is pushed or the user pressed the Escape key or clicked the window close button</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the button with b1 text is pushed or the user pressed the Return key</td>
</tr>
<tr>
<td>2</td>
<td>if the button with b2 text is pushed</td>
</tr>
</tbody>
</table>
32.13.2.5 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmt</code></td>
<td>can be used as a printf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>b0</code></td>
<td>text label for right button 0</td>
</tr>
<tr>
<td><code>b1</code></td>
<td>text label for middle button 1 (can be 0)</td>
</tr>
<tr>
<td><code>b2</code></td>
<td>text label for left button 2 (can be 0)</td>
</tr>
</tbody>
</table>

### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>reserved, FLTK 1.3 only: another dialog is still open (not possible in 1.4)</td>
</tr>
<tr>
<td>-2</td>
<td>if the dialog was closed by pushing the window close button</td>
</tr>
<tr>
<td>-1</td>
<td>if the dialog was closed by hitting <code>Escape</code></td>
</tr>
<tr>
<td>0</td>
<td>if the button with <code>b0</code> text is pushed</td>
</tr>
<tr>
<td>1</td>
<td>if the button with <code>b1</code> text is pushed</td>
</tr>
<tr>
<td>2</td>
<td>if the button with <code>b2</code> text is pushed</td>
</tr>
</tbody>
</table>

32.13.2.6 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

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

![Figure 32.10 fl_color_chooser](image)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>Title label for the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>r,g,b</td>
<td>Color components in the range 0.0 to 1.0.</td>
</tr>
<tr>
<td>in</td>
<td>cmode</td>
<td>Optional mode for color chooser. See mode(int). Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

**Return values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>

### 32.13.2.7 fl_color_chooser() [2/2]

```c
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.
32.13 Common Dialog Classes and Functions

Note

#include <FL/Fl_Color_Chooser.H>

Figure 32.11 fl_color_chooser

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>Title label for the window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>r,g,b</td>
<td>Color components in the range 0 to 255.</td>
</tr>
<tr>
<td>in</td>
<td>cmode</td>
<td>Optional mode for color chooser. See mode(int). Default -1 if none (rgb mode).</td>
</tr>
</tbody>
</table>

Return values

| 1   | if user confirms the selection |
| 0   | if user cancels the dialog |

32.13.2.8 fl_dir_chooser()

char * fl_dir_chooser (const char * message, const char * fname, int relative) [related]

Shows a file chooser dialog and gets a directory.

Note

#include <FL/FL_File_Chooser.H>

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>message</th>
<th>title bar text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fname</td>
<td>initial/default directory name</td>
</tr>
<tr>
<td>in</td>
<td>relative</td>
<td>0 for absolute path return, relative otherwise</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

the directory path string chosen by the user or NULL if user cancels

32.13.2.9  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>

![Image of file chooser dialog]

Figure 32.12 Fl_File_Chooser

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>text in title bar</td>
</tr>
<tr>
<td>pat</td>
<td>filename pattern filter</td>
</tr>
<tr>
<td>fname</td>
<td>initial/default filename selection</td>
</tr>
<tr>
<td>relative</td>
<td>0 for absolute path name, relative path name otherwise</td>
</tr>
</tbody>
</table>

Returns

the user selected filename, in absolute or relative format or NULL if user cancels
32.13.2.10 fl_file_chooser_callback()

```c
void fl_file_chooser_callback (void (*)(const char *) cb) [related]
```

Set the file chooser callback.

Note

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

32.13.2.11 fl_file_chooser_ok_label()

```c
void fl_file_chooser_ok_label (const char * l) [related]
```

Set the "OK" button label.

Note

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

32.13.2.12 fl_input()

```c
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.

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td>in</td>
<td>defstr defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

**Returns**

the user string input if OK was pushed

**Return values**

- `NULL` if Cancel was pushed or the window was closed by the user

32.13.2.13 fl_input_str() [1/2]

```c
Fl_String fl_input_str (int & ret, int maxchar, const char * fmt, ...
```

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fmt can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td>in</td>
<td>defstr defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

**Returns**

the user string input if OK was pushed

**Return values**

- `NULL` if Cancel was pushed or the window was closed by the user
const char * defstr,
... )

Shows an input dialog displaying the `fmt` message with variable arguments. Like `fl_input()`, but this method has the additional 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. The `ret` variable is set to 0 if the user clicked OK, and to a negative value if the user canceled the dialog. If the dialog was canceled, the returned string will be empty.

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

Example:
int ret;
Fl_String str = fl_input_str(ret, 0, "Enter text:", "");
if (ret < 0)
    printf("Text input was canceled.\n");
else
    printf("Text is: '%s'\n", str.c_str());
// (str goes out of scope)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code> <code>ret</code></td>
<td>0 if user clicked OK, negative if dialog was canceled</td>
</tr>
<tr>
<td><code>in</code> <code>maxchar</code></td>
<td>input size limit in characters (not bytes), use 0 for no limit</td>
</tr>
<tr>
<td><code>in</code> <code>fmt</code></td>
<td>can be used as an sprintf-like format and variables for the message text</td>
</tr>
<tr>
<td><code>in</code> <code>defstr</code></td>
<td>defines the default returned string if no text is entered</td>
</tr>
</tbody>
</table>

### Returns

- the user string input if OK was clicked which can be empty
- an empty string and set `ret` to a negative value if the user canceled the dialog

### Since

1.4.0

32.13.2.14 `fl_input_str()` [2/2]

```c
Fl_String fl_input_str (int maxchar, const char * fmt, const char * defstr, ... )
```

Shows an input dialog displaying the `fmt` message with variable arguments.

### Note

No information is given if the user canceled the dialog or clicked OK.

See also

- `fl_input_str(int &ret, int maxchar, const char *label, const char *deflt = 0, ...)`

32.13.2.15 `fl_message()`

```c
void fl_message (const char * fmt, ... )
```

Shows an information message dialog box.

#include `<FL/fl_ask.H>`
32.13.2.16  fl_message_hotspot() [1/2]

```
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.

```
#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()`

32.13.2.17  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

- `in enable` non-zero enables hotspot behavior, 0 disables hotspot

32.13.2.18  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 cannot be changed. If you set an image() you should scale it to the available size, i.e. `w()` and `h()` of the icon box.

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

### 32.13.2.19 fl_message_icon_label()

```c
void fl_message_icon_label (const char * str )
```

Sets the icon label of the dialog window used in many common dialogs.

This icon label will be used in the next call of one of the common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`. The label `str` is stored internally as a reference, it must be in scope until the dialog function (e.g. `fl_choice`) is called.

It applies only to the next call of one of the common dialogs and will be reset after that call so the next dialog will use its default label unless set again.

**Note**

This label string must be short, usually only one character so it fits in the icon box. You can use any valid UTF-8 character, e.g. the Euro sign ("€") which is three bytes in UTF-8 encoding.

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>str</th>
<th>icon label</th>
</tr>
</thead>
</table>

### 32.13.2.20 fl_message_position() [1/3]

```c
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).

```c
#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)

32.13.2.21 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.
#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)

32.13.2.22 fl_message_position() [3/3]

int fl_message_position (  
    int * x,  
    int * y )
 Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(),
fl_choice(), fl_input(), fl_password().
#include <FL/fl_ask.H>
The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the
hotspot mode set by fl_message_hotspot(int).
The preferred position set by any of the fl_message_position() variants affects only the next call of one of the
common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.

Parameters

| 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)
32.13.2.23 **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.

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

**Parameters**

- **title**
  - window label, string copied internally

32.13.2.24 **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.

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

**Parameters**

- **title**
  - default window label, string copied internally

32.13.2.25 **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.

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

**Parameters**

- **fmt**
  - can be used as an sprintf-like format and variables for the message text
- **defstr**
  - 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 |

32.13.2.26 fl_password_str() [1/2]

`Fl_String fl_password_str (``int & ret,``
``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

| out  | `ret` | 0 if user clicked OK, negative if dialog was canceled |
| in   | `maxchar` | input size limit in characters (not bytes), use 0 for no limit |
| in   | `fmt` | can be used as an sprintf-like format and variables for the message text |
| in   | `defstr` | defines the default returned string if no text is entered |

Returns

the user string input if OK was clicked which can be empty

an empty string and set `ret` to a negative value if the user canceled the dialog

Since

1.4.0

32.13.2.27 fl_password_str() [2/2]

`Fl_String fl_password_str (``int maxchar,``
``const char * fmt,``
``const char * defstr,``
``... )``

Shows an input dialog displaying the `fmt` message with variable arguments.

Note

No information is given if the user canceled the dialog or clicked OK.

See also

`fl_password_str(int &ret, int maxchar, const char *label, const char *deflt = 0, ...)"
32.13.3 Variable Documentation

32.13.3.1 error

```c
void(* Fl::error)(const char *format,...) = Fl_System_Driver::error [static]
```

FLTK calls Fl::error() to output a normal error message.
The default version on Windows displays the error message in a MessageBox window.
The default version on all other platforms prints the error message to stderr.
You can override the behavior by setting the function pointer to your own routine.

**Note**

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

32.13.3.2 fatal

```c
void(* Fl::fatal)(const char *format,...) = Fl_System_Driver::fatal [static]
```

FLTK calls Fl::fatal() to output a fatal error message.
The default version on Windows displays the error message in a MessageBox window.
The default version on all other platforms prints the error message to stderr.
You can override the behavior by setting the function pointer to your own routine.

**Note**

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

32.13.3.3 warning

```c
void(* Fl::warning)(const char *format,...) = Fl_System_Driver::warning [static]
```

FLTK calls Fl::warning() to output a warning message.
The default version on Windows returns without printing a warning message, because Windows programs normally don't have stderr (a console window) enabled.
The default version on all other platforms prints the warning message to stderr.
You can override the behavior by setting the function pointer to your own routine.

**Note**

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

32.14 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*
32.14 File names and URI utility functions

Typedefs

- typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
  
  File sorting function.

Functions

- void fl_decode_uri (char *uri)
  
  Decodes a URL-encoded string.
- int fl_filename_absolute (char *to, int tolen, const char *from)
  
  Makes a filename absolute from a relative filename.
- int fl_filename_expand (char *to, int tolen, const char *from)
  
  Expands a filename containing shell variables and tilde (~).
- const char * fl_filename_ext (const char *buf)
  
  Gets the extension of a filename.
- void fl_filename_free_list (struct dirent ***, int n)
  
  Free the list of filenames that is generated by fl_filename_list().
- int fl_filename_isdir (const char *name)
  
  Determines if a file exists and is a directory from its filename.
- int fl_filename_list (const char *d, struct dirent ****, Fl_File_Sort_F *s=fl_numericsort)
  
  Portable and const-correct wrapper for the scandir() function.
- int fl_filename_match (const char *name, const char *pattern)
  
  Checks if a string s matches a pattern p.
- const char * fl_filename_name (const char *filename)
  
  Gets the file name from a path.
- int fl_filename_relative (char *to, int tolen, const char *from)
  
  Makes a filename relative to the current working directory.
- char * fl_filename_setext (char *to, int tolen, const char *ext)
  
  Replaces the extension in buf of max.
- int fl_open_uri (const char *uri, char *msg, int msglen)
  
  Opens the specified Uniform Resource Identifier (URI).

32.14.1 Detailed Description

File names and URI functions defined in <FL/filename.H>

32.14.2 Typedef Documentation

32.14.2.1 Fl_File_Sort_F

typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

File sorting function.

See also

   fl_filename_list()
32.14.3.1 fl_decode_uri()

```c
void fl_decode_uri(
    char * uri
)
```
Decodes a URL-encoded string.
In a Uniform Resource Identifier (URI), all non-ASCII bytes and several others (e.g., '<', ' ', '') are URL-encoded using 3 bytes by "%XY" where XY is the hexadecimal value of the byte. This function decodes the URI restoring its original UTF-8 encoded content. Decoding is done in-place.

32.14.3.2 fl_filename_absolute()

```c
int fl_filename_absolute(
    char * to,
    int tolen,
    const char * from
)
```
Makes a filename absolute from a relative filename.

```c
#include <FL/filename.H>
[..]
fl_chdir("/var/tmp");
fl_filename_absolute(out, sizeof(out), "foo.txt"); // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), ".foo.txt"); // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), ".../log/messages"); // out="/var/log/messages"
```

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting absolute filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the absolute filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>relative filename</td>
</tr>
</tbody>
</table>

**Returns**

0 if no change, non zero otherwise

32.14.3.3 fl_filename_expand()

```c
int fl_filename_expand(
    char * to,
    int tolen,
    const char * from
)
```
Expands a filename containing shell variables and tilde (~).
Currently handles these variants:

- "~username" // if 'username' does not exist, result will be unchanged
- "~/file"
- "$VARNAME" // does NOT handle ${VARNAME}

**Examples:**

```c
#include <FL/filename.H>
[..]
putenv("TMPDIR=/var/tmp");
fl_filename_expand(out, sizeof(out), ".fred/.cshrc"); // out="/usr/fred/.cshrc"
fl_filename_expand(out, sizeof(out), "./.cshrc"); // out="/usr/<yourname>/.cshrc"
fl_filename_expand(out, sizeof(out), "/TMPDIR/foo.txt"); // out="/var/tmp/foo.txt"
```

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting expanded filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the expanded filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>filename containing shell variables</td>
</tr>
</tbody>
</table>
32.14 File names and URI utility functions

Returns
0 if no change, non zero otherwise

32.14.3.4 fl_filename_ext()

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

32.14.3.5 fl_filename_free_list()

void fl_filename_free_list (  
    struct dirent *** list,  
    int n )

Free the list of filenames that is generated by fl_filename_list().
Free everything that was allocated by a previous call to fl_filename_list(). Use the return values as parameters for this function.

Parameters

| in,out | list | table containing the resulting directory listing |
| in     | n    | number of entries in the list |

32.14.3.6 fl_filename_isdir()

int fl_filename_isdir (  
    const char * n )

Determines if a file exists and is a directory from its filename.
#include <FL/filename.H>
[...]  
fl_filename_isdir(“/etc”);  // returns non-zero  
fl_filename_isdir(“/etc/hosts”);  // returns 0

Parameters

| in     | n   | the filename to parse |

Returns
non zero if file exists and is a directory, zero otherwise

Generated by Doxygen
32.14.3.7 fl_filename_list()

```c
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.fn.
d_name is the null-terminated file name. A pointers array to these dirent's is created and a pointer to the array is returned in *list. The number of entries is given as a return value. If there is an error reading the directory a number less than zero is returned, and errno has the reason; errno does not work under Windows.

Include:
```c
#include <FL/filename.H>
```

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>d</th>
<th>the name of the directory to list. It does not matter if it has a trailing slash.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>list</td>
<td>table containing the resulting directory listing</td>
</tr>
<tr>
<td>in</td>
<td>sort</td>
<td>sorting functor:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_alphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared according to their ASCII ordering uppercase before lowercase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casealphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared equally case is not significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_casenumeric: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared equally case is not significant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fl_numeric: The files are sorted in ascending &quot;alphanumeric&quot; order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared according to their ASCII ordering - uppercase before lowercase.</td>
</tr>
</tbody>
</table>

**Returns**

the number of entries if no error, a negative value otherwise.

**Todo** should support returning OS error messages

32.14.3.8 fl_filename_match()

```c
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.
32.14 File names and URI utility functions

- 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".
- \( \backslash x \) only applies to the fl_filename_match special characters * [ ]
- \( \backslash x \) needs a double \() or the compiler will complain about non-standard escape sequences.

Include:
```cpp
#include <FL/filename.H>
```

Parameters
```
in s the string to check for a match
```
```
in p the string pattern
```

Returns
- non zero if the string matches the pattern

### 32.14.3.9 fl_filename_name()

```cpp
const char * fl_filename_name (  
    const char * filename  
)
```

Gets the file name from a path.
Similar to basename(3), exceptions shown below.
```cpp
#include <FL/filename.H>
[..]  
const char * out;  
out = fl_filename_name("/usr/lib"); // out="lib"  
out = fl_filename_name("/usr/"); // out="" (basename(3) returns "usr" instead)  
out = fl_filename_name("/"); // out="" (basename(3) returns "/" instead)  
out = fl_filename_name("."); // out="."  
out = fl_filename_name(".."); // out=".."
```

Returns
- a pointer to the char after the last slash, or to filename if there is none.

### 32.14.3.10 fl_filename_relative()

```cpp
int fl_filename_relative (  
    char * to,  
    int tolen,  
    const char * from  
)
```

Makes a filename relative to the current working directory.
```cpp
#include <FL/filename.H>
[..]  
fl_chdir("/var/tmp/somedir"); // set cwd to /var/tmp/somedir  
[..]  
char out[FL_PATH_MAX];  
fl_filename_relative(out, sizeof(out), "/var/tmp/somedir/foo.txt"); // out="foo.txt", return=1  
fl_filename_relative(out, sizeof(out), "/var/tmp/foo.txt"); // out="../foo.txt", return=1  
fl_filename_relative(out, sizeof(out), ".foo.txt"); // out="foo.txt", return=0 (no change)  
fl_filename_relative(out, sizeof(out), ".foo.txt"); // out="..foo.txt", return=0 (no change)  
fl_filename_relative(out, sizeof(out), ".foo.txt"); // out="..foo.txt", return=0 (no change)
```

Generated by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>to</th>
<th>resulting relative filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tolen</td>
<td>size of the relative filename buffer</td>
</tr>
<tr>
<td>in</td>
<td>from</td>
<td>absolute filename</td>
</tr>
</tbody>
</table>

#### Returns

0 if no change, non zero otherwise

#### 32.14.3.11 fl_filename_setext()

```c
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.

#### 32.14.3.12 fl_open_uri()

```c
int fl_open_uri (const char * uri, char * msg, int msglen)
```

Opens the specified Uniform Resource Identifier (URI).

Uses an operating-system dependent program or interface. For URIs using the "ftp", "http", or "https" schemes, the system default web browser is used to open the URI, while "mailto" and "news" URIs are typically opened using the system default mail reader and "file" URIs are opened using the file system navigator.

On success, the (optional) `msg` buffer is filled with the command that was run to open the URI; on Windows, this will always be "open uri".

On failure, the `msg` buffer is filled with an English error message.

**Note**

**Platform Specific Issues: Windows**

With "file:" based URIs on Windows, you may encounter issues with anchors being ignored. Example: "file:///c:/some/index.html#anchor" may open in the browser without the "#anchor" suffix. The behavior seems to vary across different Windows versions. Workaround: open a link to a separate html file that redirects to the desired "file:" URI.

**Example**

```c
#include <FL/filename.H>
...
char errmsg[512];
if ( !fl_open_uri("http://google.com/", errmsg, sizeof(errmsg)) ) {
    char warnmsg[768];
    sprintf(warnmsg, "Error: %s", errmsg);
    fl_alert(warnmsg);
}
```
32.15 **Fl_string**

**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

### 32.15.1 Detailed Description

### 32.15.2 Function Documentation

#### 32.15.2.1 fl_strdup()

```c
char * fl_strdup(const char *s)
```

Cross platform interface to POSIX function `strdup()`.

The `fl_strdup()` function returns a pointer to a new string which is a duplicate of the string `s`. Memory for the new string is obtained with `malloc(3)`, and can be freed with `free(3)`.

**Implementation:**

- POSIX: `strdup()`
- WinAPI: `_strdup()`
Chapter 33

Class Documentation

33.1 Fl_GIF_Image::GIF_FRAME::CPAL Struct Reference

Public Attributes

- uchar b
- uchar g
- uchar r

The documentation for this struct was generated from the following file:

- Fl_GIF_Image.H

33.2 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

33.3 Fl Class Reference

The Fl is the FLTK global (static) class containing state information and global methods for the current application.

#include <Fl.H>

Public Types

- enum Fl_Option {
  OPTION_ARROW_FOCUS = 0, OPTION_VISIBLE_FOCUS, OPTION_DND_TEXT, OPTION_SHOW.Tooltips,
  OPTION_FNFC_USES_GTK, OPTION_PRINTER_USES_GTK, OPTION_SHOW_SCALING, OPTION_FNFC_USES_ZENITY,
  OPTION_LAST
}

  Enumerator for global FLTK options.

Static Public Member Functions

- static int abi_check (const int val=FL_ABI_VERSION)
  Returns whether the runtime library ABI version is correct.
- static int abi_version ()
Returns the compiled-in value of the FL_ABI_VERSION constant.

- static int **add_awake_handler** (Fl_Awake_Handler, void *)
  Adds an awake handler for use in awake().

- static void **add_check** (Fl_Timeout_Handler, void *data=0)
  FLTK will call this callback just before it flushes the display and waits for events.

- static void **add_clipboard_notify** (Fl_Clipboard_Notify_Handler, 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 *data=0)
  Adds file descriptor fd to listen to.

- static void **add_fd** (int fd, int when, Fl_FD_Handler cb, void *data=0)
  Adds file descriptor fd to listen to.

- static void **add_handler** (Fl_Event_Handler h)
  Install a function to parse unrecognized events.

- static void **add_idle** (Fl_Idle_Handler cb, void *data=0)
  Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).

- static void **add_system_handler** (Fl_System_Handler h, void *data)
  Install a function to intercept system events.

- static void **add_timeout** (double t, Fl_Timeout_Handler cb, void *data=0)
  Adds a one-shot timeout callback.

- static int **api_version** ()
  Returns the compiled-in value of the FL_API_VERSION constant.

- static int **arg** (int argc, char **argv, int &i)
  Parse a single switch from argv, starting at word i.

- static void **args** (int argc, char **argv)
  Parse all command line switches matching standard FLTK options only.

- static 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 ()
  Gets the current autolink mode for Cairo support.
- static void cairo_autolink_context (bool alink)
  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.
- static cairo_t * cairo_cc ()
  Gets the current Cairo context linked with a fltk window.
- static void cairo_cc (cairo_t c, bool own=false)
  Sets the current Cairo context to c.
- static void cairo_flush (cairo_t c)
  Flush Cairo drawings on Cairo context c.
- static cairo_t * cairo_make_current (Fl_Window * w)
  Provides a Cairo context for window wi.
- static Fl_Callback_Reason callback_reason ()
  Give the reason for calling a callback.
- static int check ()
  Same as Fl::wait(0).
- static void clear_widget_pointer (Fl_Widget const * w)
  Clears a widget pointer in the watch list.
- static int clipboard_contains (const char * type)
  Returns non 0 if the clipboard contains data matching type.
- static void 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 * w, void *)
  Default callback for window widgets.
- static void delete_widget (Fl_Widget * w)
  Schedules a widget for deletion at the next call to the event loop.
- static void disable_im ()
  Disables the system input methods facilities.
- static void display (const char *)
  Sets the X or Wayland display to use for all windows.
- static int dnd ()
Initiate a Drag And Drop operation.

- static int dnd_text_ops ()
  Gets whether drag and drop text operations are supported.
- static void dnd_text_ops (int v)
  Sets whether drag and drop text operations are supported.
- static void do_widget_deletion ()
  Deletes widgets previously scheduled for deletion.
- static int draw_box_active ()
  Determines if the currently drawn box is active or inactive.
- static int draw_GL_text_with_textures ()
  Returns whether 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_EventDispatch event_dispatch ()
  Return the current event dispatch function.
- static void event_dispatch (FL_EventDispatch d)
  Set a new event dispatch function.
- static int event_dx ()
  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int event_dy ()
  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int event_inside (const Fl_Widget *)
  Returns whether or not the mouse event is inside a given child widget.
• static int event_inside (int, int, int, int)
  Returns whether or not the mouse event is inside the given rectangle.

• static int event_is_click()
  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".

• static void event_is_click (int i)
  Clears the value returned by Fl::event_is_click().

• static int event_key ()
  Gets which key on the keyboard was last pushed.

• static int event_key (int key)
  Returns true if the given key was held down (or pressed) during the last event.

• static int event_length ()
  Returns the length of the text in Fl::event_text().

• static int event_original_key ()
  Returns the keycode of the last key event, regardless of the NumLock state.

• static int event_shift ()
  Returns non-zero if the Shift key is pressed.

• static int event_state ()
  Returns the keyboard and mouse button states of the last event.

• static int event_state (int mask)
  Returns non-zero if any of the passed event state bits are turned on.

• static const char * event_text ()
  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.

• static int event_x ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

• static int event_x_root ()
  Returns the mouse position on the screen of the event.

• static int event_y ()
  Returns the mouse position of the event relative to the Fl_Window it was passed to.

• static int event_y_root ()
  Returns the mouse position on the screen of the event.

• static Fl_Window * first_window ()
  Returns the first top-level window in the list of shown() windows.

• static void first_window (Fl_Window *)
  Sets the window that is returned by first_window().

• static void flush ()
  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.

• static Fl_Widget * focus ()
  Gets the current Fl::focus() widget.

• static void focus (Fl_Widget *)
  Sets the widget that will receive FL_KEYBOARD events.

• static void foreground (uchar, uchar, uchar)
  Changes fl_color(FL_FOREGROUND_COLOR).

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

• static int get_awake_handler_ (Fl_Awake_Handler &, void *&)
  Gets the last stored awake handler for use in awake().

• static Fl_Box_Draw_F * get_boxtype (Fl_Boxtype)
  Gets the current box drawing function for the specified box type.

• static unsigned get_color (Fl_Color i)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)
  Returns the RGB value(s) for the given FLTK color index.
• static void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)
  Returns the RGBA value(s) for the given FLTK color index.
• static const char * get_font (Fl_Font)
  Gets the string for this face.
• static const char * get_font_name (Fl_Font, int *attributes=0)
  Get a human-readable string describing the family of this face.
• static int get_font_sizes (Fl_Font, int *&sizep)
  Return an array of sizes in sizep.
• static int get_key (int key)
  Returns true if the given key is held down now.
• static void get_mouse (int &, int &)
  Return where the mouse is on the screen by doing a round-trip query to the server.
• static void get_system_colors ()
  Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().
• static int gl_visual (int, int *alist=0)
  This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.
• static Fl_Window * grab ()
  Returns the window that currently receives all events.
• static void grab (Fl_Window &win)
  See grab(Fl_Window*)
• static void grab (Fl_Window *)
  Selects the window to grab.
• static int h ()
  Returns the height in pixels of the main screen work area.
• static int handle (int, Fl_Window *)
  Handle events from the window system.
• static int handle_ (int, Fl_Window *)
  Handle events from the window system.
• static int has_check (Fl_Timeout_Handler, void *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 cb, void *data=0)
  Returns true if the timeout exists and has not been called yet.
• static void hide_all_windows ()
  Hide all visible window to make FLTK leav Fl::run().
• static int is_scheme (const char *name)
  Returns whether the current scheme is the given name.
• static void keyboard_screen_scaling (int value)
  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
• static int lock ()
  The lock() method blocks the current thread until it can safely access FLTK widgets and data.
• static int menu_linespacing ()
  Gets the default line spacing used by menus.
• static void menu_linespacing (int H)
  Sets the default line spacing used by menus.
• static Fl_Window * modal ()
  Returns the top-most modal() window currently shown.
• static Fl_Window * next_window (const Fl_Window *)
  Returns the next top-level window in the list of shown() windows.
• static Fl_Timestamp now ()
  Set a time stamp at this point in time.
• static bool option (Fl_Option opt)
  FLTK library options management.
• static void option (Fl_Option opt, bool val)
  Override an option while the application is running.
• static void own_colormap ()
  Makes FLTK use its own colormap.
• static void paste (Fl_Widget &receiver)
  Backward compatibility only.
• static void paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)
  Pastes the data from the selection buffer (source is 0) or the clipboard (source is 1) into receiver.
• static int program_should_quit ()
  Returns non-zero when a request for program termination was received and accepted.
• static void program_should_quit (int should_i)
  Indicate to the FLTK library whether a program termination request was received and accepted.
• static Fl_Widget * pushed ()
  Gets the widget that is being pushed.
• static void pushed (Fl_Widget *)
  Sets the widget that is being pushed.
• static Fl_Widget * readqueue ()
  Reads the default callback queue and returns the first widget.
• static int ready ()
  This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.
• static void redraw ()
  Redraws all widgets.
• static void release ()
  Releases the current grabbed window, equals grab(0).
• static void release_widget_pointer (Fl_Widget * &w)
  Releases a widget pointer from the watch list.
• static int reload_scheme ()
  Called internally when setting a new scheme according to scheme name.
• static void remove_check (Fl_Timeout_Handler, void *=0)
  Removes a check callback.
• static void remove_clipboard_notify (Fl_Clipboard_Notify_Handler h)
  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
• static void remove_fd (int)
  Removes a file descriptor handler.
• static void remove_fd (int, int when)
  Removes a file descriptor handler.
• static void remove_handler (Fl_Event_Handler)
  Removes a previously added event handler.
• static void remove_idle (Fl_Idle_Handler cb, void *data=0)
  Removes the specified idle callback, if it is installed.
• static void remove_system_handler (Fl_System_Handler h)
  Removes a previously added system event handler.
• static void remove_timeout (Fl_Timeout_Handler cb, void *data=0)
  Removes a timeout callback from the timer queue.
• static void repeat_timeout (double t, Fl_Timeout_Handler cb, void *data=0)
  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
• static int run ()
  Calls Fl::wait() repeatedly as long as any windows are displayed.
• static void run_checks ()
• static void run_idle ()
• static const char * scheme ()
  See void scheme(const char *name)
• static int scheme (const char *name)
  Sets the current widget scheme.
• static int screen_count ()
  Gets the total count of available screens.
• static void screen_dpi (float &h, float &v, int n=0)
  Gets the screen resolution in dots-per-inch for the given screen.
• static Fl_Screen_Driver * screen_driver ()
  Returns a pointer to the unique Fl_Screen_Driver object of the platform.
• static int screen_num (int x, int y)
  Gets the screen number of a screen that contains the specified screen position x, y.
• static int screen_num (int x, int y, int w, int h)
  Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.
• static float screen_scale (int n)
  Current value of the GUI scaling factor for screen number n (n \[0 .. Fl::screen_count()-1\])
• static void screen_scale (int n, float factor)
  Sets the value of the GUI scaling factor for screen number n (n \[0 .. Fl::screen_count()-1\]).
• static int screen_scaling_supported ()
  See if scaling factors are supported by this platform.
• static void screen_work_area (int &X, int &Y , int &W, int &H)
  Gets the bounding box of the 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 double seconds_between (Fl_Timestamp &back, Fl_Timestamp &further_back)
  Return the time in seconds between two time stamps.
• static double seconds_since (Fl_Timestamp &then)
  Return the time in seconds between now and a previously taken time stamp.
• static void selection (Fl_Widget &owner, const char *s, int len)
  Changes the current selection.
• static Fl_Widget * selection_owner ()
back-compatibility only: Gets the widget owning the current selection

• static void selection_owner (Fl_Widget *)
  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the
  owner to NULL, without changing the actual text of the selection.

• static int selection_to_clipboard ()
  Returns the current selection_to_clipboard mode.

• static void selection_to_clipboard (int mode)
  Copies selections on X11 directly to the clipboard if enabled.

• static void set_abort (Fl_Abort_Handler f)
  For back compatibility, sets the void Fl::fatal handler callback.

• static void set_atclose (Fl_Atclose_Handler f)
  For back compatibility, sets the Fl::atclose handler callback.

• static void set_box_color (Fl_Color)
  Sets the drawing color for the box that is currently drawn.

• static void set_boxtype (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar)
  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)
  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 *)
  Run a command line on the computer.

• static Fl_System_Driver * system_driver ()
  Returns a pointer to the unique Fl_System_Driver object of the platform.

• static int test_shortcut (Fl_Shortcut)
  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in
  Fl_Button).

• static void thread_message ()
  The thread_message() method returns the last message that was sent from a child by the awake() method.

• static long ticks_between (Fl_Timestamp &back, Fl_Timestamp &further_back)
  Return the time in ticks (60Hz) between two time stamps.

• static long ticks_since (Fl_Timestamp &then)
  Return the time in ticks (60Hz) between now and a previously taken time stamp.

• static void unlock ()
  The unlock() method releases the lock that was set using the lock() method.
• static int use_high_res_GL ()
  returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.
• static void use_high_res_GL (int val)
  sets whether GL windows should be drawn at high resolution on Apple computers with retina displays
• static double version ()
  Returns the compiled-in value of the FL_VERSION constant.
• static int visible_focus ()
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.
• static void visible_focus (int v)
  Gets or sets the visible keyboard focus on buttons and other non-text widgets.
• static int visual (int)
  Selects a visual so that your graphics are drawn correctly.
• static int w ()
  Returns the width in pixels of the main screen work area.
• static int wait ()
  Waits until "something happens" and then returns.
• static double wait (double time)
  Waits a maximum of time_to_wait seconds or until "something happens".
• static void watch_widget_pointer (Fl_Widget ∗&w)
  Adds a widget pointer to the widget watch list.
• static int x ()
  Returns the leftmost x coordinate of the main screen work area.
• static int y ()
  Returns the topmost y coordinate of the main screen work area.

Static Public Attributes

• static void(* atclose )(Fl_Window ∗, void ∗)
  Back compatibility: default window callback handler.
• static char const *const clipboard_image = "image"
  Denotes image data.
• static char const *const clipboard_plain_text = "text/plain"
  Denotes plain textual data.
• static void(* error )(const char ∗,...) = Fl_System_Driver::error
  FLTK calls Fl::error() to output a normal error message.
• static void(* fatal )(const char ∗,...) = Fl_System_Driver::fatal
  FLTK calls Fl::fatal() to output a fatal error message.
• static const char ∗const help = helpmsg+13
  Usage string displayed if Fl::args() detects an invalid argument.
• static void(* idle )()
  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
• static void(* warning )(const char ∗,...) = Fl_System_Driver::warning
  FLTK calls Fl::warning() to output a warning message.

Friends

• class Fl_System_Driver

33.3.1 Detailed Description
The Fl is the FLTK global (static) class containing state information and global methods for the current application.
33.3.2 Member Enumeration Documentation

### Fl_Option

```cpp
enum Fl::Fl_Option
```

Enumerator for global FLTK options. These options can be set system wide, per user, or for the running application only.

See also

- `Fl::option(Fl_Option, bool)
- `Fl::option(Fl_Option)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTION_ARROW_FOCUS</td>
<td>When switched on, moving the text cursor beyond the start or end of a text in a text widget will change focus to the next text widget. (This is considered 'old' behavior) When switched off (default), the cursor will stop at the end of the text. Pressing Tab or Ctrl-Tab will advance the keyboard focus. See also: <code>Fl::input::tab_navi()</code></td>
</tr>
<tr>
<td>OPTION_VISIBLE_FOCUS</td>
<td>If visible focus is switched on (default), FLTK will draw a dotted rectangle inside the widget that will receive the next keystroke. If switched off, no such indicator will be drawn and keyboard navigation is disabled.</td>
</tr>
<tr>
<td>OPTION_DND_TEXT</td>
<td>If text drag-and-drop is enabled (default), the user can select and drag text from any text widget. If disabled, no dragging is possible, however dropping text from other applications still works.</td>
</tr>
<tr>
<td>OPTION_SHOW_TOOLTIPS</td>
<td>If tooltips are enabled (default), hovering the mouse over a widget with a tooltip text will open a little tooltip window until the mouse leaves the widget. If disabled, no tooltip is shown.</td>
</tr>
<tr>
<td>OPTION_FNFC_USES_GTK</td>
<td>When switched on (default), <code>Fl_Native_File_Chooser</code> 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), <code>Fl_Printer</code> runs the GTK printer dialog if the GTK library is available on the platform (linux/unix only). When switched off, the GTK printer dialog isn't used even if the GTK library is available.</td>
</tr>
<tr>
<td>OPTION_SHOW_SCALING</td>
<td>When switched on (default), the library shows in a transient yellow window the zoom factor value. When switched off, no such window gets displayed.</td>
</tr>
<tr>
<td>OPTION_FNFC_USES_ZENITY</td>
<td>Meaningful for the Wayland/X11 platform only. When switched on (default), the library uses a Zenity-based file dialog. When switched off, the GTK file dialog is used instead.</td>
</tr>
</tbody>
</table>

### OPTION_LAST

For internal use only.

33.3.3 Member Function Documentation

#### abi_check()

```cpp
static int Fl::abi_check ( const int val = FL_ABI_VERSION ) [inline], [static]
```

Returns whether the runtime library ABI version is correct. This enables you to check the ABI version of the linked FLTK library at runtime.

Generated by Doxygen
Returns 1 (true) if the compiled ABI version (in the header files) and the linked library ABI version (used at runtime) are the same, 0 (false) otherwise.

Argument \texttt{val} can be used to query a particular library ABI version. Use for instance 10303 to query if the runtime library is compatible with FLTK ABI version 1.3.3. This is rarely useful.

The default \texttt{val} argument is \texttt{FL_ABI_VERSION}, which checks the version defined at configure time (i.e. in the header files at program compilation time) against the linked library version used at runtime. This is particularly useful if you linked with a shared object library, but it also concerns static linking.

See also

\texttt{Fl::abi_version()}

### 33.3.3.2 \texttt{abi_version()}

\begin{verbatim}
int Fl::abi_version ( ) [static]
\end{verbatim}

Returns the compiled-in value of the \texttt{FL_ABI_VERSION} constant.

This is useful for checking the version of a shared library.

### 33.3.3.3 \texttt{add_check()}

\begin{verbatim}
void Fl::add_check (  
    Fl_Timeout_Handler cb,  
    void * argp = 0 ) [static]
\end{verbatim}

FLTK will call this callback just before it flushes the display and waits for events.

This is different than an idle callback because it is only called once, then FLTK calls the system and tells it not to return until an event happens.

This can be used by code that wants to monitor the application's state, such as to keep a display up to date. The advantage of using a check callback is that it is called only when no events are pending. If events are coming in quickly, whole blocks of them will be processed before this is called once. This can save significant time and avoid the application falling behind the events.

Sample code:

```c
bool state_changed; // anything that changes the display turns this on
void callback(void*) {  
    if (!state_changed) return;
    state_changed = false;
    do_expensive_calculation();
    widget->redraw();
}
main() {  
    Fl::add_check(callback);
    return Fl::run();
}
```

### 33.3.3.4 \texttt{add_fd()} [1/2]

\begin{verbatim}
void Fl::add_fd (  
    int fd,  
    Fl_FD_Handler cb,  
    void * d = 0 ) [static]
\end{verbatim}

Adds file descriptor \texttt{fd} to listen to.

See \texttt{Fl::add_fd(int fd, int when, Fl_FD_Handler cb, void* = 0)} for details.

### 33.3.3.5 \texttt{add_fd()} [2/2]

\begin{verbatim}
void Fl::add_fd (  
    int fd,  
    int when,  
    Fl_FD_Handler cb,  
    void * d = 0 ) [static]
\end{verbatim}

Add file descriptor \texttt{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.

Under macOS, Fl::add_fd() opens the display if that's not been done before.

### 33.3.3.6 add_idle()

```c
void Fl::add_idle (    
    Fl_Idle_Handler cb,  
    void * data = 0 ) [static]
```

Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait()) return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).

The idle function can be used to get background processing done.

You can have multiple idle callbacks. To remove an idle callback use Fl::remove_idle().

Fl::wait() and Fl::check() call idle callbacks, but Fl::ready() does not.

The idle callback can call any FLTK functions, including Fl::wait(), Fl::check(), and Fl::ready().

FLTK will not recursively call the idle callback.

### 33.3.3.7 add_timeout()

```c
void Fl::add_timeout (  
    double time,  
    Fl_Timeout_Handler cb,  
    void * data = 0 ) [static]
```

Adds a one-shot timeout callback.

The callback function cb will be called by Fl::wait() at time seconds after this function is called. The callback function must have the signature Fl_Timeout_Handler. The optional data argument is passed to the callback (default: NULL).

The timer is removed from the timer queue before the callback function is called. It is safe to reschedule the timeout inside the callback function.

You can have multiple timeout callbacks, even the same timeout callback with different timeout values and/or different data values. They are all considered different timer objects.

To remove a timeout while it is active (pending) use Fl::remove_timeout().

If you need more accurate, repeated timeouts, use Fl::repeat_timeout() to reschedule the subsequent timeouts. Please see Fl::repeat_timeout() for an example.

Since version 1.4, a timeout can be started from a child thread under the condition that the call to Fl::add_timeout is wrapped in Fl::lock() and Fl::unlock().

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>delta time in seconds until the timer expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

#### See also

- Fl_Timeout_Handler
- Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
- Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)
- Fl::has_timeout(Fl_Timeout_Handler cb, void *data)
### 33.3.3.8 api_version()

```cpp
int Fl::api_version() [static]
```

Returns the compiled-in value of the FL_API_VERSION constant. This is useful for checking the version of a shared library.

### 33.3.3.9 arg()

```cpp
int Fl::arg (int argc, char ** argv, int & i) [static]
```

Parse a single switch from `argv`, starting at word `i`. Returns the number of words eaten (1 or 2, or 0 if it is not recognized) and adds the same value to `i`. This is the default argument handler used internally by `Fl::args(...)`, but you can use this function if you prefer to step through the standard FLTK switches yourself. All standard FLTK switches except `-bg2` may be abbreviated to just one letter and case is ignored:

- `-bg` or `-background` color
  Sets the background color using `Fl::background()`.

- `-bg2` 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` or `-foreground` color
  Sets the foreground color using `Fl::foreground()`.

- `-geometry WxH+X+Y`
  Sets the initial window position and size according to the standard X geometry string.

- `-iconic`
  Iconifies the window using `Fl_Window::iconize()`.

- `-kbd` and `-nokbd`
  Enables or disables visible keyboard focus for non-text widgets using `Fl::visible_focus()`.

- `-name string`
  Sets the window class using `Fl_Window::xclass()`.

- `-scheme string`
  Sets the widget scheme using `Fl::scheme()`.

- `-title string`
  Sets the window title using `Fl_Window::label()`.

- `-tooltips` and `-notooltips`
  Enables or disables tooltips using `Fl_Tooltip::enable()`.

If your program requires other switches in addition to the standard FLTK options, you will need to pass your own argument handler to `Fl::args(int,char **,int&,Fl_Args_Handler)` explicitly.

### 33.3.3.10 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.
33.3.3.11  args() [2/2]

int Fl::args(
    int argc,
    char ** argv,
    int & i,
    Fl_Args_Handler cb = 0)  [static]

Parse command line switches using the cb argument handler.
Returns 0 on error, or the number of words processed.

FLTK provides this as an entirely optional command line switch parser. You don't have to call it if you don't want to.
Everything it can do can be done with other calls to FLTK.
To use the switch parser, call Fl::args(...) near the start of your program. This does not open the display, instead
switches that need the display open are stashed into static variables. Then you must display your first window by
calling window->show(argc,argv), which will do anything stored in the static variables.
Providing an argument handler callback cb lets you define your own switches. It is called with the same argc and
argv, and with i set to the index of the switch to be processed. The cb handler should return zero if the switch is
unrecognized, and not change i. It should return non-zero to indicate the number of words processed if the switch
is recognized, i.e. 1 for just the switch, and more than 1 for the switch plus associated parameters. i should be
incremented by the same amount.
The cb handler is called before any other tests, so you can also override any standard FLTK switch (this is why
FLTK can use very short switches instead of the long ones all other toolkits force you to use). See Fl::arg() for
descriptions of the standard switches.

On return i is set to the index of the first non-switch. This is either:

- The first word that does not start with '-'.
- The word '-' (used by many programs to name stdin as a file)
- The first unrecognized switch (return value is 0).
- argc

The return value is i unless an unrecognized switch is found, in which case it is zero. If your program takes no
arguments other than switches you should produce an error if the return value is less than argc.
A usage string is displayed if Fl::args() detects an invalid argument on the command-line. You can change the
message by setting the Fl::help pointer.
A very simple command line parser can be found in examples/howto-parse-args.cxx
The simpler Fl::args(int argc, char **argv) form is useful if your program does not have command line switches of
its own.

33.3.3.12  background()

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.

33.3.3.13  background2()

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.
33.3.3.14 box_border_radius_max() [1/2]

static int Fl::box_border_radius_max ( ) [inline], [static]
Get the maximum border radius of all "rounded" boxtypes in pixels.
Since
1.4.0

33.3.3.15 box_border_radius_max() [2/2]

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

33.3.3.16 box_color()

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.

33.3.3.17 box_dh()

int Fl::box_dh ( Fl_Boxtype t ) [static]
Returns the height offset for the given boxtype.
See also
box_dy().

33.3.3.18 box_dw()

int Fl::box_dw ( Fl_Boxtype t ) [static]
Returns the width offset for the given boxtype.
See also
box_dy().
33.3.3.19  box_dx()

int Fl::box_dx (  
    Fl_Boxtype t  )  [static]
Returns the X offset for the given boxtype.
See also
    box_dy()

33.3.3.20  box_dy()

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 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():
    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().

33.3.3.21  box_shadow_width() [1/2]

static int Fl::box_shadow_width ( )  [inline],  [static]
Get the box shadow width of all "shadow" boxtypes in pixels.
Since
    1.4.0

33.3.3.22  box_shadow_width() [2/2]

static void Fl::box_shadow_width (  
    int W  )  [inline],  [static]
Set the box shadow width of all "shadow" boxtypes in pixels.
Must be at least 1, default = 3. There is no upper limit.
Since
    1.4.0

33.3.3.23  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).
### 33.3.3.24 display()

```cpp
void Fl::display (const char * d) [static]
```

Sets the X or Wayland display to use for all windows. This sets the environment variable $DISPLAY or $WAYLAND_DISPLAY to the passed string, so this only works before you show() the first window or otherwise open the display. This does nothing on other platforms.

### 33.3.3.25 dnd_text_ops() [1/2]

```cpp
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.

### 33.3.3.26 dnd_text_ops() [2/2]

```cpp
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.

### 33.3.3.27 draw_box_active()

```cpp
int Fl::draw_box_active () [static]
```

Determines if the currently drawn box is active or inactive. If inactive, the box color should be changed to the inactive color.

See also

```
Fl::box_color(Fl_Color c)
```

### 33.3.3.28 draw_GL_text_with_textures() [1/2]

```cpp
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
```

### 33.3.3.29 draw_GL_text_with_textures() [2/2]

```cpp
static void Fl::draw_GL_text_with_textures (int val) [inline], [static]
```

Sets whether OpenGL uses textures to draw all text. By default, FLTK draws OpenGL text using textures, if the necessary hardware support is available. Call `Fl::draw_GL_text_with_textures(0)` once in your program before the first call to `gl_font()` to have FLTK draw instead OpenGL text using a legacy, platform-dependent procedure. It's recommended not to deactivate textures under the MacOS platform because the MacOS legacy procedure is extremely rudimentary.
Parameters

| val | use 0 to prevent FLTK from drawing GL text with textures |

See also

```
gl_texture_pile_height(int max)
```

Version

1.4.0

### 33.3.3.30 flush()

```cpp
void Fl::flush () [static]
```

Causes all the windows that need it to be redrawn and graphics forced out through the pipes. This is what `wait()` does before looking for events.

Note: in multi-threaded applications you should only call `Fl::flush()` from the main thread. If a child thread needs to trigger a redraw event, it should instead call `Fl::awake()` to get the main thread to process the event queue.

### 33.3.3.31 get_system_colors()

```cpp
void Fl::get_system_colors () [static]
```

Read the user preference colors from the system and use them to call `Fl::foreground()`, `Fl::background()`, and `Fl::background2()`.

This is done by `Fl_Window::show(argc, argv)` before applying the `-fg` and `-bg` switches.

On X this reads some common values from the Xdefaults database. KDE users can set these values by running the “krdb” program, and newer versions of KDE set this automatically if you check the “apply style to other X programs” switch in their control panel.

### 33.3.3.32 gl_visual()

```cpp
static int Fl::gl_visual (  
    int ,  
    int * a list = 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.

### 33.3.3.33 has_timeout()

```cpp
int Fl::has_timeout (  
    Fl_Timeout_Handler cb,  
    void * data = 0 ) [static]
```

Returns true if the timeout exists and has not been called yet.

Both arguments `cb` and `data` must match with at least one timer in the queue of active timers to return true (1). Note

It is a known inconsistency that `Fl::has_timeout()` does not use the `data` argument as a wildcard (match all) if it is zero (NULL) which `Fl::remove_timeout()` does. This is so for backwards compatibility with FLTK 1.3.x. Therefore using 0 (zero, NULL) as the timeout `data` value is discouraged unless you’re sure that you don’t need to use `Fl::has_timeout(callback, (void *)0);` or `Fl::remove_timeout(callback, (void *)0);`.

---

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>Timer callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>User data</td>
</tr>
</tbody>
</table>

Returns

whether the timer was found in the queue

Return values

<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>

33.3.3.34 hide_all_windows()

void Fl::hide_all_windows ( ) [static]

Hide all visible window to make FLTK leave Fl::run().

Fl::run() will run as long as there are visible windows. Call hide_all_windows() will hide all windows, effectively terminating the Fl::run() loop.

See also

Fl::run()

33.3.3.35 is_scheme()

static int Fl::is_scheme (const char * name ) [inline], [static]

Returns whether the current scheme is the given name.

This is a fast inline convenience function to support scheme-specific code in widgets, e.g. in their draw() methods, if required.

Use a valid scheme name, not NULL (although NULL is allowed, this is not a useful argument - see below).

If Fl::scheme() has not been set or has been set to the default scheme ("none" or "base"), then this will always return 0 regardless of the argument, because Fl::scheme() is NULL in this case.

Note

The stored scheme name is always lowercase, and this method will do a case-sensitive compare, so you must provide a lowercase string to return the correct value. This is intentional for performance reasons.

Example:

if (Fl::is_scheme("gtk+")) { your_code_here(); }

Parameters

| in | name | lowercase string of requested scheme name. |

Returns

1 if the given scheme is active, 0 otherwise.

See also

Fl::scheme(const char *name)
33.3.3.36 menu_linespacing() [1/2]

int Fl::menu_linespacing ( ) [static]

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

33.3.3.37 menu_linespacing() [2/2]

void Fl::menu_linespacing ( int H ) [static]

Sets the default line spacing used by menus.

Default is 4.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>H</th>
<th>The new default line spacing between menu items, in pixels.</th>
</tr>
</thead>
</table>

33.3.3.38 now()

Fl_Timestamp Fl::now ( ) [static]

Set a time stamp at this point in time.

The time stamp is an opaque type and does not represent the time of day or some time and date in the calendar. It is used with Fl::seconds_between() and Fl::seconds_since() to measure elapsed time.

```c
Fl_Timestamp start = Fl::now();
// do something
double s = Fl::seconds_since(start);
printf("That operation took %g seconds\n", s);
```

Depending on the system the resolution may be milliseconds or microseconds. Under certain conditions (particularly on Windows) the value in member sec may wrap around and does not represent a real time (maybe runtime of the system). Function seconds_since() below uses this to subtract two timestamps which is always a correct delta time with milliseconds or microseconds resolution.

**Todo** Fl::system_driver()::gettime() was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses gettimeofday() with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: winmm.lib (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

Returns

this moment in time as an opaque time stamp

See also

Fl::seconds_since(Fl_Timestamp& then) Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back)

33.3.3.39 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:

```cpp
if ( Fl::option(Fl::OPTION_ARROW_FOCUS) )
    { ..on.. }
else
    { ..off.. }
```

Note

As of FLTK 1.3.0, options can be managed within fluid, using the menu Edit/Global FLTK Settings.

Parameters

<table>
<thead>
<tr>
<th>opt</th>
<th>which option</th>
</tr>
</thead>
</table>

Returns

true or false

See also

enum Fl::Fl_Option

Fl::option(Fl::Option, bool)

Since

FLTK 1.3.0

33.3.3.40 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:

```cpp
Fl::option(Fl::OPTION_ARROW_FOCUS, true);  // on
Fl::option(Fl::OPTION_ARROW_FOCUS, false);  // off
```

Parameters

<table>
<thead>
<tr>
<th>opt</th>
<th>which option</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>set to true or false</td>
</tr>
</tbody>
</table>

See also

enum Fl::Fl_Option

bool Fl::option(Fl::Option)

33.3.3.41 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.

### 33.3.3.42 program_should_quit() [1/2]

```c
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

### 33.3.3.43 program_should_quit() [2/2]

```c
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

### 33.3.3.44 readqueue()

```c
Fl_Widget * Fl::readqueue ( ) [static]
```

Reads the default callback queue and returns the first widget.
All Fl_Widgets that don’t have a callback defined use the default callback `static Fl_Widget::default_callback()` that puts a pointer to the widget in a queue. This method reads the oldest widget out of this queue.
The queue (FIFO) is limited (currently 20 items). If the queue overflows, the oldest entry (`Fl_Widget *`) is discarded.
Relying on the default callback and reading the callback queue with `Fl::readqueue()` is not recommended. If you need a callback, you should set one with `Fl_Widget::callback(Fl_Callback * cb, void *data)` or one of its variants.

See also

- `Fl_Widget::callback()`
- `Fl_Widget::callback(Fl_Callback * cb, void *data)`
- `Fl_Widget::default_callback()`

### 33.3.3.45 ready()

```c
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()`).
33.3.3.46 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*)

33.3.3.47 reload_scheme()

int Fl::reload_scheme ( ) [static]
Called internally when setting a new scheme according to scheme name.
Loads or reloads the current scheme selection.

Returns
    Always 1 (this may change in the future)

See void Fl::scheme(const char *name)

33.3.3.48 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.

33.3.3.49 remove_timeout()

void Fl::remove_timeout ( 
        FL_Timeout_Handler cb,
        void * data = 0 ) [static]
Removes a timeout callback from the timer queue.
This method removes all matching timeouts, not just the first one. This may change in the future.
If the data argument is NULL (the default!) only the callback cb must match, i.e. all timer entries with this callback are removed.
It is harmless to remove a timeout callback that no longer exists.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>cb</th>
<th>Timer callback to be removed (must match)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data</td>
<td>Wildcard if NULL (default), must match otherwise</td>
</tr>
</tbody>
</table>

33.3.3.50 repeat_timeout()

void Fl::repeat_timeout ( 
        double time,
        FL_Timeout_Handler cb,
        void * data = 0 ) [static]
Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
You should call this method only inside a timeout callback of the same or a logically related timer from whose expiration time the new timeout shall be scheduled. Otherwise the timing accuracy can't be improved and the exact behavior is undefined.
If you call this outside a timeout callback the behavior is the same as `Fl::add_timeout()`.

Example: The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```c
#include <FL/Fl.H>
#include <stdio.h>

void callback(void *) {
    printf("TICK\n");
    Fl::repeat_timeout(1.0, callback); // retrigger timeout
}

int main() {
    Fl_Window win(100, 100);
    win.show();
    Fl::add_timeout(1.0, callback); // set up first timeout
    return Fl::run();
}
```

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>delta time in seconds until the timer expires</td>
</tr>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

#### 33.3.3.51 run()

```c
int Fl::run ( ) [static]
```

Calls `Fl::wait()` repeatedly as long as any windows are displayed. When all the windows are closed it returns zero (supposedly it would return non-zero on any errors, but FLTK calls exit directly for these). A normal program will end `main()` with return `Fl::run()`.

**Note**

`Fl::run()` and `Fl::wait()` (but not `Fl::wait(double)`) both return when all FLTK windows are closed. Therefore, a MacOS FLTK application possessing `Fl_Sys_Menu_Bar` items able to create new windows and expected to keep running without any open window cannot use these two functions. One solution is to run the event loop as follows:

```c
while (!Fl::program_should_quit()) Fl::wait(1e20);
```

#### 33.3.3.52 scheme()

```c
int Fl::scheme ( const char * s ) [static]
```

Sets the current widget scheme.

NULL will use the scheme defined in the FLTK_SCHEME environment variable or the scheme resource under X11. Otherwise, any of the following schemes can be used:

- "none" - This is the default look-n-feel which resembles old Windows (95/98/Me/NT/2000) and old GTK/KDE
- "base" - This is an alias for "none"
- "plastic" - This scheme is inspired by the Aqua user interface on macOS
- "gtk+" - This scheme is inspired by the Red Hat Bluecurve theme
- "gleam" - This scheme is inspired by the Clearlooks Glossy scheme. (Colin Jones and Edmanuel Torres)
- "oxy" - This is a subset of Dmitrij K's oxy scheme (STR 2675, 3477)

If the given scheme name is unknown, the default scheme will be used.

Setting the scheme (name) is case insensitive, but the stored scheme name will always be lowercase and `Fl::scheme()` will return this lowercase name or NULL if no scheme or the default scheme ("none" or "base") was set.
Parameters

| in  | s          | Scheme name of NULL |

Returns

Current scheme name or NULL

Return values

| NULL | if the scheme has not been set or is the default scheme |

See also

Fl::is_scheme()

### 33.3.3.53 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.

### 33.3.3.54 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

| in  | W          | The new default size for widget scrollbars, in pixels |

### 33.3.3.55 seconds_between()

```cpp
double Fl::seconds_between ( Fl_Timestamp & back, Fl_Timestamp & further_back ) [static]
```

Return the time in seconds between two time stamps.

Parameters

| in   | back         | a previously taken time stamp |
| in   | further_back | an even earlier time stamp    |

Returns

elapsed seconds and fractions of a second
33.3.3.56  seconds_since()

double Fl::seconds_since ( Fl_Timestamp & then ) [static]

Return the time in seconds between now and a previously taken time stamp.

Parameters

- `then`: a previously taken time stamp

Returns

elapsed seconds and fractions of a second

See also

- Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back), Fl::now()

33.3.3.57  set_box_color()

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)

33.3.3.58  set_idle()

static void Fl::set_idle ( Fl_Old_Idle_Handler cb ) [inline], [static]

Sets an idle callback.

**Deprecated**  This method is obsolete - use the add_idle() method instead.

33.3.3.59  ticks_between()

long Fl::ticks_between ( Fl_Timestamp & back, Fl_Timestamp & further_back ) [static]

Return the time in ticks (60Hz) between two time stamps.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>back</th>
<th>a previously taken time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>further_back</td>
<td>an even earlier time stamp</td>
</tr>
</tbody>
</table>

Returns

elapsed ticks in 60th of a second

See also

Fl::ticks_since(Fl_Timestamp& then), Fl::now()

33.3.3.60  ticks_since()

long Fl::ticks_since (Fl_Timestamp & then) [static]

Return the time in ticks (60Hz) between now and a previously taken time stamp. Ticks are a convenient way to time animations 'per frame'. Even though modern computers use all kinds of screen refresh rates, 60Hz is a very good base for animation that is typically shown in user interface graphics.

Parameters

| in    | then | a previously taken time stamp |

Returns

elapsed ticks in 60th of a second

See also

Fl::ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back), Fl::now()

33.3.3.61  use_high_res_GL() [1/2]

static int Fl::use_high_res_GL ( ) [inline], [static]

returns whether GL windows should be drawn at high resolution on Apple computers with retina displays. Default is no.

Version

1.3.4

33.3.3.62  use_high_res_GL() [2/2]

static void Fl::use_high_res_GL (int val) [inline], [static]

sets whether GL windows should be drawn at high resolution on Apple computers with retina displays

Version

1.3.4
33.3.3.63  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.

33.3.3.64  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.

33.3.3.65  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.

33.3.3.66  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).

33.3.3.67  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.
### 33.3.3.68 wait() [2/2]

```c
double Fl::wait ( double time_to_wait ) [static]
```

Waits a maximum of `time_to_wait` seconds or until "something happens".
See `Fl::wait()` for the description of operations performed when "something happens".

**Returns**

Always 1 on Windows. Otherwise, it is positive if an event or fd happens before the time elapsed. It is zero if nothing happens. It is negative if an error occurs (this will happen on X11 if a signal happens).

### 33.3.4 Member Data Documentation

### 33.3.4.1 help

```c
const char *const Fl::help = helpmsg+13 [static]
```

Usage string displayed if Fl::args() detects an invalid argument.
This may be changed to point to customized text at run-time.

### 33.3.4.2 idle

```c
void(* Fl::idle)() [static]
```

The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
This is now used as part of a higher level system allowing multiple idle callback functions to be called.

See also

- `add_idle()`, `remove_idle()`

The documentation for this class was generated from the following files:

- Fl.H
- Fl.cxx
- Fl_abort.cxx
- Fl_add_idle.cxx
- Fl_arg.cxx
- fl_boxtype.cxx
- Fl_Cairo.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_Timeout.cxx
- Fl_visual.cxx
- Fl_Widget.cxx
- Fl_Window.cxx
- screen_xywh.cxx
33.4 **Fl_Adjuster Class Reference**

The **Fl_Adjuster** widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

```cpp
#include <Fl_Adjuster.H>
```

Inheritance diagram for Fl_Adjuster:

```
Fl_Widget
   |
   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 () FL_OVERRIDE**

  Draws the widget.

- **int handle (int) FL_OVERRIDE**

  Handles the specified event.

- **void value_damage () FL_OVERRIDE**

  Asks for partial redraw.

**Additional Inherited Members**

33.4.1 **Detailed Description**

The **Fl_Adjuster** widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

When you press a button and drag to the right the value increases. When you drag to the left it decreases. The largest button adjusts by 100 * 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.
33.4.2 Constructor & Destructor Documentation

33.4.2.1 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.

33.4.3 Member Function Documentation

33.4.3.1 draw()

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:
Fl_Widget ∗s = &scrollbar;  // scrollbar is an embedded Fl_Scrollbar
s->draw();  // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

33.4.3.2 handle()

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</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.
33.4.3.3 soft() [1/2]

```cpp
int Fl_Adjuster::soft ( ) const [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

33.4.3.4 soft() [2/2]

```cpp
void Fl_Adjuster::soft ( int s ) [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

33.4.3.5 value_damage()

```cpp
void Fl_Adjuster::value_damage ( ) [protected], [virtual]
```

Asks for partial redraw.
Reimplemented from Fl_Valuator.
The documentation for this class was generated from the following files:
- Fl_Adjuster.H
- Fl_Adjuster.cxx

33.5 Fl_Anim_GIF_Image Class Reference

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF\textsuperscript{SM} images.

Inheritance diagram for Fl_Anim_GIF_Image:

```
Fl_Image
 ↓
Fl_Pixmap
 ↓
Fl_GIF_Image
 ↓
Fl_Anim_GIF_Image
```

Public Types

- enum Flags {
  DONT_START = 1 ,
  DONT_RESIZE_CANVAS = 2 ,
  DONT_SET_AS_IMAGE = 4 ,
  OPTIMIZE_MEMORY = 8 ,
  LOG_FLAG = 64 ,
  DEBUG_FLAG = 128 }

When opening an Fl_Anim_GIF_Image there are some options that can be passed in a flags value.

Public Member Functions

- Fl_Widget * canvas ( ) const
  Gets the current widget, that is used to display the frame images.
- void canvas (Fl_Widget *canvas, unsigned short flags=0)
  Link the image back to a widget for automated animation.
- int canvas_h ( ) const
  Return the height of the animation canvas.
• int canvas_w () const
  Return the width of the animation canvas.
• void color_average (Fl_Color c, float i) FL_OVERRIDE
  Applies a color average to all frames.
• Fl_Image * copy () const
• Fl_Image * copy (int W, int H) const FL_OVERRIDE
  Copy and resize the animation frames.
• int debug () const
• void delay (int frame, double delay)
  Set the delay of frame [0-frames() -1] in seconds.
• double delay (int frame_) const
  Return the delay of frame [0-frames() -1] in seconds.
• void desaturate () FL_OVERRIDE
  Desaturate to all frames of the animation.
• void draw (int x, int y, int w, int h, int cx=0, int cy=0) FL_OVERRIDE
  Draw the current frame of the animation.
• Fl_Anim_GIF_Image ()
  Create an empty animated GIF image shell.
• Fl_Anim_GIF_Image (const char *filename, Fl_Widget *canvas=0, unsigned short flags=0)
  Load an animated GIF image from a file.
• Fl_Anim_GIF_Image (const char *imagename, const unsigned char *data, const size_t length, Fl_Widget *canvas=0, unsigned short flags=0)
  Load an animated GIF image from memory.
• int frame () const
  Return the current frame.
• void frame (int frame)
  Set the current frame.
• int frame_h (int frame) const
  Return the frame dimensions of a frame.
• bool frame_uncache () const
  Return the active frame_uncache() setting.
• void frame_uncache (bool uncache)
  Use frame_uncache() to set or forbid frame image uncaching.
• int frame_w (int frame) const
  Return the frame dimensions of a frame.
• int frame_x (int frame) const
  Return the frame position of a frame.
• int frame_y (int frame) const
  Return the frame position of a frame.
• int frames () const
  Get the number of frames in the animation.
• Fl_Image * image () const
  Return the current frame image.
• Fl_Image * image (int frame) const
  Return the image of the given frame index.
• bool is_animated () const
  Check if this is a valid animation with more than one frame.
• bool load (const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)
  Load an animation from a file or from a memory block.
• const char * name () const
  Return the name of the played file as specified in the constructor.
• bool next()
  
  Show the next frame if the animation is stopped.

• bool playing() const
  
  Return if the animation is currently running or stopped.

• Fl_Anim_GIF_Image & resize (double scale)
  
  Resizes the image to the specified size, replacing the current image.

• Fl_Anim_GIF_Image & resize (int w, int h)
  
  Resizes the image to the specified size, replacing the current image.

• double speed() const
  
  Get the animation speed factor.

• void speed (double speed)
  
  Set the animation speed factor.

• bool start()
  
  The start() method (re-)starts the playing of the frames.

• bool stop()
  
  The stop() method stops the playing of the frames.

• void uncache () FL_OVERRIDE
  
  Uncache all cached image data now.

• bool valid () const
  
  Check if animation is valid.

• ~Fl_Anim_GIF_Image () FL_OVERRIDE
  
  Release the image and all cached data.

Static Public Member Functions

• static int frame_count (const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)
  
  Get the number of frames in a GIF file or in a GIF compressed data block.

Static Public Attributes

• static bool loop = true
  
  The loop flag can be used to (dis-)allow loop count.

• static double min_delay = 0.
  
  The min_delay value can be used to set a minimum value for the frame delay for playback.

Protected Member Functions

• void clear_frames()

• bool next_frame()

• void on_extension_data (Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE

• void on_frame_data (Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE

• void scale_frame()

• void set_frame()

• void set_frame (int frame)

Static Protected Member Functions

• static void cb_animate (void *d)
Additional Inherited Members

33.5.1 Detailed Description

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF<sup>SM</sup> images. The class loads all images contained in the file and animates them by cycling through them as defined by the delay times in the image file.

The user must supply an FLTK widget as "container" in order to see the animation by specifying it in the constructor or later using the canvas() method.

33.5.2 Member Enumeration Documentation

### Flags

- **DONT_START**: This flag indicates to the loader that it should not start the animation immediately after successful load, which is the default. It can be started later using the `start()` method.
- **DONT_RESIZE_CANVAS**: This flag indicates to the loader that it should not resize the canvas widget of the animation to the dimensions of the animation, which is the default. Needed for special use cases.
- **DONT_SET_AS_IMAGE**: This flag indicates to the loader that it should not set the animation as `image()` member of the canvas widget, which is the default. Needed for special use cases.
- **OPTIMIZE_MEMORY**: Often frames change just a small area of the animation canvas. This flag indicates to the loader to try using less memory by storing frame data not as canvas-sized images but use the sizes defined in the GIF file. The drawbacks are higher cpu usage during playback and maybe minor artefacts when resized.
- **LOG_FLAG**: This flag can be used to print informations about the decoding process to the console.
- **DEBUG_FLAG**: This flag can be used to print even more informations about the decoding process to the console.

33.5.3 Constructor & Destructor Documentation

#### Fl_Anim_GIF_Image() [1/2]

Fl_Anim_GIF_Image::Fl_Anim_GIF_Image (  
    const char * filename,  
    Fl_Widget * canvas = 0,  
    unsigned short flags = 0  
)

Load an animated GIF image from a file. This constructor creates an animated image form a GIF-formatted file. Optionally it applies the canvas() method after successful load. If **DONT_START** is not specified in the **flags** parameter it calls `start()` after successful load.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filename</code></td>
<td>path and name of GIF file in the file system</td>
</tr>
<tr>
<td><code>canvas</code></td>
<td>a widget that will show and animate the GIF, or NULL</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>see Flags for details, or 0</td>
</tr>
</tbody>
</table>
33.5 Fl_Anim_GIF_Image Class Reference

Note
The GIF image must be decoupled from the canvas by calling myGif->canvas(NULL); before deleting the canvas.

33.5.3.2 Fl_Anim_GIF_Image() [2/2]

Fl_Anim_GIF_Image::Fl_Anim_GIF_Image (const char * imagename, const unsigned char * data, const size_t length, Fl_Widget * canvas = 0, unsigned short flags = 0 )

Load an animated GIF image from memory.
This constructor creates an animated image form a GIF-formatted block in memory. Optionally it applies the canvas() method after successful load. If DONT_START is not specified in the flags parameter it calls start() after successful load.
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.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagename</td>
<td>a name given to this image or NULL</td>
</tr>
<tr>
<td>data</td>
<td>pointer to the start of the GIF image in memory</td>
</tr>
<tr>
<td>length</td>
<td>length of the GIF image in memory</td>
</tr>
<tr>
<td>canvas</td>
<td>a widget that will show and animate the GIF, or NULL</td>
</tr>
<tr>
<td>flags</td>
<td>see Flags for details, or 0</td>
</tr>
</tbody>
</table>

Note
The GIF image must be decoupled from the canvas by calling myGif->canvas(NULL); before deleting the canvas.

33.5.3.3 ~Fl_Anim_GIF_Image()

Fl_Anim_GIF_Image::~Fl_Anim_GIF_Image ( )
Release the image and all cached data.
Also removes the animation timer.

33.5.4 Member Function Documentation

33.5.4.1 canvas() [1/2]

Fl_Widget * Fl_Anim_GIF_Image::canvas ( ) const
Gets the current widget, that is used to display the frame images.

Returns
a pointer to a widget
33.5.4.2 canvas() [2/2]

```cpp
void Fl_Anim_GIF_Image::canvas (  
    Fl_Widget * canvas,  
    unsigned short flags = 0 )
```

Link the image back to a widget for automated animation.
This method sets current widget, that is used to display the frame images. The flags parameter specifies whether
the canvas widget is resized to the animation dimensions and/or its image() method will be used to set the current
frame image during animation.

**Parameters**

| in | canvas | a pointer to the widget that will show the animation |
| in | flags  | see Flags |

**Note**

The GIF image must be decoupled from the canvas by calling myGif->canvas(NULL); before deleting
the canvas.

33.5.4.3 canvas_h()

```cpp
int Fl_Anim_GIF_Image::canvas_h ( ) const
```

Return the height of the animation canvas.

**Returns**

the width in pixel units

33.5.4.4 canvas_w()

```cpp
int Fl_Anim_GIF_Image::canvas_w ( ) const
```

Return the width of the animation canvas.

**Returns**

the width in pixel units

33.5.4.5 color_average()

```cpp
void Fl_Anim_GIF_Image::color_average (  
    Fl_Color c,  
    float i ) [virtual]
```

Applies a color average to all frames.
The color_average() method averages the colors in the image with the provided FLTK color value.

**Parameters**

| in | c | blend color |
| in | i | a value between 0.0 and 1.0 where 0 results in the blend color, and 1 returns the original image |

Reimplemented from Fl_Image.
33.5.4.6 copy()  

Fl_Anim_GIF_Image::copy (  
  int W,  
  int H ) const [virtual]  

Copy and resize the animation frames.  
The virtual copy() method makes a copy of the animated image and resizes all of its frame images to W x H using  
the current resize method.  

Parameters  

| in | W,H | new size in FLTK pixel units |

Returns  

the resized copy of the animation  

Reimplemented from Fl_Image.  

33.5.4.7 delay() [1/2]  

void Fl_Anim_GIF_Image::delay (  
  int frame,  
  double delay )  

Set the delay of frame [0−frames() −1] in seconds.  

Parameters  

| in | frame | index into frame list |
| in | delay | to next frame in seconds |

33.5.4.8 delay() [2/2]  

double Fl_Anim_GIF_Image::delay (  
  int frame ) const  

Return the delay of frame [0−frames() −1] in seconds.  

Parameters  

| in | frame | index into frame list |

Returns  

delay to next frame in seconds  

33.5.4.9 desaturate()  

void Fl_Anim_GIF_Image::desaturate ( ) [virtual]  

Desaturate to all frames of the animation.  

Reimplemented from Fl_Image.  

33.5.4.10 draw()  

void Fl_Anim_GIF_Image::draw (
Draw the current frame of the animation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, w, h</td>
<td>int</td>
<td>target rectangle</td>
</tr>
<tr>
<td>cx, cy</td>
<td>int</td>
<td>source offset</td>
</tr>
</tbody>
</table>

Reimplemented from `Fl_Image`.

33.5.4.11 frame() [1/2]

```cpp
int Fl_Anim_GIF_Image::frame ( ) const
```

Return the current frame.

Returns

- the current frame index in the range for 0 to `frames()-1`
- -1 if the image has no frames.

33.5.4.12 frame() [2/2]

```cpp
void Fl_Anim_GIF_Image::frame ( int frame )
```

Set the current frame.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>int</td>
<td>index into list of frames</td>
</tr>
</tbody>
</table>

33.5.4.13 frame_count()

```cpp
int Fl_Anim_GIF_Image::frame_count ( const char * name, const unsigned char * imgdata = NULL, size_t imglength = 0 ) [static]
```

Get the number of frames in a GIF file or in a GIF compressed data block.

The static `frame_count()` method is just a convenience method for getting the number of images (frames) stored in a GIF file.

As this count is not readily available in the GIF header, the whole GIF file has be parsed (which is done here by using a temporary `Fl_Anim_GIF_Image` object for simplicity). So this call may be slow with large files.

If `imgdata` is NULL, the image will be read from the file. Otherwise, it will be read from memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>const char</td>
<td>path and name of GIF file in the file system, ignored when reading from memory</td>
</tr>
<tr>
<td>imgdata</td>
<td>const unsigned char</td>
<td>pointer to the start of the GIF image in memory, or NULL to read from a file</td>
</tr>
<tr>
<td>imglength</td>
<td>size_t</td>
<td>length of the GIF image in memory, or 0</td>
</tr>
</tbody>
</table>
33.5 Fl_Anim_GIF_Image Class Reference

Returns
the number of frames in the animation

33.5.4.14 frame_h()

int Fl_Anim_GIF_Image::frame_h (int frame) const

Return the frame dimensions of a frame. Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in  | frame | index into frame list |

Retruns
height in FLTK pixle units

33.5.4.15 frame_uncache() [1/2]

bool Fl_Anim_GIF_Image::frame_uncache ( ) const

Return the active frame_uncache() setting.

Returns
ture if caching is disabled

33.5.4.16 frame_uncache() [2/2]

void Fl_Anim_GIF_Image::frame_uncache (bool uncache)

Use frame_uncache() to set or forbid frame image uncaching. If frame uncaching is set, frame images are not offscreen cached for re-use and will be re-created every time they are displayed. This saves a lot of memory on the expense of cpu usage and should be carefully considered. Per default frame caching will be done.

Parameters

| in  | uncache | true to disable caching |

33.5.4.17 frame_w()

int Fl_Anim_GIF_Image::frame_w (int frame) const

Return the frame dimensions of a frame. Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in  | frame | index into frame list |
Returns

width in FLTK pixle units

33.5.4.18 frame_x()

```cpp
int Fl_Anim_GIF_Image::frame_x (int frame) const
```

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in   | frame       | index into frame list |

Returns

x position in FLTK pixle units

33.5.4.19 frame_y()

```cpp
int Fl_Anim_GIF_Image::frame_y (int frame) const
```

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

| in   | frame       | index into frame list |

Returns

y position in FLTK pixle units

33.5.4.20 frames()

```cpp
int Fl_Anim_GIF_Image::frames () const
```

Get the number of frames in the animation.

Returns

the number of frames

33.5.4.21 image()[1/2]

```cpp
Fl_Imag e * Fl_Anim_GIF_Image::image () const
```

Return the current frame image.

Returns

a pointer to the image or NULL if this is not an animation.
33.5.4.22 image() [2/2]

Fl_Image * Fl_Anim_GIF_Image::image ( int frame_ ) const

Return the image of the given frame index.

Parameters

| in  | frame_  | index into list of frames |

Returns

image data or NULL if the frame number is not valid.

33.5.4.23 is_animated()

bool Fl_Anim_GIF_Image::is_animated ( ) const

Check if this is a valid animation with more than one frame. The is_animated() method is just a convenience method for testing the valid flag and the frame count being greater 1.

Returns

true if the animation is valid and has multiple frames.

33.5.4.24 load()

bool Fl_Anim_GIF_Image::load ( const char * name, const unsigned char * imgdata = NULL, size_t imglength = 0 )

Load an animation from a file or from a memory block. The load() method is either used from the constructor to load the image from the given file, or to re-load an existing animation from another file.

Parameters

| in  | name  | path and name of GIF file in the file system, or the image name when reading from memory |
| in  | imgdata  | pointer to the start of the GIF image in memory, or NULL to read from a file |
| in  | imglength  | length of the GIF image in memory, or 0 |

Returns

true if the animation loaded correctly

33.5.4.25 name()

const char * Fl_Anim_GIF_Image::name ( ) const

Return the name of the played file as specified in the constructor. If read from a memory block, this returns the name of the animation.

Returns

pointer to a C string
33.5.4.26  next()

bool Fl_Anim_GIF_Image::next ( )
Show the next frame if the animation is stopped.

Returns
true if the animation has frames

33.5.4.27  on_extension_data()

void Fl_Anim_GIF_Image::on_extension_data ( Fl_GIF_Image::GIF_FRAME & f ) [protected], [virtual]
Reimplemented from Fl_GIF_Image.

33.5.4.28  on_frame_data()

void Fl_Anim_GIF_Image::on_frame_data ( Fl_GIF_Image::GIF_FRAME & f ) [protected], [virtual]
Reimplemented from Fl_GIF_Image.

33.5.4.29  playing()

bool Fl_Anim_GIF_Image::playing ( ) const [inline]
Return if the animation is currently running or stopped.

Returns
true if the animation is running

33.5.4.30  resize [1/2]

Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize ( double scale )
Resizes the image to the specified size, replacing the current image.
If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>rescale factor in relation to current size</td>
<td></td>
</tr>
</tbody>
</table>

33.5.4.31  resize [2/2]

Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize ( int w, int h )
Resizes the image to the specified size, replacing the current image.
If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>w,h</th>
</tr>
</thead>
<tbody>
<tr>
<td>new size of the animation frames</td>
<td></td>
</tr>
</tbody>
</table>
33.5.4.32 speed [1/2]

double Fl_Anim_GIF_Image::speed ( ) const
Get the animation speed factor.

Returns
the current speed factor

33.5.4.33 speed [2/2]

void Fl_Anim_GIF_Image::speed ( double speed )
Set the animation speed factor.
The speed() method changes the playing speed to speed x original speed. E.g. to play at half speed call it with 0.5, for double speed with 2.

Parameters
in speed floating point speed factor

33.5.4.34 start()

bool Fl_Anim_GIF_Image::start ( )
The start() method (re-)starts the playing of the frames.

Returns
true if the animation has frames

33.5.4.35 stop()

bool Fl_Anim_GIF_Image::stop ( )
The stop() method stops the playing of the frames.

Returns
true if the animation has frames

33.5.4.36 uncache()

void Fl_Anim_GIF_Image::uncache ( ) [virtual]
Uncache all cached image data now.
Re-implemented from Fl_Pixmap.
Reimplemented from Fl_Image.

33.5.4.37 valid()

bool Fl_Anim_GIF_Image::valid ( ) const
Check if animation is valid.
Returns
true if the class has successfully loaded and the image has at least one frame.

### 33.5.5 Member Data Documentation

#### 33.5.5.1 loop

```cpp
bool Fl_Anim_GIF_Image::loop = true [static]
```

The loop flag can be used to (dis-)allow loop count. If set (which is the default), the animation will be stopped after the number of repeats specified in the GIF file (typically this count is set to 'forever' anyway). If cleared the animation will always be 'forever', regardless of what is specified in the GIF file.

#### 33.5.5.2 min_delay

```cpp
double Fl_Anim_GIF_Image::min_delay = 0. [static]
```

The `min_delay` value can be used to set a minimum value for the frame delay for playback. This is to prevent CPU hogs caused by images with very low delay rates. This is a global value for all `Fl_Anim_GIF_Image` objects.

The documentation for this class was generated from the following files:

- `Fl_Anim_GIF_Image.H`
- `Fl_Anim_GIF_Image.cxx`

### 33.6 Fl_Bitmap Class Reference

The `Fl_Bitmap` class supports caching and drawing of mono-color (bitmap) images.

```cpp
#include <Fl_Bitmap.H>
```

Inheritance diagram for `Fl_Bitmap`:

```
Fl_XBM_Image
   `-- Fl_Bitmap
       `-- Fl_Image
```

#### Public Member Functions

- `Fl_Image * copy () const`
- `Fl_Image * copy (int W, int H) const FL_OVERRIDE`

  Creates a resized copy of the image.

- `void draw (int X, int Y)`
- `void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE`

  Draws the image to the current drawing surface with a bounding box.

- `Fl_Bitmap (const char * bits, int bits_length, int W, int H)`

  The constructors create a new bitmap from the specified bitmap data.

- `Fl_Bitmap (const uchar * bits, int bits_length, int W, int H)`

  The constructors create a new bitmap from the specified bitmap data.

- `Fl_Bitmap (const uchar * bits, int W, int H)`

  The constructors create a new bitmap from the specified bitmap data.
The constructors create a new bitmap from the specified bitmap data.

- void label(Fl_Menu_Item *m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.
- void label(Fl_Widget *w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache() FL_OVERRIDE
  If the image has been cached for display, delete the cache data.
- virtual ~Fl_Bitmap()
  The destructor frees all memory and server resources that are used by the bitmap.

Public 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

33.6.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images. Images are drawn using the current color.

33.6.2 Constructor & Destructor Documentation

33.6.2.1 Fl_Bitmap() [1/4]

Fl_Bitmap::Fl_Bitmap(
    const uchar *bits,
    int W,
    int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.

See also

Fl_Bitmap(const uchar *bits, int bits_length, int W, int H)

33.6.2.2 Fl_Bitmap() [2/4]

Fl_Bitmap::Fl_Bitmap(
    const char *bits,
    int W,
    int H ) [inline]

The constructors create a new bitmap from the specified bitmap data.

See also

Fl_Bitmap(const char *bits, int bits_length, int W, int H)
33.6.2.3 Fl_Bitmap() [3/4]

Fl_Bitmap::Fl_Bitmap (  
  const uchar ∗ bits,  
  int bits_length,  
  int W,  
  int H )

The constructors create a new bitmap from the specified bitmap data.  
If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>bitmap data, one pixel per bit, rows are rounded to the next byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
<tr>
<td>W</td>
<td>image width in pixels</td>
</tr>
<tr>
<td>H</td>
<td>image height in pixels</td>
</tr>
</tbody>
</table>

See also

Fl_Bitmap(const char ∗ bits, int bits_length, int W, int H), Fl_Bitmap(const uchar ∗ bits, int W, int H)

33.6.2.4 Fl_Bitmap() [4/4]

Fl_Bitmap::Fl_Bitmap (  
  const char ∗ bits,  
  int bits_length,  
  int W,  
  int H )

The constructors create a new bitmap from the specified bitmap data.  
If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>bitmap data, one pixel per bit, rows are rounded to the next byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
<tr>
<td>W</td>
<td>image width in pixels</td>
</tr>
<tr>
<td>H</td>
<td>image height in pixels</td>
</tr>
</tbody>
</table>

See also

Fl_Bitmap(const char ∗ bits, int bits_length, int W, int H), Fl_Bitmap(const uchar ∗ bits, int W, int H)

33.6.3 Member Function Documentation

33.6.3.1 copy()

Fl_Image ∗ Fl_Bitmap::copy (  
  int W,  
  int H ) const [virtual]

Creates a resized copy of the image.  
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().

For the new image the following equations are true:

- \( w() == \text{data}_w() == W \)
- \( h() == \text{data}_h() == H \)

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

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.

### 33.6.3.2 draw()

```cpp
void Fl_Bitmap::draw (  
    int X,  
    int Y,  
    int W,  
    int H,  
    int cx = 0,  
    int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box. Arguments \( X, Y, W, H \) specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the \( cx \) and \( cy \) arguments. In other words: fl_push_clip(X, Y, W, H) is applied, the image is drawn with its upper-left corner at \( X-cx, Y-cy \) and its own width and height, fl_pop_clip() is applied. Reimplemented from Fl_Image.

### 33.6.3.3 label() [1/2]

```cpp
void Fl_Bitmap::label (  
    Fl_Menu_Item * m ) [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

**Deprecated** Please use Fl_Menu_Item::image() instead.

Reimplemented from Fl_Image.

### 33.6.3.4 label() [2/2]

```cpp
void Fl_Bitmap::label (  
    Fl_Widget * widget ) [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.
**Deprecated** Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

### 33.6.3.5 uncache()

```cpp
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

## 33.7 Fl_BMP_Image Class Reference

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

### Include

```cpp
#include <Fl_BMP_Image.H>
```

Inheritance diagram for Fl_BMP_Image:

```
Fl_BMP_Image
    |     |
    |   FL_RGB_Image
    |        |
    |       FL_BMP_Image
    |           |
    |          FL_ICO_Image
```

### Public Member Functions

- **Fl_BMP_Image** (const char *filename)
  
  This constructor loads the named BMP image from the given BMP filename.

- **Fl_BMP_Image** (const char *imagename, const unsigned char *data, const long length=-1)
  
  This constructor loads a BMP image from memory.

### Protected Member Functions

- void **load_bmp_** (class Fl_Image_Reader &rdr, int ico_height=0, int ico_width=0)

### Additional Inherited Members

#### 33.7.1 Detailed Description

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.
33.7.2.1 Fl_BMP_Image() [1/2]

Fl_BMP_Image::Fl_BMP_Image (const char * filename )

This constructor loads the named BMP image from the given BMP filename.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not
be opened or read, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image
could not be loaded for another reason.

Parameters

| in  | filename | a full path and name pointing to a BMP file. |

See also

Fl_BMP_Image::Fl_BMP_Image(const char * imagename, const unsigned char *data, const long length = -1);

33.7.2.2 Fl_BMP_Image() [2/2]

Fl_BMP_Image::Fl_BMP_Image (const char * imagename,
const unsigned char * data,
const long length = -1 )

This constructor loads a BMP image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way
to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to
the list of shared images and will be available by that name.
The destructor frees all memory and server resources that are used by the image.
The (new and optional) third parameter length should be used so buffer overruns (i.e. truncated images) can be
checked. See note below.
If length is not used

• it defaults to -1 (unlimited size)
• buffer overruns will not be checked.

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

| in  | imagename | A name given to this image or NULL |
| in  | data      | Pointer to the start of the BMP image in memory. |
| in  | length    | Length of the BMP image in memory. |

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:
33.8  **Fl_Box Class Reference**

This widget simply draws its box, and possibly its label.

```c
#include <Fl_Box.H>
```

Inheritance diagram for Fl_Box:

```
Fl_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)**

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

### Protected Member Functions

- **void draw () FL_OVERRIDE**
  
  Draws the widget.

### Additional Inherited Members

33.8.1 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.

33.8.2 Constructor & Destructor Documentation

33.8.2.1 Fl_Box()

```c
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.
33.8.3 Member Function Documentation

33.8.3.1 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:
```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

33.8.3.2 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 event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.

The documentation for this class was generated from the following files:

- Fl_Box.H
- Fl_Box.cxx

33.9 Fl_Browser Class Reference

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.

```cpp
#include <Fl_Browser.H>
```

Inheritance diagram for Fl_Browser:
Public Types

- enum Fl_Line_Position { TOP, BOTTOM, MIDDLE }
  
  *For internal use only?*

Public Member Functions

- void add (const char *newtext, void *d=0)
  
  *Adds a new line to the end of the browser.*

- void bottomline (int line)
  
  *Scrolls the browser so the bottom item in the browser is showing the specified line.*

- void clear ()
  
  *Removes all the lines in the browser.*

- char column_char () const
  
  *Gets the current column separator character.*

- void column_char (char c)
  
  *Sets the column separator to c.*

- const int * column_widths () const
  
  *Gets the current column width array.*

- void column_widths (const int *arr)
  
  *Sets the current array to arr.*

- void * data (int line) const
  
  *Returns the user data() for specified line.*

- void data (int line, void *d)
  
  *Sets the user data for specified line to d.*

- void display (int line, int val=1)
  
  *For back compatibility.*

- int displayed (int line) const
  
  *Returns non-zero if line has been scrolled to a position where it is being displayed.*

- Fl_Browser (int X, int Y, int W, int H, const char *L=0)
  
  *The constructor makes an empty browser.*

- char format_char () const
  
  *Gets the current format code prefix character, which by default is '@'.*

- void format_char (char c)
  
  *Sets the current format code prefix character to c.*

- void hide () FL_OVERRIDE
  
  *Hides the entire Fl_Browser widget – opposite of show().*

- void hide (int line)
  
  *Makes line invisible, preventing selection by the user.*
• \texttt{Fl\_Image \* icon (int line) const}
  
  Returns the icon currently defined for line.
• void \texttt{icon (int line, Fl\_Image \*icon)}
  
  Set the image icon for line to the value icon.
• void \texttt{insert (int line, const char \*newtext, void \*d=0)}
  
  Insert a new entry whose label is newtext above given line, optional data d.
• void \texttt{lineposition (int line, Fl\_Line\_Position pos)}
  
  Updates the browser so that line is shown at position pos.
• int \texttt{load (const char \*filename)}
  
  Clears the browser and reads the file, adding each line from the file to the browser.
• void \texttt{make\_visible (int line)}
  
  Make the item at the specified line visible().
• void \texttt{middleline (int line)}
  
  Scrolls the browser so the middle item in the browser is showing the specified line.
• void \texttt{move (int to, int from)}
  
  Line from is removed and reinserted at to.
• void \texttt{remove (int line)}
  
  Remove entry for given line number, making the browser one line shorter.
• void \texttt{remove\_icon (int line)}
  
  Removes the icon for line.
• void \texttt{replace (int a, const char \*b)}
  
  For back compatibility only.
• int \texttt{select (int line, int val=1)}
  
  Sets the selection state of the item at line to the value val.
• int \texttt{selected (int line) const}
  
  Returns 1 if specified line is selected, 0 if not.
• void \texttt{show () \texttt{FL\_OVERRIDE}}
  
  Shows the entire Fl\_Browser widget – opposite of hide().
• void \texttt{show (int line)}
  
  Makes line visible, and available for selection by user.
• int \texttt{size () const}
  
  Returns how many lines are in the browser.
• void \texttt{size (int W, int H)}
  
  Swaps two browser lines a and b.
• const char \* \texttt{text (int line) const}
  
  Returns the label text for the specified line.
• void \texttt{text (int line, const char \*newtext)}
  
  Sets the text for the specified line to newtext.
• Fl\_Fontsize \texttt{textsize () const}
  
  Gets the default text size (in pixels) for the lines in the browser.
• void \texttt{textsize (Fl\_Fontsize newSize)}
  
  Sets the default text size (in pixels) for the lines in the browser to newSize.
• int \texttt{topline () const}
  
  Returns the line that is currently visible at the top of the browser.
• void \texttt{topline (int line)}
  
  Scrolls the browser so the top item in the browser is showing the specified line.
• int \texttt{value () const}
  
  Returns the line number of the currently selected line, or 0 if none selected.
• void \texttt{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 FL_OVERRIDE

    The height of the entire list of all visible() items in pixels.

• int incr_height () const FL_OVERRIDE

    The default ‘average’ item height (including inter-item spacing) in pixels.

• void insert (int line, FL_BLINE *item)

    Inserts specified item above line.

• void * item_at (int line) const FL_OVERRIDE

    Returns the item at specified line.

• void draw (void *item, int X, int Y, int W, int H) const FL_OVERRIDE

    Draws item at the position specified by XYWH.

• void * item_first () const FL_OVERRIDE

    Returns the very first item in the list.

• int item_height (void *item) const FL_OVERRIDE

    Returns height of item in pixels.

• void * item_last () const FL_OVERRIDE

    Returns the very last item in the list.

• void * item_next (void *item) const FL_OVERRIDE

    Returns the next item after item.

• void * item_prev (void *item) const FL_OVERRIDE

    Returns the previous item before item.

• void select (void *item, int val) FL_OVERRIDE

    Change the selection state of item to the value val.

• int item_selected (void *item) const FL_OVERRIDE

    See if item is selected.

• void swap (void *a, void *b) FL_OVERRIDE

    Swap the items a and b.

• const char * text (void *item) const FL_OVERRIDE

    Returns the label text for item.

• int width (void *item) const FL_OVERRIDE

    Returns width of item in pixels.

• int lineno (void *item) const

    Returns line number corresponding to item, or zero if not found.

• void swap (FL_BLINE *a, FL_BLINE *b)

    Swap the two items a and b.
Additional Inherited Members

33.9.1 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 33.2 Fl_Hold_Browser](image)

![Figure 33.3 Fl_Multi_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 \( \leq \text{size}() \).

Each line contains a null-terminated string of text and a void * data pointer. The text string is displayed, the void * pointer can be used by the callbacks to reference the object the text describes.

The base class does nothing when the user clicks on it. The subclasses Fl_Select_Browser, Fl_Hold_Browser, and Fl_Multi_Browser react to user clicks to select lines in the browser and do callbacks.

The base class Fl_Browser provides the scrolling and selection mechanisms of this and all the subclasses, but the dimensions and appearance of each item are determined by the subclass. You can use Fl_Browser to display information other than text, or text that is dynamically produced from your own data structures. If you find that loading the browser is a lot of work or is inefficient, you may want to make a subclass of Fl_Browser.

Some common coding patterns used for working with Fl_Browser:

```c
// How to loop through all the items in the browser
for ( int t=1; t<browser->size(); t++ ) { // index 1 based.!
    printf("item \#\d, label='\s\n", t, browser->text(t));
}
```

Note: If you are subclassing Fl_Browser, it’s more efficient to use the protected methods item_first() and item_next(), since Fl_Browser internally uses linked lists to manage the browser’s items. For more info, see find_item(int).

33.9.2 Constructor & Destructor Documentation

33.9.2.1 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>
33.9.3 Member Function Documentation

33.9.3.1 _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

| in  | line | The line number to be removed. (1 based) Must be in range! |

Returns

Pointer to browser item that was removed (and is no longer valid).

See also

add(), insert(), remove(), swap(int,int), clear()

33.9.3.2 add()

void Fl_Browser::add (const char * newtext, void * d = 0)
Adds a new line to the end of the browser.
The text string newtext may contain format characters; see format_char() for details. newtext is copied using
the strdup() function, and can be NULL to make a blank line.
The optional void* argument d will be the data() for the new item.

Parameters

| in | newtext | The label text used for the added item |
| in | d       | Optional user data() for the item (0 if unspecified) |

See also

add(), insert(), remove(), swap(int,int), clear()

33.9.3.3 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. |
33.9 Fl_Browser Class Reference

See also
topline(), middleline(), bottomline(), displayed(), lineposition()

33.9.3.4 clear()

void Fl_Browser::clear ()
Removes all the lines in the browser.
See also
add(), insert(), remove(), swap(int,int), clear()

33.9.3.5 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()

33.9.3.6 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()

33.9.3.7 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 it’s own column. After the last column any remaining text is for-
matted into the space between the last column and the right edge of the browser, even if the text contains instances
of column_char() . The default value is a one-element array of just a zero, which means there are no columns.
Example:
Fl_Browser *b = new Fl_Browser(.,.);
static int widths[] = { 50, 50, 15, 70, 70, 40, 40, 70, 70, 50, 0 }; // widths for each column
b->column_widths(widths); // assign array to widget
b->column_char(‘\t’); // use tab as the column character
b->add("USER\tPID\tCPU\tMEM\tVSZ\tRSS\tTTY\tSTAT\tSTART\tTIME\tCOMMAND");
b->add("root\t2888\t0.0\t0.0\t1352\t0\ttty3\tSW\tAug15\t0:00\@b@f/sbin/mingetty tty3");
b->add("root\t13115\t0.0\t0.0\t1352\t0\ttty2\tSW\tAug30\t0:00\@b@f/sbin/mingetty tty2");
[..]
See also
column_char(), column_widths()
33.9.3.8 column_widths() [2/2]

```c
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()`

33.9.3.9 data() [1/2]

```c
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)

33.9.3.10 data() [2/2]

```c
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) |

33.9.3.11 display()

```c
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()`
33.9.3.12  displayed()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line to be checked</th>
</tr>
</thead>
</table>

Returns

1 if visible, 0 if not visible.

**See also**

topline(), middleline(), bottomline(), displayed(), lineposition()

33.9.3.13  find_line()

```c
FL_BLINE * Fl_Browser::find_line (    
    int line ) const [protected]
```

Returns the item for specified `line`.

Note: This call is slow. It's fine for e.g. responding to user clicks, but slow if called often, such as in a tight sorting
loop. Finding an item 'by line' involves a linear lookup on the internal linked list. The performance hit can be
significant if the browser's contents is large, and the method is called often (e.g. during a sort). If you're writing a
subclass, use the protected methods item_first(), item_next(), etc. to access the internal linked list more efficiently.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>line</th>
<th>The line number of the item to return. (1 based)</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>item</th>
<th>that was found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if line is out of range.</td>
</tr>
</tbody>
</table>

**See also**

item_at(), find_line(), lineno()

33.9.3.14  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.

33.9.3.15 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

33.9.3.16 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_.

Generated by Doxygen
### 33.9.3.17 hide() [1/2]
void Fl_Browser::hide () [inline], [virtual]
Hides the entire Fl_Browser widget – opposite of show().
Reimplemented from Fl_Widget.

### 33.9.3.18 hide() [2/2]
void Fl_Browser::hide ( int line )
Makes line invisible, preventing selection by the user.
The line can still be selected under program control. This changes the full_height() if the state was changed. When a line is made invisible, lines below it are moved up in the display. redraw() is called automatically if a change occurred.

**Parameters**

| in  | line | The line to be hidden. (1 based) |

See also

- show(int), hide(int), display(), visible(), make_visible()

### 33.9.3.19 icon() [1/2]
Fl_Image * Fl_Browser::icon ( int line ) const
Returns the icon currently defined for line.
If no icon is defined, NULL is returned.

**Parameters**

| in  | line | The line whose icon is returned. |

**Returns**

- The icon defined, or NULL if none.

### 33.9.3.20 icon() [2/2]
void Fl_Browser::icon ( int line,
                        Fl_Image * icon )
Set the image icon for line to the value icon.
Caller is responsible for keeping the icon allocated. The line is automatically redrawn.

**Parameters**

| in  | line | The line to be modified. If out of range, nothing is done. |
| in  | icon | The image icon to be assigned to the line. If NULL, any previous icon is removed. |
33.9.3.21 incr_height()

```cpp
template 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_.

33.9.3.22 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>Line position for insert. (1 based) If <code>line &gt; size()</code>, the entry will be added at the end.</td>
</tr>
<tr>
<td>in</td>
<td>newtext</td>
<td>The label text for the new line.</td>
</tr>
<tr>
<td>in</td>
<td>d</td>
<td>Optional pointer to user data to be associated with the new line.</td>
</tr>
</tbody>
</table>

33.9.3.23 insert() [2/2]

```cpp
void Fl_Browser::insert(
    int line,
    FL_BLINE * item) [protected]
```

Insert specified `item` above `line`.
If `line > size()` then the line is added to the end.
Caveat: See efficiency note in `find_line()`.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>line</td>
<td>The new line will be inserted above this line (1 based).</td>
</tr>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be added.</td>
</tr>
</tbody>
</table>

33.9.3.24 item_at()

```cpp
void * Fl_Browser::item_at(
    int line) const [inline], [protected], [virtual]
```

Return the item at specified `line`.
Parameters

| in  | line | The line of the item to return. (1 based) |

Returns

The item, or NULL if line out of range.

See also

- `item_at()`, `find_line()`, `lineno()`

Reimplemented from `Fl_Browser_`.

### 33.9.3.25 item_draw()

```c
void Fl_Browser::item_draw ( 
    void * item, 
    int X, 
    int Y, 
    int W, 
    int H ) const [protected], [virtual]
```

Draws `item` at the position specified by `X Y W H`. The `W` and `H` values are used for clipping. Should only be called within the context of an FLTK `draw()`.

Parameters

| in  | item | The item to be drawn |
| in  | X,Y,W,H | position and size. |

Implements `Fl_Browser_`.

### 33.9.3.26 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’\n”, item_text(i));  
}
```

Returns

The first item, or NULL if list is empty.

See also

- `item_first()`, `item_last()`, `item_next()`, `item_prev()`

Implements `Fl_Browser_`.

### 33.9.3.27 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 height of the item in pixels.

See also

item_height(), item_width(), incr_height(), full_height()

Implements Fl_Browser_.

33.9.3.28 item_last()

```cpp
void * Fl_Browser::item_last () const [protected], [virtual]
```

Returns the very last item in the list.

Example of use:

```cpp
// Walk the browser in reverse, from end to start
for ( void *i=item_last(); i; i=item_prev(i) ) {
   printf("item label='s'\n", item_text(i));
}
```

Returns

The last item, or NULL if list is empty.

See also

item_first(), item_last(), item_next(), item_prev()

Reimplemented from Fl_Browser_.

33.9.3.29 item_next()

```cpp
void * Fl_Browser::item_next ( void *item ) const [protected], [virtual]
```

Returns the next item after item.

Parameters

| in | item | The 'current' item |

Returns

The next item after item, or NULL if there are none after this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

33.9.3.30 item_prev()

```cpp
void * Fl_Browser::item_prev ( void *item ) const [protected], [virtual]
```
Returns the previous item before item.
**Parameters**

- **in item** The ‘current’ item

**Returns**

The previous item before `item`, or NULL if there are none before this one.

**See also**

- `item_first()`, `item_last()`, `item_next()`, `item_prev()`

**Implements Fl_Browser_._**

### 33.9.3.31 item_select()

```cpp
void Fl_Browser::item_select (  
    void * item,  
    int val ) [protected], [virtual]
```

Change the selection state of `item` to the value `val`.

**Parameters**

- **in item** The item to be changed.
- **in val** The new selection state: 1 selects, 0 de-selects.

**See also**

- `select()`, `selected()`, `value()`, `item_select()`, `item_selected()`

Reimplemented from Fl_Browser_._

### 33.9.3.32 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_._

### 33.9.3.33 item_swap()

```cpp
void Fl_Browser::item_swap (  
    void * a,  
    void * b ) [inline], [protected], [virtual]
```

Generated by Doxygen
Swap the items \texttt{a} and \texttt{b}.
You must call \texttt{redraw()} to make any changes visible.

Parameters

\begin{itemize}
\item \texttt{in \texttt{a,b}} \hspace{1cm} \text{the items to be swapped.}
\end{itemize}

See also

\texttt{swap(int,int), item_swap()}

Reimplemented from \texttt{Fl_Browser}.  

33.9.3.34 \texttt{item_text()}

\begin{verbatim}
const char * Fl_Browser::item_text ( void * item ) const [protected], [virtual]
\end{verbatim}

Returns the label text for \texttt{item}.

Parameters

\begin{itemize}
\item \texttt{in \texttt{item}} \hspace{1cm} \text{The item whose label text is returned.}
\end{itemize}

Returns

The item's text string. (Can be NULL)

Reimplemented from \texttt{Fl_Browser}.  

33.9.3.35 \texttt{item_width()}

\begin{verbatim}
int Fl_Browser::item_width ( void * item ) const [protected], [virtual]
\end{verbatim}

Returns width of \texttt{item} in pixels.
This takes into account embedded @ codes within the \texttt{text()} label.

Parameters

\begin{itemize}
\item \texttt{in \texttt{item}} \hspace{1cm} \text{The item whose width is returned.}
\end{itemize}

Returns

The width of the item in pixels.

See also

\texttt{item_height(), item_width(), incr_height(), full_height()}

Implements \texttt{Fl_Browser}.  

33.9.3.36 \texttt{lineno()}

\begin{verbatim}
int Fl_Browser::lineno ( void * item ) const [protected]
\end{verbatim}

Returns line number corresponding to \texttt{item}, or zero if not found.
Caveat: See efficiency note in \texttt{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()

33.9.3.37 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) |
|     | pos   | position.              |

See also

topline(), middleline(), bottomline()

33.9.3.38 load()

int Fl_Browser::load (  
  const char * filename )

Clears the browser and reads the file, adding each line from the file to the browser. If the filename is NULL or a zero-length string then this just clears the browser. This returns zero if there was any error in opening or reading the file, in which case errno is set to the system error. The data() of each line is set to NULL.

Parameters

| in  | filename | The filename to load |

Returns

1 if OK, 0 on error (errno has reason)

See also

add()

33.9.3.39 make_visible()

void Fl_Browser::make_visible (  
  int line ) [inline]
Make the item at the specified line visible(). Functionally similar to show(int line). If line is out of range, redisplay top or bottom of list as appropriate.

Parameters

| in | The line to be made visible. |

See also

- show(int), hide(int), display(), visible(), make_visible()

### 33.9.3.40 middleline()

```cpp
void Fl_Browser::middleline (int line) [inline]
```

Scrolls the browser so the middle item in the browser is showing the specified line.

Parameters

| in | The line to be displayed in the middle. |

See also

- topline(), middleline(), bottomline(), displayed(), lineposition()

### 33.9.3.41 move()

```cpp
void Fl_Browser::move (int to, int from)
```

Line from is removed and reinserted at to. Note: to is calculated after line from gets removed.

Parameters

| in | Destination line number (calculated after line from is removed) |
| in | Line number of item to be moved |

### 33.9.3.42 remove()

```cpp
void Fl_Browser::remove (int line)
```

Remove entry for given line number, making the browser one line shorter. You must call redraw() to make any changes visible.

Parameters

| in | Line to be removed. (1 based) |
| in | If line is out of range, no action is taken. |
See also

```
add(), insert(), remove(), swap(int,int), clear()
```

33.9.3.43  remove_icon()

```cpp
void Fl_Browser::remove_icon {
    int line
}
```

Removes the icon for line.
It's ok to remove an icon if none has been defined.

Parameters

| in | line | The line whose icon is to be removed. |

33.9.3.44  select()

```cpp
int Fl_Browser::select {
    int line,
    int val = 1
}
```

Sets the selection state of the item at line to the value val.
If val is not specified, the default is 1 (selects the item).

Parameters

| 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()
```

33.9.3.45  selected()

```cpp
int Fl_Browser::selected {
    int line
}
```

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()
```
### 33.9.3.46 show() [1/2]

```cpp
void Fl_Browser::show () [inline], [virtual]
```

Shows the entire Fl_Browser widget – opposite of hide().
Reimplemented from Fl_Widget.

### 33.9.3.47 show() [2/2]

```cpp
void Fl_Browser::show ( int line )
```

Makes `line` visible, and available for selection by user.
Opposite of `hide(int)`. This changes the `full_height()` if the state was changed. `redraw()` is called automatically if a change occurred.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>line</code></td>
<td>The line to be shown. (1 based)</td>
</tr>
</tbody>
</table>

**See also**

`show(int), hide(int), display(), visible(), make_visible()`

### 33.9.3.48 size()

```cpp
int Fl_Browser::size () const [inline]
```

Returns how many lines are in the browser.
The last line number is equal to this. Returns 0 if browser is empty.

### 33.9.3.49 swap() [1/2]

```cpp
void Fl_Browser::swap ( FL_BLINE * a, FL_BLINE * b ) [protected]
```

Swap the two items `a` and `b`.
Uses `swapping()` to ensure list updates correctly.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>The two items to be swapped.</td>
</tr>
</tbody>
</table>

**See also**

`swap(int,int), item_swap()`

### 33.9.3.50 swap() [2/2]

```cpp
void Fl_Browser::swap ( int a, int b )
```

Swaps two browser lines `a` and `b`.
You must call `redraw()` to make any changes visible.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a,b</code></td>
<td>The two lines to be swapped. (both 1 based)</td>
</tr>
</tbody>
</table>
See also

\texttt{swap(int,int), item\_swap()}

### 33.9.3.51 \texttt{text()} [1/2]

\begin{verbatim}
const char * Fl_Browser::text ( int line ) const
\end{verbatim}

Returns the label text for the specified \texttt{line}.
Return value can be NULL if \texttt{line} is out of range or unset. The parameter \texttt{line} is 1 based.

**Parameters**

\begin{itemize}
  \item \texttt{in line} The line number of the item whose text is returned. (1 based)
\end{itemize}

**Returns**

The text string (can be NULL)

### 33.9.3.52 \texttt{text()} [2/2]

\begin{verbatim}
void Fl_Browser::text ( int line, const char * newtext )
\end{verbatim}

Sets the text for the specified \texttt{line} to \texttt{newtext}.
Text may contain format characters; see \texttt{format\_char()} for details. \texttt{newtext} is copied using the \texttt{strdup()} function, and can be NULL to make a blank line.
Does nothing if \texttt{line} is out of range.

**Parameters**

\begin{itemize}
  \item \texttt{in line} The line of the item whose text will be changed. (1 based)
  \item \texttt{in newtext} The new string to be assigned to the item.
\end{itemize}

### 33.9.3.53 \texttt{textsize()}

\begin{verbatim}
void Fl_Browser::textsize ( Fl_Fontsize newSize )
\end{verbatim}

Sets the default text size (in pixels) for the lines in the browser to \texttt{newSize}.
This method recalculates all item heights and caches the total height internally for optimization of later item changes.
This can be slow if there are many items in the browser.
It returns immediately (w/o recalculation) if \texttt{newSize} equals the current \texttt{textsize()}.
You \textit{may} need to call \texttt{redraw()} to see the effect and to have the scrollbar positions recalculated.
You should set the text size \textit{before} populating the browser with items unless you really need to change the size later.

### 33.9.3.54 \texttt{topline()} [1/2]

\begin{verbatim}
int Fl_Browser::topline ( ) const
\end{verbatim}

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.

### 33.9.3.55 topline [2/2]

```cpp
void Fl_Browser::topline ( int line ) [inline]
```

Scrolls the browser so the top item in the browser is showing the specified line.

**Parameters**

| in  | line | The line to be displayed at the top. |

**See also**

topline(), middleline(), bottomline(), displayed(), lineposition()

### 33.9.3.56 value [1/2]

```cpp
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()

### 33.9.3.57 value [2/2]

```cpp
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()

### 33.9.3.58 visible()

```cpp
int Fl_Browser::visible ( int line ) const
```

Returns non-zero if the specified line is visible, 0 if hidden.

Use show(int), hide(int), or make_visible(int) to change an item’s visible state.

**Parameters**

| in  | line | The line in the browser to be tested. (1 based) |
See also

    show(int), hide(int), display(), visible(), make_visible()

The documentation for this class was generated from the following files:

- Fl_Browser.H
- Fl_Browser.cxx
- Fl_Browser_load.cxx

### 33.10 Fl_Browser_ Class Reference

This is the base class for browsers.

```cpp
#include <Fl_Browser_.H>
```

Inheritance diagram for Fl_Browser_:

![Inheritance diagram](image)

#### Public Types

- enum {
  `HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4, HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH ALWAYS = 7`
}

Values for `has_scrollbar()`.

#### Public Member Functions

- int **deselect** (int docallbacks=0)
  
  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

- void **display** (void *item)
  
  Displays the item, scrolling the list as necessary.

- int **handle** (int event) FL_OVERRIDE
  
  Handles the event within the normal widget bounding box.

- uchar **has_scrollbar** () const
  
  Returns the current scrollbar mode, see `Fl_Browser_::has_scrollbar(uchar)`

- void **has_scrollbar** (uchar mode)
  
  Sets whether the widget should have scrollbars or not (default `Fl_Browser_::BOTH`).

- int **hposition** () const
  
  Gets the horizontal scroll position of the list as a pixel position `pos`.

- void **hposition** (int)
  
  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 x, int y)
• **void resize** (int X, int Y, int W, int H) **FL_OVERRIDE**
  Repositions and/or resizes the browser.

• **void scrollbar_left ()**
  Moves the vertical scrollbar to the lefthand side of the list.

• **void scrollbar_right ()**
  Moves the vertical scrollbar to the righthand side of the list.

• **int scrollbar_size () const**
  Gets the current size of the scrollbars’ troughs, in pixels.

• **void scrollbar_size (int newSize)**
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.

• **int scrollbar_width () const**
  Returns the global value Fl::scrollbar_size().

• **void scrollbar_width (int width)**
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• **int select (void ∗item, int val=1, int docallbacks=0)**
  Sets the selection state of item to val, and returns 1 if the state changed or 0 if it did not.

• **int select_only (void ∗item, int docallbacks=0)**
  Selects item and returns 1 if the state changed or 0 if it did not.

• **void sort (int flags=0)**
  Sorts the items in the browser based on flags.

• **Fl_Color textcolor () const**
  Gets the default text color for the lines in the browser.

• **void textcolor (Fl_Color col)**
  Sets the default text color for the lines in the browser to color col.

• **Fl_Font textfont () const**
  Gets the default text font for the lines in the browser.

• **void textfont (Fl_Font font)**
  Sets the default text font for the lines in the browser to font.

• **Fl_Fontsize textsize () const**
  Gets the default text size (in pixels) for the lines in the browser.

• **void textsize (Fl_Fontsize newSize)**
  Sets the default text size (in pixels) for the lines in the browser to size.

• **int vposition () const**
  Gets the vertical scroll position of the list as a pixel position pos.

• **void vposition (int pos)**
  Sets the vertical scroll position of the list to pixel position pos.

### Public Attributes

- **Fl_Scrollbar hscrollbar**
  Horizontal scrollbar.
- **Fl_Scrollbar scrollbar**
  Vertical scrollbar.
Protected Member Functions

- void **bbox** (int &X, int &Y, int &W, int &H) const
  
  *Returns the bounding box for the interior of the list's display window, inside the scrollbars.*

- void **deleting** (void **item**)
  
  *This method should be used when **item** is being deleted from the list.*

- int **displayed** (void **item**) const
  
  *Returns non-zero if **item** has been scrolled to a position where it is being displayed.*

- void **draw** () **FL_OVERRIDE**
  
  *Draws the list within the normal widget bounding box.*

- void **find_item** (int ypos)
  
  *This method returns the item under mouse y position ypos.*

- **Fl_Browser** (int X, int Y, int W, int H, const char **L=0**)
  
  *The constructor makes an empty browser.*

- virtual int **full_height** () const
  
  *This method may be provided by the subclass to indicate the full height of the item list, in pixels.*

- virtual int **full_width** () const
  
  *This method may be provided by the subclass to indicate the full width of the item list, in pixels.*

- virtual int **incr_height** () const
  
  *This method may be provided to return the average height of all items to be used for scrolling.*

- void **inserting** (void **a**, void **b**)
  
  *This method should be used when an item is in the process of being inserted into the list.*

- virtual void *item_at** (int index) const
  
  *This method must be provided by the subclass to return the item for the specified index.*

- virtual void **item_draw** (void **item**, int X, int Y, int W, int H) const =0
  
  *This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.*

- virtual void **item_first** () const =0
  
  *This method must be provided by the subclass to return the first item in the list.*

- virtual int **item_height** (void **item**) const =0
  
  *This method must be provided by the subclass to return the height of **item** in pixels.*

- virtual void **item_last** () const
  
  *This method must be provided by the subclass to return the last item in the list.*

- virtual void *item_next** (void **item**) const =0
  
  *This method must be provided by the subclass to return the item in the list after **item**.*

- virtual void **item_prev** (void **item**) const =0
  
  *This method must be provided by the subclass to return the item in the list before **item**.*

- virtual int **item_quick_height** (void **item**) const
  
  *This method may be provided by the subclass to return the height of the item, in pixels.*

- virtual void **item_select** (void **item**, int val=1)
  
  *This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.*

- virtual int **item_selected** (void **item**) const
  
  *This method must be implemented by the subclass if it supports multiple selections; returns the selection state for **item**.*

- virtual void **item_swap** (void **a**, void **b**)
  
  *This optional method should be provided by the subclass to efficiently swap browser items **a** and **b**, such as for sorting.*

- virtual const char **item_text** (void **item**) const
  
  *This optional method returns a string (label) that may be used for sorting.*

- virtual int **item_width** (void **item**) const =0
  
  *This method must be provided by the subclass to return the width of the **item** in pixels.*

- 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

#### 33.10.1 Detailed Description

This is the base class for browsers.

To be useful it must be subclassed and several virtual functions defined. The Forms-compatible browser and the file chooser’s browser are subclassed off of this.

This has been designed so that the subclass has complete control over the storage of the data, although because next() and prev() functions are used to index, it works best as a linked list or as a large block of characters in which the line breaks must be searched for.

A great deal of work has been done so that the “height” of a data object does not need to be determined until it is drawn. This is useful if actually figuring out the size of an object requires accessing image data or doing stat() on a file or doing some other slow operation.

Callbacks are called when the value changes with FL_REASON_CHANGED. If FL_WHEN_RELEASE is set, callbacks are called when the mouse button is released with FL_REASON_CHANGED or FL_REASON←RESELECTED if the selection did not change. If FL_WHEN_ENTER_KEY is set, callbacks are also called when key presses or double clicks change the selection.

#### 33.10.1.1 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.

#### 33.10.2 Member Enumeration Documentation

##### 33.10.2.1 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

Enumerator

<table>
<thead>
<tr>
<th>Enumerators</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>

33.10.3 Constructor & Destructor Documentation

33.10.3.1 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>

33.10.4 Member Function Documentation

33.10.4.1 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

| out  | X,Y,W,H | The returned bounding box. |
|------|---------| (The original contents of these parameters are overwritten) |
33.10.4.2 deleting()

```cpp
void Fl_Browser_::deleting ( 
    void * item ) [protected]
```

This method should be used when `item` is being deleted from the list. It allows the `Fl_Browser_` to discard any cached data it has on the item. This method does not actually delete the item, but handles the follow up bookkeeping after the item has just been deleted.

Parameters

| in | item | The item being deleted. |

33.10.4.3 deselect()

```cpp
int Fl_Browser_::deselect ( 
    int docallbacks = 0 
)
```

Deselects all items in the list and returns 1 if the state changed or 0 if it did not. If the optional `docallbacks` parameter is non-zero, deselect tries to call the callback function for the widget.

Parameters

| in | docallbacks | If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default). |

33.10.4.4 display()

```cpp
void Fl_Browser_::display ( 
    void * item )
```

Displays the `item`, scrolling the list as necessary.

Parameters

| in | item | The item to be displayed. |

See also

`display()`, `displayed()`

33.10.4.5 displayed()

```cpp
int Fl_Browser_::displayed ( 
    void * item ) const [protected]
```

Returns non-zero if `item` has been scrolled to a position where it is being displayed. Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the `hide()`/`show()` status of the widget or item.

Parameters

| in | item | The item to check |
Returns

1 if visible, 0 if not visible.

See also

display(), displayed()

### 33.10.4.6 draw()

```cpp
void Fl_Browser_::draw () [protected], [virtual]
```

Draws the list within the normal widget bounding box.
Implements Fl_Widget.

### 33.10.4.7 find_item()

```cpp
void * Fl_Browser_::find_item ( int ypos ) [protected]
```

This method returns the item under mouse y position ypos. NULL is returned if no item is displayed at that position.

Parameters

| in  | ypos | The y position (eg. Fl::event_y()) to find an item under. |

Returns

The item, or NULL if not found

### 33.10.4.8 full_height()

```cpp
int Fl_Browser_::full_height () const [protected], [virtual]
```

This method may be provided by the subclass to indicate the full height of the item list, in pixels.
The default implementation computes the full height from the item heights. Includes the items that are scrolled off screen.

Returns

The height of the entire list, in pixels.

Reimplemented in Fl_Browser.

### 33.10.4.9 full_width()

```cpp
int Fl_Browser_::full_width () const [protected], [virtual]
```

This method may be provided by the subclass to indicate the full width of the item list, in pixels.
The default implementation computes the full width from the item widths.

Returns

The maximum width of all the items, in pixels.
33.10.4.10 handle()

```cpp
int Fl_Browser_::handle (int event) [virtual]
```

Handles the `event` within the normal widget bounding box.

**Parameters**

- **event** The event to process.

**Returns**

- 1 if event was processed, 0 if not.

Reimplemented from `Fl_Widget`.
Reimplemented in `Fl_Check_Browser`.

33.10.4.11 has_scrollbar()

```cpp
void Fl_Browser_::has_scrollbar (uchar mode) [inline]
```

Sets whether the widget should have scrollbars or not (default `Fl_Browser_::BOTH`). By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the widget. `has_scrollbar()` changes this based on the value of `mode`:

- 0 - No scrollbars.
- `Fl_Browser_::HORIZONTAL` - Only a horizontal scrollbar.
- `Fl_Browser_::VERTICAL` - Only a vertical scrollbar.
- `Fl_Browser_::BOTH` - The default is both scrollbars.
- `Fl_Browser_::HORIZONTAL_ALWAYS` - Horizontal scrollbar always on, vertical always off.
- `Fl_Browser_::VERTICAL_ALWAYS` - Vertical scrollbar always on, horizontal always off.
- `Fl_Browser_::BOTH_ALWAYS` - Both always on.

33.10.4.12 hposition [1/2]

```cpp
int Fl_Browser_::hposition () const [inline]
```

Gets the horizontal scroll position of the list as a pixel position `pos`.
The position returned is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' indicates the left 18 pixels of the list are scrolled off the left edge of the screen.

See also

- `position()`, `hposition()`

33.10.4.13 hposition [2/2]

```cpp
void Fl_Browser_::hposition (int pos)
```

Sets the horizontal scroll position of the list to pixel position `pos`.
The position is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' scrolls the left 18 pixels of the list off the left edge of the screen.
Parameters

| in | pos | The horizontal position (in pixels) to scroll the browser to. |

See also

vposition(), hposition()

33.10.4.14 incr_height()

```cpp
int Fl_Browser_::incr_height ( ) const [protected], [virtual]
```

This method may be provided to return the average height of all items to be used for scrolling.
The default implementation uses the height of the first item.

Returns

The average height of items, in pixels.

Reimplemented in Fl_Browser.

33.10.4.15 inserting()

```cpp
void Fl_Browser_::inserting ( 
  void * a,
  void * b ) [protected]
```

This method should be used when an item is in the process of being inserted into the list.
It allows the Fl_Browser_ to update its cache data as needed, scheduling a redraw for the affected lines. This
method does not actually insert items, but handles the follow up bookkeeping after items have been inserted.

Parameters

| in | a | The starting item position |
| in | b | The new item being inserted |

33.10.4.16 item_at()

```cpp
virtual void * Fl_Browser_::item_at ( 
  int index ) const [inline], [protected], [virtual]
```

This method must be provided by the subclass to return the item for the specified index.

Parameters

| in | index | The index of the item to be returned |

Returns

The item at the specified index.

Reimplemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.17 item_draw()

```cpp
virtual void Fl_Browser_::item_draw ( 
  void * item,
```
int X,
int Y,
int W,
int H) const [protected], [pure virtual]
This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.
Implemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.18 item_first()
virtual void * Fl_Browser_::item_first ( ) const [protected], [pure virtual]
This method must be provided by the subclass to return the first item in the list.
See also
   item_first(), item_next(), item_last(), item_prev()
Implemented in Fl_Browser, and Fl_Check_Browser.

33.10.4.19 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_Check_Browser, and Fl_Browser.

33.10.4.20 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.

33.10.4.21 item_next()
virtual void * Fl_Browser_::item_next ( void * item ) const [protected], [pure virtual]
This method must be provided by the subclass to return the item in the list after item.
See also
   item_first(), item_next(), item_last(), item_prev()
Implemented in Fl_Check_Browser, and Fl_Browser.
33.10.4.22 item_prev()

virtual void * Fl_Browser_::item_prev (  
    void * item ) const [protected], [pure virtual]
This method must be provided by the subclass to return the item in the list before item.

See also
    item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.23 item_quick_height()

int Fl_Browser_::item_quick_height (  
    void * item ) const [protected], [virtual]
This method may be provided by the subclass to return the height of the item, in pixels.
Allow for two additional pixels for the list selection box. This method differs from item_height in that it is only called
for selection and scrolling operations. The default implementation calls item_height.

Parameters

| in  | item          | The item whose height to return. |

Returns

The height, in pixels.

33.10.4.24 item_select()

void Fl_Browser_::item_select (  
    void * item,  
    int val = 1 ) [protected], [virtual]
This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val
for the item.
Sets the selection state for item, where optional val is 1 (select, the default) or 0 (de-select).

Parameters

| in  | item          | The item to be selected |
| in  | val           | The optional selection state: 1=select, 0=de-select. The default is to select the item (1). |

Reimplemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.25 item_selected()

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. |
33.10 Fl_Browser_ Class Reference

Reimplemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.26 item_swap()

virtual void Fl_Browser_::item_swap (  
    void * a,  
    void * b ) [inline], [protected], [virtual]

This optional method should be provided by the subclass to efficiently swap browser items a and b, such as for sorting.

Parameters

- **in a,b** The two items to be swapped.

Reimplemented in Fl_Browser, and Fl_Check_Browser.

33.10.4.27 item_text()

virtual const char * Fl_Browser_::item_text (  
    void * item ) const [inline], [protected], [virtual]

This optional method returns a string (label) that may be used for sorting.

Parameters

- **in item** The item whose label text is returned.

Returns

- The item's text label. (Can be NULL if blank)

Reimplemented in Fl_Browser, and Fl_Check_Browser.

33.10.4.28 item_width()

virtual int Fl_Browser_::item_width (  
    void * item ) const [protected], [pure virtual]

This method must be provided by the subclass to return the width of the item in pixels. Allow for two additional pixels for the list selection box.

Parameters

- **in item** The item whose width is returned.

Returns

- The width of the item in pixels.

Implemented in Fl_Check_Browser, and Fl_Browser.

33.10.4.29 leftedge()

int Fl_Browser_::leftedge ( ) const [protected]

This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
Returns
The X position of the left edge of the list, in pixels.

See also
Fl_Browser_::bbox()

33.10.4.30  new_list()

void Fl_Browser_::new_list ( ) [protected]
This method should be called when the list data is completely replaced or cleared.
It informs the Fl_Browser_ widget that any cached information it has concerning the items is invalid. This method
does not clear the list, it just handles the follow up bookkeeping after the list has been cleared.

33.10.4.31  position() [1/2]

int Fl_Browser_::position ( ) const [inline]

Deprecated  "in 1.4.0 - use vposition() instead"

33.10.4.32  position() [2/2]

void Fl_Browser_::position ( int pos ) [inline]

Deprecated  "in 1.4.0 - use vposition(pos) instead"

33.10.4.33  redraw_line()

void Fl_Browser_::redraw_line ( void * item ) [protected]
This method should be called when the contents of item has changed, but not its height.

Parameters

| In  | item       | The item that needs to be redrawn. |

See also
redraw_lines(), redraw_line()

33.10.4.34  redraw_lines()

void Fl_Browser_::redraw_lines ( ) [inline], [protected]
This method will cause the entire list to be redrawn.

See also
redraw_lines(), redraw_line()
33.10.4.35 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**

<table>
<thead>
<tr>
<th>in</th>
<th>( a )</th>
<th>Item being replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>( b )</td>
<td>Item to replace 'a'</td>
</tr>
</tbody>
</table>

33.10.4.36 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\_.

33.10.4.37 scrollbar_left()

```cpp
void Fl_Browser_::scrollbar_left ( ) [inline]
```

Moves the vertical scrollbar to the lefthand side of the list. For back compatibility.

33.10.4.38 scrollbar_right()

```cpp
void Fl_Browser_::scrollbar_right ( ) [inline]
```

Moves the vertical scrollbar to the righthand side of the list. For back compatibility.

33.10.4.39 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\(\)
### scrollbar_size() [2/2]

```cpp
class Fl_Browser_ {
    void 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.</td>
</tr>
<tr>
<td></td>
<td>If 0 (default), scrollbar size tracks the global <code>Fl::scrollbar_size()</code></td>
</tr>
</tbody>
</table>

See also

- `Fl::scrollbar_size()`

### scrollbar_width() [1/2]

```cpp
class Fl_Browser_ {
    int scrollbar_width ( ) const [inline]
}
```

Returns the global value `Fl::scrollbar_size()`.

**Deprecated**

Use `scrollbar_size()` instead.

**Todo**

This method should eventually be removed in 1.4+

### scrollbar_width() [2/2]

```cpp
class Fl_Browser_ {
    void scrollbar_width (int width) [inline]
}
```

Sets the global `Fl::scrollbar_size()`, and forces this instance of the widget to use it.

**Deprecated**

Use `scrollbar_size()` instead.

**Todo**

This method should eventually be removed in 1.4+

### select()

```cpp
class Fl_Browser_ {
    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.

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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The item whose selection state is to be changed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The new selection state (1=select, 0=de-select)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>docallbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).</td>
</tr>
</tbody>
</table>

**Generated by Doxygen**
Returns

1 if state was changed, 0 if not.

### 33.10.4.44 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>

### 33.10.4.45 selection()

```cpp
void * Fl_Browser_::selection ( ) const [inline], [protected]
```

Returns the item currently selected, or NULL if there is no selection.

For multiple selection browsers this call returns the currently focused item, even if it is not selected. To find all selected items, call `Fl_Multi_Browser::selected()` for every item in question.

### 33.10.4.46 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>FL_SORT_ASCENDING – sort in ascending order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FL_SORT_DESCENDING – sort in descending order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL_SORT_CASEINSENSITIVE – add this to sort case-insensitively</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values other than the above will cause undefined behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other flags may appear in the future.</td>
</tr>
</tbody>
</table>

### 33.10.4.47 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>a,b</th>
<th>Items being swapped.</th>
</tr>
</thead>
</table>
33.10.48  textfont()

\texttt{Fl\_Font Fl\_Browser\_::textfont() const [inline]}

Gets the default text font for the lines in the browser.

See also

\texttt{textfont(), textsize(), textcolor()}

33.10.49  vposition() [1/2]

\texttt{int Fl\_Browser\_::vposition() const [inline]}

Gets the vertical scroll position of the list as a pixel position \texttt{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

\texttt{position(), hposition()}

33.10.50  vposition() [2/2]

\texttt{void Fl\_Browser\_::vposition(int \texttt{pos})}

Sets the vertical scroll position of the list to pixel position \texttt{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

\begin{center}
\begin{tabular}{|l|l|}
\hline
\texttt{pos} & The vertical position (in pixels) to scroll the browser to. \\
\hline
\end{tabular}
\end{center}

See also

\texttt{vposition(), hposition()}

33.10.5  Member Data Documentation

33.10.5.1  hscrollbar

\texttt{Fl\_Scrollbar Fl\_Browser\_::hscrollbar}

Horizontal scrollbar.

Public, so that it can be accessed directly.

33.10.5.2  scrollbar

\texttt{Fl\_Scrollbar Fl\_Browser\_::scrollbar}

Vertical scrollbar.

Public, so that it can be accessed directly.

Use \texttt{scrollbar\_left()} or \texttt{scrollbar\_right()} to change what side the vertical scrollbar is drawn on.

Use \texttt{scrollbar\_align(int)} (see \texttt{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

### 33.11 Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

```c
#include <Fl_Button.H>
```

Inheritance diagram for Fl_Button:

![Inheritance Diagram](image)

#### 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._

- **int handle (int) FL_OVERRIDE**
  
  _Handles the specified event._

- **int set ()**
  
  _Same as value(1)._

- **void setonly ()**
  
  _Turns on this button and turns off all other radio buttons in the group (calling value(1) or set() does not do this)._

- **int shortcut () const**
  
  _Returns the current shortcut key for the button._

- **void shortcut (const char *s)**
  
  _for backwards compatibility_

- **void shortcut (int s)**
  
  _Sets the shortcut key to s._

- **char value () const**
  
  _Returns the current value of the button (0 or 1)._ 

- **int value (int v)**
  
  _Sets the current value of the button._
Protected Member Functions

- void `draw()` FL_OVERRIDE
  Draws the widget.
- void `simulate_key_action()`

Static Protected Member Functions

- static void `key_release_timeout()`

Static Protected Attributes

- static `Fl_Widget_Tracker *key_release_tracker = 0`

Additional Inherited Members

### 33.11.1 Detailed Description

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for `type(uchar)` and `when(uchar)`. Buttons can also generate callbacks in response to `FL_SHORTCUT` events. The button can either have an explicit `shortcut(int s)` value or a letter shortcut can be indicated in the `label()` with an '&' character before it. For the label shortcut it does not matter if `Alt` is held down, but if you have an input field in the same window, the user will have to hold down the `Alt` key so that the input field does not eat the event first as an `FL_KEYBOARD` event.

See also

`Fl_Widget::shortcut_label(int)`

For an `Fl_Button` object, the `type()` call returns one of:

- `FL_NORMAL_BUTTON (0)`: `value()` remains unchanged after button press.
- `FL_TOGGLE_BUTTON`: `value()` is inverted after button press.
- `FL_RADIO_BUTTON`: `value()` is set to 1 after button press, and all other buttons in the current group with `type() == FL_RADIO_BUTTON` are set to zero.

For an `Fl_Button` object, the following `when()` values are useful, the default being `FL_WHEN_RELEASE`:

- 0: The callback is not done, instead `changed()` is turned on.
- `FL_WHEN_RELEASE`: The callback is done after the user successfully clicks the button, or when a shortcut is typed. The reason is `FL_REASON_RELEASED`.
- `FL_WHEN_CHANGED`: The callback is done each time the `value()` changes (when the user pushes and releases the button, and as the mouse is dragged around in and out of the button). The reason is set to `FL_REASON_CHANGED`.
- `FL_WHEN_NOT_CHANGED`: The callback is done when the mouse button is released, but the value did not change. The reason is set to `FL_REASON_SELECTED`.

### 33.11.2 Constructor & Destructor Documentation

#### 33.11.2.1 Fl_Button()

```c
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.
A button may request callbacks with when() FL_WHEN_CHANGED, FL_WHEN_NOT_CHANGED, and FL_WHEN_RELEASE, triggering the callback reasons FL_REASON_CHANGED, FL_REASON_SELECTED, and
FL_REASON_DESELECTED.

Parameters

| in | X,Y,W,H | position and size of the widget |
| in | L       | widget label, default is no label |

### 33.11.3 Member Function Documentation

#### 33.11.3.1 clear()

int Fl_Button::clear () [inline]

Same as value(0).

See also

value(int v)

#### 33.11.3.2 down_box() [1/2]

**Fl_Boxtype** Fl_Button::down_box () const [inline]

Returns the current down box type, which is drawn when value() is non-zero.

Return values

| Fl_Boxtype |

#### 33.11.3.3 down_box() [2/2]

void Fl_Button::down_box ( Fl_Boxtype b ) [inline]

Sets the down box type.
The default value of 0 causes FLTK to figure out the correct matching down version of box().
Some derived classes (e.g. Fl_Round_Button and Fl_Light_Button use down_box() for special purposes. See docs
of these classes.

Parameters

| in | b | down box type |
33.11.3.4 draw()

```cpp
void Fl_Button::draw () [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```
Implements `Fl_Widget`.
Reimplemented in `Fl_Light_Button`, `Fl_Return_Button`, and `Fl_Shortcut_Button`.

33.11.3.5 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>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`.
Reimplemented in `Fl_Light_Button`, `Fl_Repeat_Button`, `Fl_Return_Button`, and `Fl_Shortcut_Button`.

33.11.3.6 set()

```cpp
int Fl_Button::set ( ) [inline]
```

Same as `value(1)`.

See also

- `value(int v)`

33.11.3.7 shortcut()

```cpp
int Fl_Button::shortcut ( ) const [inline]
```

Returns the current shortcut key for the button.

Return values

<table>
<thead>
<tr>
<th>int</th>
<th></th>
</tr>
</thead>
</table>
33.11.3.8 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

| in  | s            | bitwise OR of key and shift flags |

33.11.3.9 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

| in  | v            | button value. |

See also

   set(), clear()

The documentation for this class was generated from the following files:

   • Fl_Button.H
   • Fl_Button.cxx

33.12 Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.
#include <Fl_Cairo.H>

Public Member Functions

   • bool autolink () const
      
      Gets the autolink option. See Fl::cairo_autolink_context(bool)
   • void autolink (bool b)
      
      Sets the autolink option, only available with --enable-cairoext.
   • cairo_t * cc () const
      
      Gets the current cairo context.
   • void cc (cairo_t *c, bool own=true)
      
      Sets the current cairo context.
   • void * gc () const
33.12 Class Documentation

**cc()**

```cpp
void Fl_Cairo_State::cc (cairo_t *c, bool own = true) [inline]
```

Sets the current cairo context.

- **own** == **true** (the default) indicates that the cairo context `c` will be deleted by FLTK internally when another cc is set later.
- **own** == **false** indicates cc deletion is handled externally by the user program.

The documentation for this class was generated from the following files:

- Fl_Cairo.H
- Fl_Cairo.cxx

33.13 Fl_Cairo_Window Class Reference

This defines an FLTK window with Cairo support.

```cpp
#include <Fl_Cairo_Window.H>
```

Inheritance diagram for Fl_Cairo_Window:

```
Fl_Widget

Fl_Group

Fl_Window

Fl_Double_Window

Fl_Cairo_Window
```

Generated by Doxygen
Public Types

- typedef void(*cairo_draw_cb)(Fl_Cairo_Window *self, cairo_t *def)

  The Cairo draw callback prototype you need to implement.

Public Member Functions

- Fl_Cairo_Window(int W, int H, const char *L=0)
- Fl_Cairo_Window(int X, int Y, int W, int H, const char *L=0)
- void set_draw_cb(cairo_draw_cb cb)

  You must provide a draw callback that implements your Cairo rendering.

Protected Member Functions

- void draw() FL_OVERRIDE

  Overloaded to provide Cairo callback support.

Additional Inherited Members

33.13.1 Detailed Description

This defines an FLTK window with Cairo support.

This class overloads the virtual draw() method for you, so that the only thing you have to do is to provide your Cairo code. All Cairo context handling is achieved transparently.

The default coordinate system for Cairo drawing commands within Fl_Cairo_Window is FLTK’s coordinate system, where the \(x, y, w, h\) values are relative to the top/left corner of the Fl_Cairo_Window, as one would expect with regular FLTK drawing commands, e.g.:

\[ (0 \times w-1), (0 \ y \ h-1) \]

**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, 1.0); // 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 \((0.0 \ldots 1.0)\) values for \(x\) and \(y\), e.g.:

\[ (0 \times 1.0), (0 \ y \ 1.0) \]

So beware of this when copy/pasting Cairo example programs that assume normalized values. If need be, you can revert to the Cairo coordinate system by simply calling cairo_scale() with the widget's \(w()\) and \(h()\) values. **Example:**

```c
static void my_cairo_draw_cb(Fl_Cairo_Window *window, cairo_t *cr) {
  cairo_scale(cr, window->w(), window->h()); // use Cairo's default coordinate system
  [...use 0.0 to 1.0 values from here on...]
}
```

See also

- examples/cairo-draw-x.cxx
- test/cairo_test.cxx

Note

Class Fl_Cairo_Window requires the FLTK library to have been built with CMake option OPTION_CAIRO or configure --enable-cairo.

You can alternatively define your custom Cairo FLTK window, and thus at least override the draw() method to provide custom Cairo support. In this case you will probably use Fl::cairo_make_current(Fl_Window*) to attach a context to your window. You should do this only when your window is the current window.

See also

- Fl_Window::current()
33.13.2  Member Function Documentation

33.13.2.1  draw()

void Fl_Cairo_Window::draw ( 
    void ) [inline], [protected], [virtual] 
Overloaded to provide Cairo callback support.
Implements Fl_Widget.

33.13.2.2  set_draw_cb()

void Fl_Cairo_Window::set_draw_cb ( 
    cairo_draw_cb cb ) [inline] 
You must provide a draw callback that implements your Cairo rendering.
This method permits you to set your Cairo callback to cb.
The documentation for this class was generated from the following file:

- Fl_Cairo_Window.H

33.14  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 ∗=0, unsigned col=0) 
  Adds the data value val with optional label str and color col to the chart.
- uchar autosize () const 
  Gets whether the chart will automatically adjust the bounds of the chart.
- void autosize (uchar n) 
  Sets whether the chart will automatically adjust the bounds of the chart.
- void bounds (double ∗a, double ∗b) const 
  Gets the lower and upper bounds of the chart values.
- void bounds (double a, double b) 
  Sets the lower and upper bounds of the chart values.
- void clear () 
  Removes all values from the chart.
- Fl_Chart (int X, int Y, int W, int H, const char ∗=0) 
  Create a new Fl_Chart widget using the given position, size and label string.
- void insert (int ind, double val, const char ∗=0, unsigned col=0) 
  Inserts a data value val at the given position ind.
- int maxsize () const 
  Gets the maximum number of data values for a chart.
- void maxsize (int m)
Sets the maximum number of data values for a chart.

- void replace (int ind, double val, const char *str=0, unsigned col=0)
  
  Replaces a data value \texttt{val} at the given position \texttt{ind}.

- int size () const
  
  Returns the number of data values in the chart.

- void size (int W, int H)
  
  Sets the widget size (width, height).

- Fl_Color textcolor () const
  
  Gets the chart's text color.

- void textcolor (Fl_Color n)
  
  Sets the chart's text color to \texttt{n}.

- Fl_Font textfont () const
  
  Gets the chart's text font.

- void textfont (Fl_Font s)
  
  Sets the chart's text font to \texttt{s}.

- Fl_Fontsize textsize () const
  
  Gets the chart's text size.

- void textsize (Fl_Fontsize s)
  
  Sets the chart's text size to \texttt{s}.

- Fl_Chart ()
  
  Destroys the \texttt{Fl_Chart} widget and all of its data.

Protected Member Functions

- void draw () FL_OVERRIDE
  
  Draws the \texttt{Fl_Chart} widget.

Static Protected Member Functions

- static void draw_barchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)
  
  Draws a bar chart.

- static void draw_horbarchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)
  
  Draws a horizontal bar chart.

- static void draw_linechart (int type, int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)
  
  Draws a line chart.

- static void draw_piechart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], int special, Fl_Color textcolor)
  
  Draws a pie chart.

Additional Inherited Members

33.14.1 Detailed Description

\texttt{Fl_Chart} displays simple charts.
It is provided for Forms compatibility.
Todo Refactor Fl_Chart::type() information.

The type of an Fl_Chart object can be set using type(uchar t) to:

<table>
<thead>
<tr>
<th>Chart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_BAR_CHART</td>
<td>Each sample value is drawn as a vertical bar.</td>
</tr>
<tr>
<td>FL_FILLED_CHART</td>
<td>The chart is filled from the bottom of the graph to the sample values.</td>
</tr>
<tr>
<td>FL_HORBAR_CHART</td>
<td>Each sample value is drawn as a horizontal bar.</td>
</tr>
<tr>
<td>FL_LINE_CHART</td>
<td>The chart is drawn as a polyline with vertices at each sample value.</td>
</tr>
<tr>
<td>FL_PIE_CHART</td>
<td>A pie chart is drawn with each sample value being drawn as a proportionate slice in the circle.</td>
</tr>
<tr>
<td>FL_SPECIALPIE_CHART</td>
<td>Like FL_PIE_CHART, but the first slice is separated from the pie.</td>
</tr>
<tr>
<td>FL_SPIKE_CHART</td>
<td>Each sample value is drawn as a vertical line.</td>
</tr>
</tbody>
</table>

### 33.14.2 Constructor & Destructor Documentation

#### 33.14.2.1 Fl_Chart()

Fl_Chart::Fl_Chart (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗ L = 0 )

Create a new Fl_Chart widget using the given position, size and label string. The default boxstyle is FL_NO_BOX.

**Parameters**

<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>

### 33.14.3 Member Function Documentation

#### 33.14.3.1 add()

void Fl_Chart::add (  
  double vs1,  
  double vs2,  
  ...    
)
const char * str = 0,
unsigned col = 0)

Adds the data value val with optional label str and color col to the chart.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>data value</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
<td>optional data label</td>
</tr>
<tr>
<td>in</td>
<td>col</td>
<td>optional data color</td>
</tr>
</tbody>
</table>

### 33.14.3.2 autosize() [1/2]

uchar Fl_Chart::autosize ( ) const [inline]

Gets whether the chart will automatically adjust the bounds of the chart.

Returns

non-zero if auto-sizing is enabled and zero if disabled.

### 33.14.3.3 autosize() [2/2]

void Fl_Chart::autosize (uchar n) [inline]

Sets whether the chart will automatically adjust the bounds of the chart.

Parameters

| in   | n    | non-zero to enable automatic resizing, zero to disable. |

### 33.14.3.4 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 |

### 33.14.3.5 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 |
33.14.3.6  draw()

void Fl_Chart::draw ( void ) [protected], [virtual]
Draws the Fl_Chart widget.
Implements Fl_Widget.

33.14.3.7  draw_barchart()

void Fl_Chart::draw_barchart ( int x, int y, int w, int h, int numb,
                               FL_CHART_ENTRY entries[],
                               double min, double max, int autosize,
                               int maxnumb, Fl_Color textcolor ) [static], [protected]

Draws a bar chart.
x, y, w, h is the bounding box, entries the array of numb entries, and min and max the boundaries.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x, y, w, h</th>
<th>Widget position and size</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>

33.14.3.8  draw_horbarchart()

void Fl_Chart::draw_horbarchart ( int x, int y, int w, int h, int numb,
                                 FL_CHART_ENTRY entries[],
                                 double min, double max, int autosize,
                                 int maxnumb, Fl_Color textcolor ) [static], [protected]

Draws a horizontal bar chart.
x, y, w, h is the bounding box, entries the array of numb entries, and min and max the boundaries.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>Widget position and size</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>

33.14.3.9 draw_linechart()

```cpp
def draw_linechart (int type,
                     int x,  
                     int y,  
                     int w,  
                     int h,  
                     int numb,
                     FL_CHART_ENTRY entries[],
                     double min,
                     double max,
                     int autosize,
                     int maxnumb,
                     Fl_Color textcolor) [static], [protected]
```

Draws a line chart. 

\(x, y, w, h\) is the bounding box, \(entries\) the array of \(numb\) entries, and \(min\) and \(max\) the boundaries.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>Chart type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x,y,w,h</td>
<td>Widget position and size</td>
</tr>
<tr>
<td>in</td>
<td>numb</td>
<td>Number of values</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>Array of values</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Lower boundary</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Upper boundary</td>
</tr>
<tr>
<td>in</td>
<td>autosize</td>
<td>Whether the chart autosizes</td>
</tr>
<tr>
<td>in</td>
<td>maxnumb</td>
<td>Maximal number of entries</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>Text color</td>
</tr>
</tbody>
</table>

33.14.3.10 draw_piechart()

```cpp
def draw_piechart (int x,  
                   int y,  
                   int w,  
                   int h,  
                   int numb,
                   FL_CHART_ENTRY entries[],
                   int special,
                   Fl_Color textcolor) [static], [protected]
```
Draws a pie chart.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y,w,h</th>
<th>bounding box</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>numb</td>
<td>number of chart entries</td>
</tr>
<tr>
<td>in</td>
<td>entries</td>
<td>array of chart entries</td>
</tr>
<tr>
<td>in</td>
<td>special</td>
<td>special (?)</td>
</tr>
<tr>
<td>in</td>
<td>textcolor</td>
<td>text color</td>
</tr>
</tbody>
</table>

### 33.14.3.11 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>

### 33.14.3.12 maxsize()

```c
void Fl_Chart::maxsize (  
    int m )
```

Sets the maximum number of data values for a chart. If you do not call this method then the chart will be allowed to grow to any size depending on available memory.

Parameters

| in  | m | maximum number of data values allowed. |

### 33.14.3.13 replace()

```c
void Fl_Chart::replace (  
    int ind,  
    double val,  
    const char ∗str = 0,  
    unsigned col = 0 )
```

Replaces 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>
</table>
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val</code></td>
<td><code>in</code></td>
<td>data value</td>
</tr>
<tr>
<td><code>str</code></td>
<td><code>in</code></td>
<td>optional data label</td>
</tr>
<tr>
<td><code>col</code></td>
<td><code>in</code></td>
<td>optional data color</td>
</tr>
</tbody>
</table>

### 33.14.3.14 size()

```cpp
void Fl_Chart::size (  
    int W,  
    int H ) [inline]
```

Sets the widget size (width, height).
This is the same as calling `Fl_Widget::size(int W, int H);`

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td><code>W,H</code></td>
<td>new width and height of the widget</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- `Fl_Chart.H`
- `Fl_Chart.cxx`

### 33.15 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.

### 33.15.1 Detailed Description

For internal use only.
The documentation for this struct was generated from the following file:

- `Fl_Chart.H`

### 33.16 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:
Public Member Functions

- **int add (char *s)**
  
  Add a new unchecked line to the end of the browser.

- **int add (char *s, int b)**
  
  Add a new line to the end of the browser.

- **int add (const char *s)**
  
  See int Fl_Check_Browser::add(char *s)

- **int add (const char *s, int b)**
  
  See int Fl_Check_Browser::add(char *s)

- **void check_all ()**
  
  Sets all the items checked.

- **void check_none ()**
  
  Sets all the items unchecked.

- **int checked (int item) const**
  
  Gets the current status of item item.

- **void checked (int item, int b)**
  
  Sets the check status of item item to b.

- **void clear ()**
  
  Remove every item from the browser.

- **Fl_Check_Browser (int x, int y, int w, int h, const char *l=0)**
  
  The constructor makes an empty browser.

- **void * item_at (int index) const FL_OVERRIDE**
  
  This method must be provided by the subclass to return the item for the specified index.

- **void item_swap (int ia, int ib) FL_OVERRIDE**
  
  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) FL_OVERRIDE**
  
  *Handles the event within the normal widget bounding box.*

- **void item_draw (void *, int, int, int, int) const FL_OVERRIDE**
  
  *This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.*

- **void * item_first () const FL_OVERRIDE**
  
  *This method must be provided by the subclass to return the first item in the list.*

- **int item_height (void *) const FL_OVERRIDE**
  
  *This method must be provided by the subclass to return the height of item in pixels.*

- **void * item_next (void *) const FL_OVERRIDE**
  
  *This method must be provided by the subclass to return the item in the list after item.*

- **void * item_prev (void *) const FL_OVERRIDE**
  
  *This method must be provided by the subclass to return the item in the list before item.*

- **void item_select (void *, int) FL_OVERRIDE**
  
  *This method must be implemented by the subclass if it supports multiple selections; sets the selection state to val for the item.*

- **int item_selected (void *) const FL_OVERRIDE**
  
  *This method must be implemented by the subclass if it supports multiple selections; returns the selection state for item.*

- **const char * item_text (void *) (item) const FL_OVERRIDE**
  
  *This optional method returns a string (label) that may be used for sorting.*

- **int item_width (void *) const FL_OVERRIDE**
  
  *This method must be provided by the subclass to return the width of the item in pixels.*

Additional Inherited Members

33.16.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

33.16.2 Member Function Documentation

33.16.2.1 add() [1/2]

int Fl_Check_Browser::add (char * s )

Add a new unchecked line to the end of the browser.

See also

add(char *s, int b)

33.16.2.2 add() [2/2]

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.
33.16.2.3 handle()

```c++
int Fl_Check_Browser::handle ( int event ) [protected], [virtual]
```
Handles the event within the normal widget bounding box.

**Parameters**

- **event**
  The event to process.

**Returns**

- 1 if event was processed, 0 if not.

Reimplemented from Fl_Browser_.

---

33.16.2.4 item_at()

```c++
void * Fl_Check_Browser::item_at ( int index ) const [virtual]
```
This method must be provided by the subclass to return the item for the specified index.

**Parameters**

- **index**
  The index of the item to be returned.

**Returns**

- The item at the specified index.

Reimplemented from Fl_Browser_.

---

33.16.2.5 item_draw()

```c++
void Fl_Check_Browser::item_draw ( void * item, int X, int Y, int W, int H ) const [protected], [virtual]
```
This method must be provided by the subclass to draw the item in the area indicated by X, Y, W, H.

Implements Fl_Browser_.

---

33.16.2.6 item_first()

```c++
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_.

---
### 33.16.2.7  item_height()

```cpp
int Fl_Check_Browser::item_height (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the height of item in pixels. Allow for two additional pixels for the list selection box.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item whose height is returned.</td>
</tr>
</tbody>
</table>

**Returns**

The height of the specified item in pixels.

See also

- `item_height()`, `item_width()`, `item_quick_height()`

Implements Fl_Browser_.

### 33.16.2.8  item_next()

```cpp
void * Fl_Check_Browser::item_next (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list after item.

See also

- `item_first()`, `item_next()`, `item_last()`, `item_prev()`

Implements Fl_Browser_.

### 33.16.2.9  item_prev()

```cpp
void * Fl_Check_Browser::item_prev (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list before item.

See also

- `item_first()`, `item_next()`, `item_last()`, `item_prev()`

Implements Fl_Browser_.

### 33.16.2.10  item_select()

```cpp
void Fl_Check_Browser::item_select (void *item, int val) [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The item to be selected</td>
</tr>
<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_.

### 33.16.2.11 item_selected()

```c
int Fl_Check_Browser::item_selected (void *item) const [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`. The method should return 1 if `item` is selected, or 0 otherwise.

**Parameters**
- `in` `item` The item to test.

Reimplemented from Fl_Browser_.

### 33.16.2.12 item_swap()

```c
void Fl_Check_Browser::item_swap (void *a, void *b) [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.

**Parameters**
- `in` `a`, `b` The two items to be swapped.

Reimplemented from Fl_Browser_.

### 33.16.2.13 item_text()

```c
const char *Fl_Check_Browser::item_text (void *item) const [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

**Parameters**
- `in` `item` The item whose label text is returned.

**Returns**
- The item's text label. (Can be NULL if blank)

Reimplemented from Fl_Browser_.

### 33.16.2.14 item_width()

```c
int Fl_Check_Browser::item_width (void *item) const [protected], [virtual]
```

This method must be provided by the subclass to return the width of the `item` in pixels. Allow for two additional pixels for the list selection box.
Parameters

| in | item | The item whose width is returned. |

Returns

The width of the item in pixels.

Implements Fl_Browser_.

33.16.2.15 nitems()

int Fl_Check_Browser::nitems ( ) const [inline]

Returns how many lines are in the browser.
The last line number is equal to this.

33.16.2.16 remove()

int Fl_Check_Browser::remove ( int item )

Remove line n and make the browser one line shorter.
Returns the number of lines left in the browser.
The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

33.17 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

33.17.1 Detailed Description

A button with a "checkmark" to show its status.

![Fl_Check_Button](image.png)

Figure 33.5 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.

### 33.17.2 Constructor & Destructor Documentation

#### 33.17.2.1 Fl_Check_Button()

Fl_Check_Button::Fl_Check_Button (  
  int \texttt{X},  
  int \texttt{Y},  
  int \texttt{W},  
  int \texttt{H},  
  \texttt{const char} \ast \texttt{L = 0} )  

Creates a new Fl_Check_Button widget using the given position, size, and label string. The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark. The selection_color() sets the color of the checkmark. Default is FL_FOREGROUND_COLOR (usually black). You can use down_box() to change the box type of the checkmark. Default is FL_DOWN_BOX.

Parameters

| in | X,Y,W,H | position and size of the widget |
| in | L       | widget label, default is no label |

### 33.18 Fl_Choice Class Reference

A button that is used to pop up a menu.

```
#include <Fl_Choice.H>
```

Inheritance diagram for Fl_Choice:

```
Fl_Widget
  ↓
Fl_Menu_
  ↓
Fl_Choice
  ↓
Fl_Scheme_Choice
```

**Public Member Functions**

- **Fl_Choice** (int \texttt{X}, int \texttt{Y}, int \texttt{W}, int \texttt{H}, const char *\texttt{L=0})  
  *Create a new Fl_Choice widget using the given position, size and label string.*

- int handle (int) \texttt{FL_OVERRIDE}
  *Handles the specified event.*

- int value () const

---

Generated by Doxygen
33.18 Fl_Choice Class Reference

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 () FL_OVERRIDE
  Draws the widget.

Additional Inherited Members

33.18.1 Detailed Description

A button that is used to pop up a menu.

![Fl_Choice Button](image)

Figure 33.6 Fl_Choice

This is a button that, when pushed, pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects. Motif calls this an OptionButton.

The only difference between this and a Fl_Menu_Button is that the name of the most recent chosen menu item is displayed inside the box, while the label is displayed outside the box. However, since the use of this is most often to control a single variable rather than do individual callbacks, some of the Fl_Menu_Button methods are redescribed here in those terms.

When the user clicks a menu item, value() is set to that item and then:

- The item’s callback is done if one has been set; the Fl_Choice is passed as the Fl_Widget* argument, along with any userdata configured for the callback.

- If the item does not have a callback, the Fl_Choice widget’s callback is done instead, along with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

All three mouse buttons pop up the menu. The Forms behavior of the first two buttons to increment/decrement the choice is not implemented. This could be added with a subclass, however.

The menu will also pop up in response to shortcuts indicated by putting a ‘&’ character in the label(). See Fl_Button::shortcut(int s) for a description of this.

Typing the shortcut() of any of the items will do exactly the same as when you pick the item with the mouse. The ‘&’ character in item names are only looked at when the menu is popped up, however.

The inherited Fl_Widget::changed() and related methods can be used as follows:

- int Fl_Widget::changed() const This value is true when the user picks a different value. It is turned off by value() and just before doing a callback (the callback can turn it back on if desired).

- void Fl_Widget::set_changed() This method sets the changed() flag.

- void Fl_Widget::clear_changed() This method clears the changed() flag.
The inherited `Fl_Menu_::down_box()` methods can be used as follows:

- `Fl_Boxtype Fl_Menu_::down_box() const` Gets the current down box, which is used when the menu is popped up. The default down box type is `FL_DOWN_BOX`.

- `void Fl_Menu_::down_box(Fl_Boxtype b)` Sets the current down box type to `b`.

Simple example:

```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();
}
```

### 33.18.2 Constructor & Destructor Documentation

#### 33.18.2.1 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>

### 33.18.3 Member Function Documentation

#### 33.18.3.1 draw()

`void Fl_Choice::draw ( ) [protected], [virtual]`  
Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:
`Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar s->draw(); // calls Fl_Scrollbar::draw()`  
Implements `Fl_Widget`.  

Generated by Doxygen
33.18.3.2 handle()

```cpp
int Fl_Choice::handle (  
    int event) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

**See also**

- Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl.Scheme_Choice.

33.18.3.3 value() [1/3]

```cpp
int Fl_Choice::value () const [inline]
```

Gets the index of the last item chosen by the user.
The index is -1 initially.

33.18.3.4 value() [2/3]

```cpp
int Fl_Choice::value (  
    const Fl_Menu_Item * v)
```

Sets the currently selected value using a pointer to menu item.
Changing the selected value causes a redraw().

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>v</th>
<th>pointer to menu item in the menu item array.</th>
</tr>
</thead>
</table>

**Returns**

- non-zero if the new value is different to the old one.

33.18.3.5 value() [3/3]

```cpp
int Fl_Choice::value (  
    int v)
```

Sets the currently selected value using the index into the menu item array.
Changing the selected value causes a redraw().
Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & v \\
\text{index of value in the menu item array.} \\
\hline
\end{array}
\]

Returns

non-zero if the new value is different to the old one.

The documentation for this class was generated from the following files:

- Fl_Choice.H
- Fl_Choice.cxx

33.19 Fl_Clock Class Reference

This widget provides a round analog clock display.

```
#include <Fl_Clock.H>
```

Inheritance diagram for Fl_Clock:

```
Fl_Widget
   \|-- Fl_Clock_Output
       \|-- Fl_Clock
           \|-- Fl_Round_Clock
```

Public Member Functions

- `Fl_Clock (int X, int Y, int W, int H, const char *L=0)`
  Create an Fl_Clock widget using the given position, size, and label string.

- `Fl_Clock (uchar t, int X, int Y, int W, int H, const char *L)`
  Create an Fl_Clock widget using the given clock type `t`, position, size, and label string.

- `int handle (int)` FL_OVERRIDE
  Handles the specified event.

- `~Fl_Clock ()`
  The destructor removes the clock.

Additional Inherited Members

33.19.1 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.
33.19.19 Fl_Clock Class Reference

33.19.2 Constructor & Destructor Documentation

33.19.2.1 Fl_Clock() [1/2]

Fl_Clock::Fl_Clock (  
   int X,  
   int Y,  
   int W,  
   int H,  
   const char * L = 0 )

Create an Fl_Clock widget using the given position, size, and label string.
The default clock type is FL_SQUARE_CLOCK and the default boxtype is FL_UP_BOX.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
</tr>
</tbody>
</table>

33.19.2.2 Fl_Clock() [2/2]

Fl_Clock::Fl_Clock (  
   uchar t,  
   int X,  
   int Y,  
   int W,  
   int H,  
   const char * L = 0 )

See also
class Fl_Clock_Output

Figure 33.7 FL_SQUARE_CLOCK type

Figure 33.8 FL_ROUND_CLOCK type
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t</code></td>
<td>type of clock: <code>FL_ROUND_CLOCK</code> or <code>FL_SQUARE_CLOCK</code> (0)</td>
</tr>
<tr>
<td><code>X,Y,W,H</code></td>
<td>position and size of the widget</td>
</tr>
<tr>
<td><code>L</code></td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

See also

- class `Fl_Clock_Output`

### 33.19.3 Member Function Documentation

#### 33.19.3.1 `handle()`

```cpp
int Fl_Clock::handle (int event) [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

### Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the event was not used or understood</td>
</tr>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

- `Fl_Event`

Reimplemented from `Fl_Widget`.

The documentation for this class was generated from the following files:

- `Fl_Clock.H`
- `Fl_Clock.cxx`

### 33.20 `Fl_Clock_Output` Class Reference

This widget can be used to display a program-supplied time.
```c
#include <Fl_Clock.H>

Inheritance diagram for Fl_Clock_Output:

```
```
```
```

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 () FL_OVERRIDE**
  
  Draw clock with current position and size.

- **void draw (int X, int Y, int W, int H)**
  
  Draw clock with the given position and size.

Additional Inherited Members

33.20.1 Detailed Description

This widget can be used to display a program-supplied time.
The time shown on the clock is not updated. To display the current time, use Fl_Clock instead.
Values for clock type() (#include <FL/Clock.H>):

```c
#define FL_SQUARE_CLOCK 0 // Square Clock variant
#define FL_ROUND_CLOCK 1 // Round Clock variant
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK // An analog clock is square
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK // Not yet implemented
```

### 33.20.2 Constructor & Destructor Documentation

#### 33.20.2.1 Fl_Clock_Output()

```c
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

<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>

### 33.20.3 Member Function Documentation
33.20.3.1 draw [1/2]

```cpp
void Fl_Clock_Output::draw (void) [protected], [virtual]
```

Draw clock with current position and size.
Implements Fl_Widget.

33.20.3.2 draw [2/2]

```cpp
void Fl_Clock_Output::draw (int X, int Y, int W, int H) [protected]
```

Draw clock with the given position and size.

Parameters

| in X,Y,W,H | position and size |

33.20.3.3 hour()

```cpp
int Fl_Clock_Output::hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).
See also

value(), minute(), second()

33.20.3.4 minute()

```cpp
int Fl_Clock_Output::minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).
See also

value(), hour(), second()

33.20.3.5 second()

```cpp
int Fl_Clock_Output::second ( ) const [inline]
```

Returns the displayed second (0 to 60, 60=leap second).
See also

value(), hour(), minute()

33.20.3.6 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>0</th>
<th>no shadows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>draw shadows of hands (default)</td>
</tr>
</tbody>
</table>

### 33.20.3.7 shadow() [2/2]

```cpp
def 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 |

### 33.20.3.8 value() [1/3]

```cpp
def 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)

### 33.20.3.9 value() [2/3]

```cpp
def 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()

### 33.20.3.10 value() [3/3]

```cpp
def Fl_Clock_Output::value (ulong v) |
Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).```
Parameters

\begin{verbatim}
in v seconds since epoch
\end{verbatim}

See also

\begin{verbatim}
value()
\end{verbatim}

The documentation for this class was generated from the following files:

- Fl_Clock.H
- Fl_Clock.cxx

## 33.21 Fl_Color_Chooser Class Reference

The Fl_Color_Chooser widget provides a standard RGB color chooser.

```c
#include <Fl_Color_Chooser.H>
```

Inheritance diagram for Fl_Color_Chooser:

![Inheritance diagram](#)

### Public Member Functions

- double b () const
  
  *Returns the current blue value.*

- Fl_Color_Chooser (int X, int Y, int W, int H, const char *L=0)
  
  *Creates a new Fl_Color_Chooser widget using the given position, size, and label string.*

- double g () const
  
  *Returns the current green value.*

- int handle (int e) FL_OVERRIDE
  
  *Handles all events received by this widget.*

- int hsv (double H, double S, double V)
  
  *Set the hsv values.*

- double hue () const
  
  *Returns the current hue.*

- int mode ()
  
  *Returns which Fl_Color_Chooser variant is currently active.*

- void mode (int newMode)
  
  *Set which Fl_Color_Chooser variant is currently active.*

- double r () const
  
  *Returns the current red value.*

- int rgb (double R, double G, double B)
  
  *Sets the current rgb color values.*

- double saturation () const
  
  *Returns the saturation.*

- double value () const
  
  *Returns the value/brightness.*
Static Public Member Functions

- static void hsv2rgb (double H, double S, double V, double &R, double &G, double &B)
  
  This static method converts HSV colors to RGB colorspace.

- static void rgb2hsv (double R, double G, double B, double &H, double &S, double &V)
  
  This static method converts RGB colors to HSV colorspace.

Related Functions

(Note that these are not member functions.)

- int fl_color_chooser (const char ∗name, double &r, double &g, double &b, int cmode)
  
  Pops up a window to let the user pick an arbitrary RGB color.

- int fl_color_chooser (const char ∗name, uchar &r, uchar &g, uchar &b, int cmode)
  
  Pops up a window to let the user pick an arbitrary RGB color.

Additional Inherited Members

33.21.1 Detailed Description

The Fl_Color_Chooser widget provides a standard RGB color chooser.

![fl_color_chooser()](image)

You can place any number of the widgets into a panel of your own design. The diagram shows the widget as part of a color chooser dialog created by the fl_color_chooser() function. The Fl_Color_Chooser widget contains the hue box, value slider, and rgb input fields from the above diagram (it does not have the color chips or the Cancel or OK buttons). The callback is done every time the user changes the rgb value. It is not done if they move the hue control in a way that produces the same rgb value, such as when saturation or value is zero.

The fl_color_chooser() function pops up a window to let the user pick an arbitrary RGB color. They can pick the hue and saturation in the "hue box" on the left (hold down CTRL to just change the saturation), and the brightness using the vertical slider. Or they can type the 8-bit numbers into the RGB Fl_Value_Input fields, or drag the mouse across them to adjust them. The pull-down menu lets the user set the input fields to show RGB, HSV, or 8-bit RGB (0 to 255).

The user can press CTRL-C to copy the currently selected color value as text in RGB hex format with leading zeroes to the clipboard, for instance FL_GREEN would be '00FF00' (since FLTK 1.4.0). fl_color_chooser() returns non-zero if the user picks ok, and updates the RGB values. If the user picks cancel or closes the window this returns zero and leaves RGB unchanged.

If you use the color chooser on an 8-bit screen, it will allocate all the available colors, leaving you no space to exactly represent the color the user picks! You can however use fl_rectf() to fill a region with a simulated color using dithering.

Callback reasons can be FL_REASON_DRAGGED, FL_REASON_CHANGED, or FL_REASON_RESELECTED.

33.21.2 Constructor & Destructor Documentation
33.21.2.1 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

<table>
<thead>
<tr>
<th></th>
<th>X,Y,W,H</th>
<th>position and size of the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

33.21.3 Member Function Documentation

33.21.3.1 b()

double Fl_Color_Chooser::b ( ) const [inline]  

Returns the current blue value.  
$0 \leq b \leq 1.$

33.21.3.2 g()

double Fl_Color_Chooser::g ( ) const [inline]  

Returns the current green value.  
$0 \leq g \leq 1.$

33.21.3.3 handle()

int Fl_Color_Chooser::handle {  
    int e ) [virtual]  

Handles all events received by this widget.  
This specific handle() method processes the standard ‘copy’ function as seen in other input widgets. It copies the current color value to the clipboard as a string in RGB format (‘RRGGBB’).  
This format is independent of the Fl_Color_Chooser display format setting. No other formats are supplied.  
The keyboard events handled are:

- ctrl-c  
- ctrl-x  
- ctrl-Insert

All other events are processed by the parent class Fl_Group.  
This enables the user to choose a color value, press ctrl-c to copy the value to the clipboard and paste it into a color selection widget in another application window or any other text input (e.g. a preferences dialog or an editor).

Note

Keyboard event handling by the current focus widget has priority, hence moving the focus to one of the buttons or selecting text in one of the input widgets effectively disables this special method.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>current event</th>
</tr>
</thead>
</table>
Returns

1 if event has been handled, 0 otherwise

See also

\texttt{Fl\_Group::handle(int)}

Reimplemented from \texttt{Fl\_Widget}.

### 33.21.3.4 hsv()

```cpp
int Fl\_Color\_Chooser::hsv ( double H, double S, double V )
```

Set the hsv values.
The passed values are clamped (or for hue, modulus 6 is used) to get legal values. Does not do the callback.

Parameters

- \texttt{in H,S,V} color components.

Returns

1 if a new hsv value was set, 0 if the hsv value was the previous one.

### 33.21.3.5 hsv2rgb()

```cpp
void Fl\_Color\_Chooser::hsv2rgb ( double H, double S, double V, double & R, double & G, double & B ) \[static\]
```

This \textit{static} method converts HSV colors to RGB colorspace.

Parameters

- \texttt{in H,S,V} color components
- \texttt{out R,G,B} color components

### 33.21.3.6 hue()

```cpp
double Fl\_Color\_Chooser::hue ( ) const \[inline\]
```

Returns the current hue.

0 <= hue < 6. Zero is red, one is yellow, two is green, etc. \textit{This value is convenient for the internal calculations - some other systems consider hue to run from zero to one, or from 0 to 360.}

### 33.21.3.7 mode{} [1/2]

```cpp
int Fl\_Color\_Chooser::mode ( ) \[inline\]
```

Returns which \texttt{Fl\_Color\_Chooser} variant is currently active.
Returns

color modes are rgb(0), byte(1), hex(2), or hsv(3)

33.21.3.8  mode() [2/2]

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) |

33.21.3.9   r()

double Fl_COLOR_CHOOSER::r ( ) const [inline]
Returns the current red value.
0 <= r <= 1.

33.21.3.10  rgb()

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.

33.21.3.11  rgb2hsv()

void Fl_COLOR_CHOOSER::rgb2hsv (  
    double R,  
    double G,  
    double B,  
    double & H,  
    double & S,  
    double & V ) [static]
This static method converts RGB colors to HSV colorspace.

Parameters

| in   | R,G,B   | color components |

| out  | H,S,V   | color components |

Generated by Doxygen
33.21.3.12 saturation()

double Fl_Color_Chooser::saturation ( ) const [inline]

Returns the saturation.
\[0 \leq \text{saturation} \leq 1.\]

33.21.3.13 value()

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

33.22 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

#include <Fl_Copy_Surface.H>

Inheritance diagram for Fl_Copy_Surface:

\[
\text{Fl_Surface_Device} \\
\text{Fl_Widget_Surface} \\
\text{Fl_Copy_Surface}
\]

Public Member Functions

- Fl_Copy_Surface (int w, int h)
  
  the constructor

- int h ()
  
  Returns the pixel height of the copy surface.

- bool is_current () FL_OVERRIDE
  
  Is this surface the current drawing surface?

- void origin (int *x, int *y) FL_OVERRIDE
  
  Computes the coordinates of the current origin of graphics functions.

- void origin (int x, int y) FL_OVERRIDE
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- int printable_rect (int *w, int *h) FL_OVERRIDE
  
  Computes the width and height of the drawable area of the drawing surface.

- void set_current () FL_OVERRIDE
  
  Make this surface the current drawing surface.

- int w ()
  
  Returns the pixel width of the copy surface.

- ~Fl_Copy_Surface ()
  
  the destructor
33.22 Fl_Copy_Surface Class Reference

Protected Member Functions

- void **translate** (int x, int y) **FL_OVERRIDE**
  
  Translates the current graphics origin accounting for the current rotation.

- void **untranslate** () **FL_OVERRIDE**
  
  Undoes the effect of a previous translate() call.

Additional Inherited Members

33.22.1 Detailed Description

Supports copying of graphical data to the clipboard.

After creation of an Fl_Copy_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the clipboard. It's possible to draw widgets (using Fl_Copy_Surface::draw() ) or to use any of the Drawing functions or the Color & Font functions. Finally, delete the Fl_Copy_Surface object to load the clipboard with the graphical data.

Fl_Gl_Window’s can be copied to the clipboard as well.

Usage example:

```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.

33.22.2 Constructor & Destructor Documentation

33.22.2.1 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 |

33.22.3 Member Function Documentation

33.22.3.1 is_current()

```c
bool Fl_Copy_Surface: :is_current ( ) [virtual]
```

Is this surface the current drawing surface?
Reimplemented from \texttt{Fl\_Surface\_Device}.

\subsection{33.22.3.2 origin\([1/2]\)](origin)

\begin{verbatim}
void Fl_Copy_Surface::origin (  
    int * x,  
    int * y ) [virtual]
\end{verbatim}

Computes the coordinates of the current origin of graphics functions.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>x, y</th>
</tr>
</thead>
</table>
| If non-null, \(x\) and \(y\) are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from \texttt{Fl\_Widget\_Surface}.

\subsection{33.22.3.3 origin\([2/2]\)](origin)

\begin{verbatim}
void Fl_Copy_Surface::origin (  
    int x,  
    int y ) [virtual]
\end{verbatim}

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous \texttt{printable\_rect()} call. That is, \[ \text{printable} \leftarrow \text{printable\_rect(&w, &h)}; \text{origin}(w/2, 0); \] sets the graphics origin at the top center of the drawable area. Successive \texttt{origin()} calls don't combine their effects. \texttt{Origin()} calls are not affected by \texttt{rotate()} calls (for classes derived from \texttt{Fl\_Paged\_Device}).

**Parameters**

| in | x, y |
| Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from \texttt{Fl\_Widget\_Surface}.

\subsection{33.22.3.4 printable\_rect()}

\begin{verbatim}
int Fl_Copy_Surface::printable_rect (  
    int * w,  
    int * h ) [virtual]
\end{verbatim}

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to \texttt{origin()}. If the object is derived from class \texttt{Fl\_Paged\_Device}, values account for the user-selected paper type and print orientation and are changed by \texttt{scale()} calls.

**Returns**

0 if OK, non-zero if any error

Reimplemented from \texttt{Fl\_Widget\_Surface}.

\subsection{33.22.3.5 set\_current()}

\begin{verbatim}
void Fl_Copy_Surface::set_current (  
    void ) [virtual]
\end{verbatim}

Make this surface the current drawing surface. This surface will receive all future graphics requests. \texttt{Starting} from FLTK 1.4.0, the preferred API to change the current drawing surface is \texttt{Fl\_Surface\_Device::push\_current()} / \texttt{Fl\_Surface\_Device::pop\_current()}.  

\section*{Class Documentation}
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.

33.22.3.6 translate()

```cpp
void Fl_Copy_Surface::translate (  
    int x,  
    int y ) [protected], [virtual]
```

Translates the current graphics origin accounting for the current rotation. Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects. Reimplemented from Fl_Widget_Surface.

33.22.3.7 untranslate()

```cpp
void Fl_Copy_Surface::untranslate {  
    void ) [protected], [virtual]
```

Undoes the effect of a previous translate() call. Reimplemented from Fl_Widget_Surface.

The documentation for this class was generated from the following files:

- Fl_Copy_Surface.H
- Fl_Copy_Surface.cxx

33.23 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.

```cpp
#include <Fl_Counter.H>
```

Inheritance diagram for Fl_Counter:

```
Fl_Widget
  |  
  v  
Fl_Valuator
  |  
  v  
Fl_Counter
  |  
  v  
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)** FL_OVERRIDE
  Handles the specified event.
- **void lstep (double a)**
  Sets the increment for the large step buttons.
- **double step () const**
  Returns the increment for normal step buttons.
- **void step (double a)**
  Sets the increment for the normal step buttons.
- **void step (double a, double b)**
  Sets the increments for the normal and large step buttons.
- **Fl_Color textcolor () const**
  Gets the font color.
- **void textcolor (Fl_Color s)**
  Sets the font color to s.
- **Fl_Font textfont () const**
  Gets the text font.
- **void textfont (Fl_Font s)**
  Sets the text font to s.
- **Fl_Fontsize textsize () const**
  Gets the font size.
- **void textsize (Fl_Fontsize s)**
  Sets the font size to s.
- **~Fl_Counter ()**
  Destroys the valuator.

Protected Member Functions

- **void arrow_widths (int &w1, int &w2)**
  Compute sizes (widths) of arrow boxes.
- **void draw () FL_OVERRIDE**
  Draws the widget.

Additional Inherited Members

33.23.1 Detailed Description

Controls a single floating point value with button (or keyboard) arrows.
Double arrows buttons achieve larger steps than simple arrows.

See also

Fl_Spinner for value input with vertical step arrows.

![Figure 33.12 Fl_Counter](image)

The type of an Fl_Counter object can be set using Fl_Widget::type(uchar) to:

- **FL_NORMAL_COUNTER**: Displays a counter with 4 arrow buttons.
- **FL_SIMPLE_COUNTER**: Displays a counter with only 2 arrow buttons.
33.23 Fl_Counter Class Reference

33.23.2 Constructor & Destructor Documentation

33.23.2.1 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>

33.23.3 Member Function Documentation

33.23.3.1 arrow_widths()

void Fl_Counter::arrow_widths (  
    int & w1,
    int & w2 ) [protected]

Compute sizes (widths) of arrow boxes.
This method computes the two sizes of the arrow boxes of Fl_Counter. You can override it in a subclass if you want
to draw fancy arrows or change the layout. However, the basic layout is fixed and can't be changed w/o overriding
the draw() and handle() methods.

Basic layout:

```
+------+-----+-------------+-----+------+
| « | < | value | > | » |
+------+-----+-------------+-----+------+
```

The returned value w2 should be zero if the counter type() is FL_SIMPLE_COUNTER.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>w1</th>
<th>width of single arrow box</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>w2</td>
<td>width of double arrow box</td>
</tr>
</tbody>
</table>

33.23.3.2 draw()

void Fl_Counter::draw ( ) [protected], [virtual]

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call 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.
### 33.23.3 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**

- 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`.

### 33.23.4 lstep()

```cpp
void Fl_Counter::lstep (double a) [inline]
```

Sets the increment for the large step buttons.
The default value is 1.0.

**Parameters**

| in  | a | large step increment. |

### 33.23.5 step() [1/2]

```cpp
void Fl_Counter::step (double a) [inline]
```

Sets the increment for the normal step buttons.

**Parameters**

| in  | a | normal step increment. |

### 33.23.6 step() [2/2]

```cpp
void Fl_Counter::step (double a, double b) [inline]
```

Sets the increments for the normal and large step buttons.
Parameters

\[
\text{in } a, b \text{ normal and large step increments.}
\]

The documentation for this class was generated from the following files:

- Fl_Counter.H
- Fl_Counter.cxx

### 33.24 Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.

```c
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Device_Plugin:

```
Fl_Device_Plugin
    ↘
    ᵇ
Fl_Plugin
```

### Public Member Functions

- `Fl_Device_Plugin(\text{const char }*\text{pluginName})`
  
  The constructor.

- `virtual \text{const char }*\text{klass}()`
  
  Returns the class name.

- `virtual \text{const char }*\text{name}()=0`
  
  Returns the plugin name.

- `virtual int\text{print}(\text{Fl_Widget }*\text{w})=0`
  
  Prints a widget.

- `virtual \text{Fl_RGB/Image }*\text{rectangle_capture}(\text{Fl_Widget }*\text{widget, int x, int y, int w, int h})=0`
  
  Captures a rectangle of a widget as an image.

### Static Public Member Functions

- `static \text{Fl_Device_Plugin }*\text{opengl_plugin}()`
  
  Returns the OpenGL plugin.

### 33.24.1 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.

### 33.24.2 Member Function Documentation

#### 33.24.2.1 rectangle_capture()

```c
virtual \text{Fl_RGB/Image }*\text{Fl_Device_Plugin::rectangle_capture}\{
    \text{Fl_Widget }*\text{widget,}
    \text{int } x,\text{ }
    \text{int } y,
```
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

33.25 FL_Dial Class Reference

The FL_Dial widget provides a circular dial to control a single floating point value.

\include <Fl_Dial.H>

Inheritance diagram for FL_Dial:

```
public:
  short angle1() const
  
  void angle1(short a)
  
  short angle2() const
  
  void angle2(short a)
  
  void angles(short a, short b)
  
  FL_Dial(int x, int y, int w, int h, const char ∗l=0)

private:
  int handle(int)
```

Public Member Functions

- short angle1() const
  
  Sets Or gets the angles used for the minimum and maximum values.

- void angle1(short a)
  
  See short angle1() const.

- short angle2() const
  
  See short angle1() const.

- void angle2(short a)
  
  See short angle1() const.

- void angles(short a, short b)
  
  See short angle1() const.

- FL_Dial(int x, int y, int w, int h, const char ∗l=0)
  
  Creates a new FL_Dial widget using the given position, size, and label string.

- int handle(int) FL_OVERRIDE
  
  Allow subclasses to handle event based on current position and size.

Protected Member Functions

- void draw() FL_OVERRIDE
  
  Draws dial at current position and size.

- void draw(int X, int Y, int W, int H)
  
  Draws dial at given position and size.

- int handle(int event, int X, int Y, int W, int H)
  
  Allows subclasses to handle event based on given position and size.
Additional Inherited Members

33.25.1 Detailed Description

The Fl.Dial widget provides a circular dial to control a single floating point value.

![Fl.Dial widget](image)

Figure 33.13 Fl.Dial

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.

33.25.2 Constructor & Destructor Documentation

33.25.2.1 Fl.Dial()

```cpp
Fl.Dial::Fl.Dial ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗ l = 0 )
```

Creates a new Fl.Dial widget using the given position, size, and label string. The default type is FL_NORMAL_DIAL.

33.25.3 Member Function Documentation

33.25.3.1 angle1()

```cpp
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.

33.25.3.2 draw() [1/2]

```cpp
void Fl.Dial::draw ( 
    void ) [protected], [virtual]
```

Draws dial at current position and size. Implements Fl_Widget.

33.25.3.3 draw() [2/2]

```cpp
void Fl.Dial::draw ( 
    int X, 
    int Y,
```
Draws dial at given position and size.

Parameters

| in  | X,Y,W,H | position and size |

### 33.25.3.4 handle() [1/2]

```cpp
int Fl_Dial::handle (  
    int event,  
    int X,  
    int Y,  
    int W,  
    int H ) [protected]
```

Allows subclasses to handle event based on given position and size.

Parameters

| in  | event,X,Y,W,H | event to handle, related position and size. |

### 33.25.3.5 handle() [2/2]

```cpp
int Fl_Dial::handle (  
    int e ) [virtual]
```

Allow subclasses to handle event based on current position and size. Reimplemented from `Fl_Widget`.

The documentation for this class was generated from the following files:

- `Fl_Dial.H`
- `Fl_Dial.cxx`

### 33.26 Fl_Display_Device Class Reference

A display to which the computer can draw.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Display_Device:

```
Fl_Surface_Device
  ↓
Fl_Display_Device
```

**Static Public Member Functions**

- static `Fl_Display_Device * display_device ()`
  
  Returns a pointer to the unique display device.
Additional Inherited Members

33.26.1 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

33.27 Fl_Double_Window Class Reference

The Fl_Double_Window provides a double-buffered window.

```
#include <Fl_Double_Window.H>
```

Inheritance diagram for Fl_Double_Window:

```
Fl_Widget
     |
     v
Fl_Group
     |
     v
Fl_Window
     |
     v
Fl_Double_Window
     |
     v
Fl_Cairo_Window Fl_Overlay_Window
```

Public Member Functions

- `Fl_Double_Window * as_double_window () FL_OVERRIDE`
  
  Return non-null if this is an Fl_Overlay_Window object.

- `Fl_Double_Window (int W, int H, const char *l=0)`
  
  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.

- `Fl_Double_Window (int X, int Y, int W, int H, const char *l=0)`
  
  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char *label = 0)

- `void flush () FL_OVERRIDE`
  
  Forces the window to be drawn, this window is also made current and calls draw().

- `void hide () FL_OVERRIDE`
  
  Makes a widget invisible.

- `void resize (int, int, int, int) FL_OVERRIDE`
  
  Changes the size or position of the widget.

- `void show () FL_OVERRIDE`
  
  Makes a widget visible.

- `void show (int a, char **b)`
  
  Same as Fl_Window::show(int a, char **b)

- `~Fl_Double_Window ()`
  
  The destructor also deletes all the children.
Additional Inherited Members

33.27.1 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.

33.27.2 Constructor & Destructor Documentation

33.27.2.1 ~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.

33.27.3 Member Function Documentation

33.27.3.1 as_double_window()

Fl_Double_Window * Fl_Double_Window::as_double_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Overlay_Window object.
Reimplemented from Fl_Window.

33.27.3.2 flush()

void Fl_Double_Window::flush ( ) [virtual]
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.
Reimplemented in Fl_Overlay_Window.

33.27.3.3 hide()

void Fl_Double_Window::hide ( ) [virtual]
Makes a widget invisible.
See also
    show(), visible(), visible_r()
Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.

33.27.3.4 resize()

void Fl_Double_Window::resize ( 
    int x,
    int y,
    int w,
    int h ) [virtual]
Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call `resize()` a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations. `position(X, Y)` is a shortcut for `resize(X, Y, w(), h())`, and `size(W, H)` is a shortcut for `resize(x(), y(), W, H)`.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x,y</th>
<th>new position relative to the parent window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w,h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`

Reimplemented from `Fl_Widget`.
Reimplemented in `Fl_Overlay_Window`.

#### 33.27.3.5 show()

```cpp
void Fl_Double_Window::show ( ) [virtual]
```

Makes a widget visible.

An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.

The `visible()` method returns true if the widget is set to be visible. The `visible_r()` method returns true if the widget and all of its parents are visible. A widget is only visible if `visible()` is true on it and all of its parents.

Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget. `redraw()` is called if necessary on this or the parent.

See also

- `hide()`, `visible()`, `visible_r()`

Reimplemented from `Fl_Widget`.
Reimplemented in `Fl_Overlay_Window`.

The documentation for this class was generated from the following files:

- `Fl_Double_Window.H`
- `Fl_Double_Window.cxx`

#### 33.28 Fl_End Class Reference

This is a dummy class that allows you to end a `Fl_Group` in a constructor list of a class:

```cpp
#include <Fl_Group.H>
```

### Public Member Functions

- `Fl_End ()`

  *All it does is calling `Fl_Group::current()->end()`*

#### 33.28.1 Detailed Description

This is a dummy class that allows you to end a `Fl_Group` in a constructor list of a class:

```cpp
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),
```
The documentation for this class was generated from the following file:

- Fl_Group.H

### 33.29 Fl_EPS_File_Surface Class Reference

Encapsulated PostScript drawing surface.

```c
#include <Fl_PostScript.H>
```

Inheritance diagram for Fl_EPS_File_Surface:

```
Fl_Surface_Device
   ↓
Fl_Widget_Surface
   ↓
Fl_EPS_File_Surface
```

**Public Member Functions**

- int **close ()**
  
  Completes all EPS output.

- FILE ∗ **file ()**
  
  Returns the underlying FILE pointer.

- Fl_EPS_File_Surface (int width, int height, FILE ∗ eps_output, Fl_Color background=FL_WHITE, Fl_PostScript_Close_Command closef=NULL)
  
  Constructor.

- void **origin (int ∗ px, int ∗ py)** FL_OVERRIDE
  
  Computes the coordinates of the current origin of graphics functions.

- void **origin (int x, int y)** FL_OVERRIDE
  
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- int **printable_rect (int ∗ w, int ∗ h)** FL_OVERRIDE
  
  Computes the width and height of the drawable area of the drawing surface.

- void **translate (int x, int y)** FL_OVERRIDE
  
  Translates the current graphics origin accounting for the current rotation.

- void **untranslate ()** FL_OVERRIDE
  
  Undoes the effect of a previous translate() call.

- ~Fl_EPS_File_Surface ()
  
  Destructor.

**Protected Member Functions**

- Fl_PostScript_Graphics_Driver ∗ **driver ()**
  
  Returns the PostScript driver of this drawing surface.

**Additional Inherited Members**

#### 33.29.1 Detailed Description

Encapsulated PostScript drawing surface.
This drawing surface allows to store any FLTK graphics in vectorial form in an "Encapsulated PostScript" file.

Usage example:
```c
Fl_Window *win = ...; // Window to draw to an .eps file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *eps = fl_fopen("/path/to/mywindow.eps", "w");
if (eps) {
    Fl_EPS_File_Surface *surface = new Fl_EPS_File_Surface(ww, wh, eps, win->color());
    Fl_Surface_Device::push_current(surface);
    surface->draw_decorated_window(win);
    Fl_Surface_Device::pop_current();
    delete surface; // the .eps file is not complete until the destructor was run
}
```

### 33.29.2 Constructor & Destructor Documentation

#### 33.29.2.1 Fl_EPS_File_Surface()

```c
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>width,height</th>
<th>Width and height of the EPS drawing area</th>
</tr>
</thead>
<tbody>
<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>

#### 33.29.2.2 ~Fl_EPS_File_Surface()

```c
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().

### 33.29.3 Member Function Documentation

#### 33.29.3.1 close()

```c
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.

33.29.3.2 origin() [1/2]

```
void Fl_EPS_File_Surface::origin (  
    int * x,  
    int * y) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

| out | x,y | If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.

33.29.3.3 origin() [2/2]

```
void Fl_EPS_File_Surface::origin (  
    int x,  
    int y) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. Origin() calls are not affected by `rotate()` calls (for classes derived from Fl_Paged_Device).

Parameters

| in | x,y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |

Reimplemented from Fl_Widget_Surface.

33.29.3.4 printable_rect()

```
int Fl_EPS_File_Surface::printable_rect (  
    int * w,  
    int * h) [virtual]
```

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

33.29.3.5 translate()

```
void Fl_EPS_File_Surface::translate (  
    int x,  
    int y) [virtual]
```

Generated by Doxygen
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`.

### 33.29.3.6 untranslate()

```cpp
template <typename T>
void Fl_EPS_File_Surface::untranslate (
    void ) [virtual]
```

Undoes the effect of a previous `translate()` call. Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following file:

- `Fl_PostScript.H`

### 33.30 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`:

```
Fl_Widget

Fl_Group

Fl_Browser_

Fl_Browser

Fl_File_Browser
```

**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

Generated by Doxygen
Sets or gets the size of the icons.

- void iconsize (uchar s)

Sets or gets the size of the icons.

- int load (const char *directory, Fl_File_Sort_F *sort=fl_numericsort)

Loads the specified directory into the browser.

- Fl_Fontsize textsize () const
- void textsize (Fl_Fontsize s)

Additional Inherited Members

33.30.1 Detailed Description

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.

33.30.2 Constructor & Destructor Documentation

33.30.2.1 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.

33.30.3 Member Function Documentation

33.30.3.1 errmsg [1/2]

const char * Fl_File_Browser::errmsg ( ) const [inline]

Returns OS error messages, or NULL if none. Use when advised.

33.30.3.2 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);

33.30.3.3 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.

33.30.3.4 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.
33.30.3.5 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.

33.30.3.6 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.

33.30.3.7 iconsize() [1/2]

uchar Fl_File_Browser::iconsize ( ) const [inline]
Sets or gets the size of the icons.
The default size is 20 pixels.

33.30.3.8 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.

33.30.3.9 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

33.31 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 (*)(Fl_File_Chooser *, void *)) (Fl_File_Chooser *, void *, void *d=0)
  Sets the file chooser callback cb and associated data d.
**Fl_Color color ()**
- Gets the background color of the Fl_File_Browser list.

**void color (Fl_Color c)**
- Sets the background color of the Fl_File_Browser list.

**int count ()**
- Returns the number of selected files.

**char *directory ()**
- Gets the current directory.

**void directory (const char *d)**
- Sets the current directory.

**const char *filter ()**
- Gets the current filename filter patterns.

**void filter (const char *p)**
- Sets the current filename filter patterns.

**int filter_value ()**
- Gets the current filename filter selection.

**void filter_value (int f)**
- Sets the current filename filter selection.

**Fl_File_Chooser (const char *d, const char *p, int t, const char *title)**
- The constructor creates the Fl_File_Chooser dialog shown.

**int h () const**
**void hide ()**
- Hides the Fl_File_Chooser window.

**uchar iconsize ()**
- Gets the size of the icons in the Fl_File_Browser.

**void iconsize (uchar s)**
- Sets the size of the icons in the Fl_File_Browser.

**const char *label ()**
- Gets the title bar text for the Fl_File_Chooser.

**void label (const char *l)**
- Sets the title bar text for the Fl_File_Chooser.

**const char *ok_label ()**
- Gets the label for the "ok" button in the Fl_File_Chooser.

**void ok_label (const char *l)**
- Sets the label for the "ok" button in the Fl_File_Chooser.

**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 Fl_File_Browser.

**void rescan_keep_filename ()**
- Rescan the current directory without clearing the filename, then select the file if it is in the list.

**void resize (int x, int y, int w, int h)**

**void show ()**
- Shows the Fl_File_Chooser window.

**int shown ()**
- Returns non-zero if the file chooser main window show() has been called, but not hide().

**void size (int w, int h)**

**Fl_Color textcolor ()**
-
Gets the current `Fl_File_Browser` text color.

• void textcolor (Fl_Color c)
  
  Sets the current `Fl_File_Browser` text color.

• Fl_Font textfont ()
  
  Gets the current `Fl_File_Browser` text font.

• void textfont (Fl_Font f)
  
  Sets the current `Fl_File_Browser` text font.

• Fl_Fontsize textsize ()
  
  Gets the current `Fl_File_Browser` text size.

• void textsize (Fl_Fontsize s)
  
  Sets the current `Fl_File_Browser` text size.

• int type ()
  
  Gets the current type of `Fl_File_Chooser`.

• void type (int t)
  
  Sets the current type of `Fl_File_Chooser`.

• void * user_data () const
  
  Gets the file chooser user data.

• void user_data (void *d)
  
  Sets the file chooser user data d.

• void value (const char *filename)
  
  Sets the current value of the selected file.

• const char * value (int f=1)
  
  Gets the current value of the selected file(s).

• int visible ()
  
  Returns 1 if the `Fl_File_Chooser` window is visible.

• int w () const
• int x () const
• int y () const

• ~Fl_File_Chooser ()
  
  Destroys the widget and frees all memory used by it.

Public Attributes

• Fl_Button * newButton
  
  The "new directory" button is exported so that application developers can control the appearance and use.

• Fl_Check_Button * previewButton
  
  The "preview" button is exported so that application developers can control the appearance and use.

• Fl_Check_Button * showHiddenButton
  
  When checked, hidden files (i.e., filename begins with dot) are displayed.

Static Public Attributes

• static const char * add_favorites_label = "Add to Favorites"
  
  [standard text may be customized at run-time]

• static const char * all_files_label = "All Files (*)"
  
  [standard text may be customized at run-time]

• static const char * custom_filter_label = "Custom Filter"
  
  [standard text may be customized at run-time]

• static const char * existing_file_label = "Please choose an existing file!"
  
  [standard text may be customized at run-time]

• static const char * favorites_label = "Favorites"
  
  [standard text may be customized at run-time]
• static const char * filename_label = "Filename:"
  [standard text may be customized at run-time]
• static const char * filesystems_label = Fl::system_driver()->filesystems_label()
  [standard text may be customized at run-time]
• static const char * hidden_label = "Show hidden files"
  [standard text may be customized at run-time]
• static const char * manage_favorites_label = "Manage Favorites"
  [standard text may be customized at run-time]
• static const char * new_directory_label = "New Directory?"
  [standard text may be customized at run-time]
• static const char * new_directory_tooltip = "Create a new directory."
  [standard text may be customized at run-time]
• static const char * preview_label = "Preview"
  [standard text may be customized at run-time]
• static const char * save_label = "Save"
  [standard text may be customized at run-time]
• static const char * show_label = "Show:"
  [standard text may be customized at run-time]
• static Fl_File_Sort_F * sort = fl_numericsort
  the sort function that is used when loading the contents of a directory.

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.

33.31.1 Detailed Description

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.
Features include:

- Multiple filter patterns can be specified, with parenthesis around filters, and tabs to separate each pattern, e.g.:
  ```
  char pattern[] = "Image Files (*.bmp,gif,jpg,png,xbm,xpm)|t"
  "Web Files (*.htm,html,php)|t"
  "All Files (*)|t";
  ```

- If no "*" pattern is provided, then an entry for "All Files (*)" is automatically added.
- An optional file preview box is provided which can be toggled by programmer or user showing images, or the first 2048 bytes of printable text.
- Preview image loading functions can be registered to provide custom file previews.
- The favorites button shows up to 100 user-saved favorite directories, the user's home directory, and a filesystems item.
- A simple dialog is provided for managing saved directories.
- Shortcut keys are provided:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+a</td>
<td>Adds a directory to the favorites list</td>
</tr>
<tr>
<td>Alt+m</td>
<td>Manages the favorites list</td>
</tr>
<tr>
<td>Alt+f</td>
<td>Shows the filesystem list</td>
</tr>
<tr>
<td>Alt+h</td>
<td>Go to the home directory</td>
</tr>
<tr>
<td>Alt+0..9</td>
<td>going to any of the first 10 favorites</td>
</tr>
</tbody>
</table>

The Fl_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)</td>
</tr>
<tr>
<td></td>
<td>&quot;File Systems&quot; (all others)</td>
</tr>
<tr>
<td>hidden_label</td>
<td>&quot;Show hidden files:&quot;</td>
</tr>
<tr>
<td>manageFavorites_label</td>
<td>&quot;Manage Favorites&quot;</td>
</tr>
<tr>
<td>new_directory_label</td>
<td>&quot;New Directory?&quot;</td>
</tr>
<tr>
<td>new_directory_tooltip</td>
<td>&quot;Create a new directory.&quot;</td>
</tr>
<tr>
<td>preview_label</td>
<td>&quot;Preview&quot;</td>
</tr>
<tr>
<td>save_label</td>
<td>&quot;Save&quot;</td>
</tr>
<tr>
<td>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.

### 33.31.2 Constructor & Destructor Documentation

#### 33.31.2.1 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 can be a directory name or a complete file name (in which case the corresponding file is highlighted in the list and in the filename input field.)

The pattern argument can be a NULL string or "*" to list all files, or it can be a series of descriptions and filter strings separated by tab characters (\t). The format of filters is either "Description text (patterns)" or just "patterns". A file chooser that provides filters for HTML and image files might look like:

"HTML Files (*.html)\tImage Files (*.bmp,gif,jpg,png)"

The file chooser will automatically add the "All Files (*)" pattern to the end of the string you pass if you do not provide one. The first filter in the string is the default filter.

See the FLTK documentation on fl_filename_match() for the kinds of pattern strings that are supported.

The type argument can be one of the following:

- **SINGLE** - allows the user to select a single, existing file.
- **MULTI** - allows the user to select one or more existing files.
• CREATE - allows the user to select a single, existing file or specify a new filename.
• DIRECTORY - allows the user to select a single, existing directory.

The title argument is used to set the title bar text for the Fl_File_Chooser window.

33.31.3 Member Function Documentation

33.31.3.1 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.

33.31.3.2 filter()

void Fl_File_Chooser::filter ( const char * p )

Sets the current filename filter patterns.
The filter patterns use fl_filename_match(). Multiple patterns can be used by separating them with tabs, like
"*.jpg	*.png	*.gif	*". In addition, you can provide human-readable labels with the patterns inside parenthesis, like
"JPEG Files (*.jpg)	PNG Files (*.png)	GIF Files (*.gif)	All Files (*)".
Use filter(NULL) to show all files.

33.31.3.3 iconsize() [1/2]

uchar Fl_File_Chooser::iconsize ( )

Gets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the textsize().
33.31.3.4 iconsize() [2/2]

void Fl_File_Chooser::iconsize (uchar s)

Sets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the fontsize().

33.31.3.5 preview()

void Fl_File_Chooser::preview (int e)

Enable or disable the preview tile.
1 = enable preview, 0 = disable preview.

33.31.3.6 shown()

int Fl_File_Chooser::shown ()

Returns non-zero if the file chooser main window show() has been called, but not hide().

See also

Fl_Window::shown()

33.31.3.7 value()

const char * Fl_File_Chooser::value (int f = 1)

Gets the current value of the selected file(s).
f is a 1-based index into a list of file names. The number of selected files is returned by Fl_File_Chooser::count().

This sample code loops through all selected files:

```c
// Get list of filenames user selected from a MULTI chooser
for ( int t=1; t< chooser->count(); t++ ) {
  const char *filename = chooser->value(t);
  ...
}
```

33.31.4 Member Data Documentation

33.31.4.1 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

33.32 Fl_File_Icon Class Reference

The Fl_File_Icon class manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget.
#include <Fl_File_Icon.H>
Public Types

- enum {
  ANY, PLAIN, FIFO, DEVICE,
  LINK, DIRECTORY
}
- enum {
  END, COLOR, LINE, CLOSEDLINE,
  POLYGON, OUTLINEPOLYGON, VERTEX
}

Public Member Functions

- short * add (short d)
  Adds a keyword value to the icon array, returning a pointer to it.
- short * add_color (Fl_Color c)
  Adds a color value to the icon array, returning a pointer to it.
- short * add_vertex (float x, float y)
  Adds a vertex value to the icon array, returning a pointer to it.
- short * add_vertex (int x, int y)
  Adds a vertex value to the icon array, returning a pointer to it.
- void clear ()
  Clears all icon data from the icon.
- void draw (int x, int y, int w, int h, Fl_Color ic, int active=1)
  Draws an icon in the indicated area.
- Fl_File_Icon (const char *p, int t, int nd=0, short *d=0)
  Creates a new Fl_File_Icon with the specified information.
- void label (Fl_Widget *w)
  Applies the icon to the widget, registering the Fl_File_Icon label type as needed.
- void load (const char *f)
  Loads the specified icon image.
- int load_fti (const char *fti)
  Loads an SGI icon file.
- int load_image (const char *i)
  Load an image icon file from an image filename.
- Fl_File_Icon * next ()
  Returns next file icon object.
- const char * pattern ()
  Returns the filename matching pattern for the icon.
- int size ()
  Returns the number of words of data used by the icon.
- int type ()
  Returns the filetype associated with the icon, which can be one of the following:
- short * value ()
  Returns the data array for the icon.
- ~Fl_File_Icon ()
  The destructor destroys the icon and frees all memory that has been allocated for it.

Static Public Member Functions

- static Fl_File_Icon * find (const char *filename, int filetype=ANY)
  Finds an icon that matches the given filename and file type.
- static Fl_File_Icon * first ()
  Returns a pointer to the first icon in the list.
- static void labeltype (const Fl_Label *o, int x, int y, int w, int h, Fl_Align a)
Draw the icon label.

- static void load_system_icons (void)
  Loads all system-defined icons.

### 33.32.1 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.

### 33.32.2 Constructor & Destructor Documentation

#### 33.32.2.1 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>in</th>
<th>p</th>
<th>filename pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
<td>file type</td>
</tr>
<tr>
<td>in</td>
<td>nd</td>
<td>number of data values</td>
</tr>
<tr>
<td>in</td>
<td>d</td>
<td>data values</td>
</tr>
</tbody>
</table>

### 33.32.3 Member Function Documentation

#### 33.32.3.1 add()

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 |

#### 33.32.3.2 add_color()

short * Fl_File_Icon::add_color (  
  Fl_Color c ) [inline]

Adds a color value to the icon array, returning a pointer to it.

**Parameters**

| in | c | color value |
### 33.32.3.3 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 |

### 33.32.3.4 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 |

### 33.32.3.5 draw()

```cpp
void Fl_File_Icon::draw(int x, int y, int w, int h, Fl_Color ic, int active = 1)
```

Draws an icon in the indicated area.

**Parameters**

| in | x, y, w, h | position and size |
| in | ic | icon color |
| in | active | status, default is active [non-zero] |

### 33.32.3.6 find()

```cpp
Fl_File_Icon * Fl_File_Icon::find(const char *filename, int filetype = ANY) [static]
```

Finds an icon that matches the given filename and file type.

**Parameters**

| in | filename | name of file |
| in | filetype | enumerated file type |
Returns

matching file icon or NULL

### 33.32.3.7 label()

```cpp
void Fl_File_Icon::label ( Fl_Widget * w )
```

Applies the icon to the widget, registering the Fl_File_Icon label type as needed.

**Parameters**

- `w`: widget for which this icon will become the label

### 33.32.3.8 labeltype()

```cpp
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**

- `o`: label data
- `x, y, w, h`: position and size of label
- `a`: label alignment [not used]

### 33.32.3.9 load()

```cpp
void Fl_File_Icon::load ( const char * f )
```

Loads the specified icon image.
The format is deduced from the filename.

**Parameters**

- `f`: filename

### 33.32.3.10 load_fti()

```cpp
int Fl_File_Icon::load_fti ( const char * fti )
```

Loads an SGI icon file.

**Parameters**

- `fti`: icon filename
33.32 Fl_File_Icon Class Reference

Returns
0 on success, non-zero on error

33.32.3.11 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

33.32.3.12 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();`

33.32.3.13 next()

```cpp
Fl_File_Icon * Fl_File_Icon::next ( ) [inline]
```

Returns next file icon object.
See `Fl_File_Icon::first()`

33.32.3.14 type()

```cpp
int Fl_File_Icon::type ( ) [inline]
```

Returns the filetype associated with the icon, which can be one of the following:

- `Fl_File_Icon::ANY`, any kind of file.
- `Fl_File_Icon::PLAIN`, plain files.
- `Fl_File_Icon::FIFO`, named pipes.
- `Fl_File_Icon::DEVICE`, character and block devices.
- `Fl_File_Icon::LINK`, symbolic links.
- `Fl_File_Icon::DIRECTORY`, directories.

The documentation for this class was generated from the following files:

- `Fl_File_Icon.H`
- `Fl_File_Icon.cxx`
- `Fl_File_Icon2.cxx`
33.33  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.
- **int** `handle (int event) FL_OVERRIDE`
  
  Handle events in the widget.
- **const char ∗** `value ()`
  
  Returns the current value, which is a pointer to an internal buffer and is valid only until the next event is handled.
- **int** `value (const char ∗str)`
  
  Sets the value of the widget given a new string value.
- **int** `value (const char ∗str, int len)`
  
  Sets the value of the widget given a new string value and its length.

Protected Member Functions

- **void** `draw () FL_OVERRIDE`
  
  Draws the file input widget.

Additional Inherited Members

33.33.1  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.
33.33 Fl_File_Input Class Reference

![Fl_File_Input](image)

Figure 33.15 Fl_File_Input

**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.

### 33.33.2 Constructor & Destructor Documentation

#### 33.33.2.1 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>

### 33.33.3 Member Function Documentation

#### 33.33.3.1 down_box()

`void Fl_File_Input::down_box (Fl_Boxtype b)` [inline]

Sets the box type to use for the navigation bar.

#### 33.33.3.2 draw()

`void Fl_File_Input::draw (void)` [protected], [virtual]

Draws the file input widget.

Implements Fl_Widget.

#### 33.33.3.3 errorcolor() [1/2]

`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.

Generated by Doxygen
Todo  Remove Fl_File_Input::errorcolor() in FLTK 1.5.0 or higher.

33.33.3.4 errorcolor() [2/2]
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.

33.33.3.5 handle()
int Fl_File_Input::handle ( int event ) [virtual]
Handle events in the widget.
Return non zero if event is handled.

Parameters

in  event

Reimplemented from Fl_Widget.

33.33.3.6 value() [1/2]
int Fl_File_Input::value ( const char * str )
Sets the value of the widget given a new string value.
Returns non 0 on success.

Parameters

in  str  new string value

33.33.3.7 value() [2/2]
int Fl_File_Input::value ( const char * str, int len )
Sets the value of the widget given a new string value and its length.
Returns non 0 on success.

Parameters

in  str  new string value

in  len  length of value

The documentation for this class was generated from the following files:
  • Fl_File_Input.H
**33.34 Fl_Fill_Dial Class Reference**

Draws a dial with a filled arc.

#include <Fl_Fill_Dial.H>

Inheritance diagram for Fl_Fill_Dial:

```
        Fl_Widget
       /     \
      Fl_Valuator
     /       \
    Fl_Dial   Fl_Fill_Dial
```

Public Member Functions

- **Fl_Fill_Dial** (int X, int Y, int W, int H, const char *L)
  
  Creates a filled dial, also setting its type to FL_FILL_DIAL.

Additional Inherited Members

**33.34.1 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

**33.35 Fl_Fill_Slider Class Reference**

Widget that draws a filled horizontal slider, useful as a progress or value meter.

#include <Fl_Fill_Slider.H>

Inheritance diagram for Fl_Fill_Slider:

```
        Fl_Widget
       /     \
      Fl_Valuator
     /       \
    Fl_Slider   Fl_Fill_Slider
```

Public Member Functions

- **Fl_Fill_Slider** (int X, int Y, int W, int H, const char *L=0)
  
  Creates the slider from its position, size and optional title.
Additional Inherited Members

33.35.1 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

33.36 Fl_Flex Class Reference

Fl_Flex is a container (layout) widget for one row or one column of widgets.
#include <Fl_Flex.H>
Inheritance diagram for Fl_Flex:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Flex
```

Public Types

- enum { VERTICAL = 0, HORIZONTAL = 1, COLUMN = 0, ROW = 1 }

Public Member Functions

- virtual void end ()
  Ends automatic child addition and resizes all children.
- void fixed (Fl_Widget &w, int size)
  Set the horizontal or vertical size of a child widget.
- int fixed (Fl_Widget *w) const
  Return whether the given widget has a fixed size or resizes dynamically.
- void fixed (Fl_Widget *w, int size)
  Set the horizontal or vertical size of a child widget.
- Fl_Flex (int direction)
  Construct a new Fl_Flex widget specifying its layout.
- Fl_Flex (int w, int h, int direction)
  Construct a new Fl_Flex widget specifying its layout and size.
- Fl_Flex (int X, int Y, int W, int H, const char *L=0)
  Construct a new Fl_Flex widget with the given position, size, and label.
- Fl_Flex (int x, int y, int w, int h, int direction)
  Construct a new Fl_Flex widget specifying its layout, position, and size.
- int gap () const
  Return the gap size of the widget.
- void gap (int g)
  Set the gap size of the widget.
- int horizontal () const
  Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).
- void layout ()
  Calculates the layout of the widget and redraws it.
• int margin () const
  Returns the left margin size of the widget.
• int margin (int *left, int *top, int *right, int *bottom) const
  Returns all (four) margin sizes of the widget.
• void margin (int left, int top, int right, int bottom)
  Set the margin sizes at all four edges of the Fl_Flex widget.
• void margin (int m, int g=-1)
  Set the margin and optionally the gap size of the widget.
• void resize (int x, int y, int w, int h) FL_OVERRIDE
  Changes the size or position of the 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)
• void on_remove (int) FL_OVERRIDE
  Allow derived groups to act when a child widget is removed from the group.

Additional Inherited Members

33.36.1 Detailed Description

Fl_Flex is a container (layout) widget for one row or one column of widgets. It provides flexible positioning of its children either in one row or in one column. Fl_Flex is designed to be as simple as possible. You can set individual widget sizes or let Fl_Flex position and size the widgets to fit in the container. All “flexible” (i.e. non-fixed size) widgets are assigned the same width or height, respectively. For details see below.

You can set the margins around all children at the inner side of the box frame (if any). Fl_Flex supports setting different margin sizes on top, bottom, left, and right sides. The default margin size is 0 on all edges of the container. You can set the gap size between all children. The gap size is always the same between all of its children. This is similar to the ‘spacing’ of Fl_Pack. The default gap size is 0.

Fl_Flex can either consist of a single row, i.e. type(Fl_Flex::HORIZONTAL) or a single column, i.e. type(Fl_Flex::VERTICAL). The default value is Fl_Flex::VERTICAL for consistency with Fl_Pack but you can use type() to assign a row (Fl_Flex::HORIZONTAL) layout.

If type() == Fl_Flex::HORIZONTAL widgets are resized horizontally to fit in the container and their height is the full Fl_Flex height minus border size and margins. You can set a fixed widget width by using fixed().

If type() == Fl_Flex::VERTICAL widgets are resized vertically to fit in the container and their width is the full Fl_Flex width minus border size and margins. You can set a fixed widget height by using fixed().

To create arbitrary spacing you can use invisible boxes of flexible or fixed sizes (see example below). Alternate constructors let you specify the layout as Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL directly. Fl_Flex::ROW is an alias of Fl_Flex::HORIZONTAL and Fl_Flex::COLUMN is an alias of Fl_Flex::VERTICAL.

The default box type is FL_NO_BOX as inherited from Fl_Group. You may need to set a box type with a solid background depending on your layout.

Important: You should always make sure that the Fl_Flex container cannot be resized smaller than its designed minimal size. This can usually be done by setting a size_range() on the window as shown in the example below. Fl_Flex does not take care of sensible sizes. If it is resized too small the behavior is undefined, i.e. widgets may overlap and/or shrink to zero size.

Hint: In many cases Fl_Flex can be used as a drop-in replacement for Fl_Pack. This is the recommended single row/column container since FLTK 1.4.0. Its resizing behavior is much more predictable (as expected) than that of Fl_Pack which "resizes itself to shrink-wrap itself around all of the children".
**FL_Flex** containers can be nested so you can create flexible layouts with multiple columns and rows. However, if your UI design is more complex you may want to use FL_Grid instead.

**Example:**

```c
#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_Button b3(0, 0, 0, 0, "Exit");
    flex.fixed(bx, 60); // set fix width of invisible box
    flex.gap(10);
    flex.end();
    window.resizable(flex);
    window.end();
    window.size_range(300, 30);
    window.show(argc, argv);
    return Fl::run();
}
```

Since **1.4.0**

### 33.36.2 Member Enumeration Documentation

#### 33.36.2.1 anonymous enum

**anonymous enum**

**Enumerator**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VERTICAL</td>
<td>vertical layout (one column)</td>
</tr>
<tr>
<td>HORIZONTAL</td>
<td>horizontal layout (one row)</td>
</tr>
<tr>
<td>COLUMN</td>
<td>alias for VERTICAL</td>
</tr>
<tr>
<td>ROW</td>
<td>alias for HORIZONTAL</td>
</tr>
</tbody>
</table>

### 33.36.3 Constructor & Destructor Documentation

#### 33.36.3.1 FL_Flex() [1/4]

**FL_Flex::FL_Flex**

```c
    int X, Y, W, 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)`. 

---

Generated by Doxygen
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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>X,Y</code></td>
<td>int</td>
<td>position</td>
</tr>
<tr>
<td><code>W,H</code></td>
<td>int</td>
<td>size (width and height)</td>
</tr>
<tr>
<td><code>L</code></td>
<td></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)`

33.36.3.2 Fl_Flex() [2/4]

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>direction</code></td>
<td>int</td>
<td>horizontal (row) or vertical (column) layout</td>
</tr>
</tbody>
</table>

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)`

33.36.3.3 Fl_Flex() [3/4]

```cpp
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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w,h</code></td>
<td>int</td>
<td>widget size</td>
</tr>
<tr>
<td><code>direction</code></td>
<td>int</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)

33.36.3.4 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

| in  | x, y | widget position |
| in  | w, h | widget size     |
| in  | direction | horizontal (row) or vertical (column) layout |

See also

Fl_Flex::Fl_Flex(int direction)
Fl_Flex::Fl_Flex(int w, int h, int direction)
Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

33.36.4 Member Function Documentation

33.36.4.1 alloc_size()

int Fl_Flex::alloc_size {
    int size) const [protected], [virtual]

Return new size to be allocated for array of fixed size widgets.
This method is called when the array of fixed size widgets needs to be expanded. The current size is provided (size can be 0). The default method adds 8 to the current size.
This can be used in derived classes to change the allocation strategy. Note that this method only queries the new size which shall be allocated but does not allocate the memory.

Parameters

| in  | size | current size |

Returns

int new size (to be allocated)
33.36.4.2 end()

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.

33.36.4.3 fixed() [1/3]

void Fl_Flex::fixed ( Fl_Widget & w, int size ) [inline]

Set the horizontal or vertical size of a child widget.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>w</th>
<th>widget to be affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)</td>
</tr>
</tbody>
</table>

See also

fixed(Fl_Widget & w, int size)

33.36.4.4 fixed() [2/3]

int Fl_Flex::fixed ( Fl_Widget * w ) const

Return whether the given widget has a fixed size or resizes dynamically.

Parameters

| in  | w         | widget |

Returns

whether the widget has a fixed size

Return values

| 1  | the widget has a fixed size |
| 0  | the widget resizes dynamically |

33.36.4.5 fixed() [3/3]

void Fl_Flex::fixed ( Fl_Widget * child, int size )

Set the horizontal or vertical size of a child widget.
This sets either the width or height of a child widget, depending on the type() of the Fl_Flex container (Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL). The other dimension is set to the full width or height of the Fl_Flex widget minus margin sizes.
This can be used to set a fixed widget width or height of children of Fl_Flex so they are not resized dynamically. If size is 0 (zero) or negative the widget size is reset to flexible size.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>child</th>
<th>widget to be affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)</td>
</tr>
</tbody>
</table>

### 33.36.4.6 gap [1/2]

```cpp
int Fl_Flex::gap ( ) const [inline]
```

Return the gap size of the widget.

Returns

- gap size between all child widgets.

### 33.36.4.7 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**

- **g** gap size

### 33.36.4.8 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**

- \( 1 \) if `type() == Fl_Flex::HORIZONTAL`
- \( 0 \) if `type() == Fl_Flex::VERTICAL`

See class Fl_Flex documentation for details.

### 33.36.4.9 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.

### 33.36.4.10 margin [1/4]

```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 margin(int *left, int *top, int *right, int *bottom) to get all four margin values.

Returns

size of left margin.

33.36.4.11 margin() [2/4]

int Fl_Flex::margin (  
    int * left,  
    int * top,  
    int * right,  
    int * bottom ) const [inline]

Returns all (four) margin sizes of the widget. All margin sizes are returned in the given arguments. If any argument is NULL the respective value is not returned.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>left</th>
<th>returns left margin if not NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>top</td>
<td>returns top margin if not NULL</td>
</tr>
<tr>
<td>in</td>
<td>right</td>
<td>returns right margin if not NULL</td>
</tr>
<tr>
<td>in</td>
<td>bottom</td>
<td>returns bottom margin if not NULL</td>
</tr>
</tbody>
</table>

Returns

whether all margins are equal

Return values

| 1 | all margins have the same size |
| 0 | at least one margin has a different size |

33.36.4.12 margin() [3/4]

void Fl_Flex::margin (  
    int left,  
    int top,  
    int right,  
    int bottom ) [inline]

Set the margin sizes at all four edges of the Fl_Flex widget. The margin is the free space inside the widget border around all child widgets. You must use all four parameters of this method to set the four margins in the order left, top, right, bottom. Negative values are set to 0 (zero). To set all margins to equal sizes, use margin(int m) which sets all four margins to the same size.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m</code></td>
<td>Margin size, must be $\geq 0$</td>
</tr>
<tr>
<td><code>g</code></td>
<td>Gap size (ignored, if negative)</td>
</tr>
</tbody>
</table>

### See also

- `margin(int, int)`
- `gap(int)`

#### 33.36.4.13 margin()

```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 the free space inside the widget border around all child widgets. This method sets the margin to the same size at all four edges of the Fl_Flex widget. The gap size `g` is the free space between child widgets. Negative values do not change the gap value. This is the default if this argument is omitted.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m</code></td>
<td>Margin size, must be $\geq 0$</td>
</tr>
<tr>
<td><code>g</code></td>
<td>Gap size (ignored, if negative)</td>
</tr>
</tbody>
</table>

### See also

- `gap(int)`

#### 33.36.4.14 on_remove()

```cpp
void Fl_Flex::on_remove (  
    int index ) [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group. Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to remove these data structures just before the child is removed.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>Remove the child at this position in the array</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Group.

#### 33.36.4.15 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>in</th>
<th>x,y</th>
<th>new position relative to the parent window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w,h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`

Reimplemented from `Fl_Widget`.

#### 33.36.4.16 spacing() [1/2]

```cpp
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()`

#### 33.36.4.17 spacing() [2/2]

```cpp
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_Float_Input.H`
- `Fl_Float.cxx`

#### 33.37 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).

```cpp
#include <Fl_Float_Input.H>
```

Inheritance diagram for `Fl_Float_Input`:  

Generated by Doxygen
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

33.37.1 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).

33.37.2 Constructor & Destructor Documentation

33.37.2.1 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

33.38 Fl FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.

#include <Fl FormsBitmap.H>

Inheritance diagram for Fl FormsBitmap:
Public Member Functions

- **Fl_Bitmap * bitmap () const**
  
  *Gets a the current associated Fl_Bitmap objects.*

- **void bitmap (Fl_Bitmap *B)**
  
  *Sets a new bitmap.*

- **Fl_FormsBitmap (Fl_Boxtype, int, int, int, int, const char *=0)**
  
  *Creates a bitmap widget from a box type, position, size and optional label specification.*

- **void set (int W, int H, const uchar *bits)**
  
  *Sets a new bitmap bits with size W,H.*

Protected Member Functions

- **void draw () FL_OVERRIDE**
  
  *Draws the bitmap and its associated box.*

Additional Inherited Members

33.38.1 Detailed Description

Forms compatibility Bitmap Image Widget.

33.38.2 Member Function Documentation

33.38.2.1 draw()

```cpp
void Fl_FormsBitmap::draw ( ) [protected], [virtual]
```

*Draws the bitmap and its associated box.*

Implements **Fl_Widget**.

33.38.2.2 set()

```cpp
void Fl_FormsBitmap::set ( int W, int H, const uchar *bits )
```

*Sets a new bitmap bits with size W,H.*

*Deletes the previous one.*

*The documentation for this class was generated from the following files:*

- Fl_FormsBitmap.H
- forms_bitmap.cxx

33.39 Fl_FormsPixmap Class Reference

Forms pixmap drawing routines.

#include <Fl_FormsPixmap.H>

Inheritance diagram for 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 () FL_OVERRIDE**

  Draws the widget.

Additional Inherited Members

33.39.1 Detailed Description

Forms pixmap drawing routines.

33.39.2 Constructor & Destructor Documentation

33.39.2.1 Fl_FormsPixmap()

Fl_FormsPixmap::Fl_FormsPixmap (  
  Fl_Boxtype t,  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char */L = 0 )

Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>position and size</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>widget label, default is no label</td>
</tr>
</tbody>
</table>

33.39.3 Member Function Documentation

33.39.3.1 draw()

void Fl_FormsPixmap::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
33.40 FL_FormsText Class Reference

```
FL_Widget *s = scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements FL_Widget.

### 33.39.3.2 Pixmap()

```
void Fl_FormsPixmap::Pixmap ( Fl_Pixmap * B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

**Parameters**

| in | B | existing pixmap |

### 33.39.3.3 set()

```
void Fl_FormsPixmap::set ( char *const * bits )
```

Set/create the internal pixmap using raw data.

**Parameters**

| in | bits | raw data |

The documentation for this class was generated from the following files:

- Fl_FormsPixmap.H
- formsPixmap.cxx

### 33.40 FL_FormsText Class Reference

Inheritance diagram for FL_FormsText:

```
Fl_Widget

|<| Fl_FormsText |
```

**Public Member Functions**

- FL_FORMSTEXT (Fl_Boxtype b, int X, int Y, int W, int H, const char *l=0)

**Protected Member Functions**

- void draw () FL_OVERRIDE

  Draws the widget.

**Additional Inherited Members**

### 33.40.1 Member Function Documentation

Generated by Doxygen
33.40.1.1 draw()

void Fl_FormsText::draw ( ) [protected], [virtual]

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```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

33.41 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.

- **int handle (int e) FL_OVERRIDE**

  * Handles the specified event.

- **~Fl_Free ()**

  * The destructor will call the handle function with the event FL_FREE_MEM.

**Protected Member Functions**

- **void draw () FL_OVERRIDE**

  * Draws the widget.

**Additional Inherited Members**

33.41.1 Detailed Description

Emulation of the Forms "free" widget.

This emulation allows the free demo to run, and appears to be useful for porting programs written in Forms which use the free widget or make subclasses of the Forms widgets.

There are five types of free, which determine when the handle function is called:

- **FL_NORMAL_FREE** normal event handling.
- **FL_SLEEPING_FREE** deactivates event handling (widget is inactive).
- **FL_INPUT_FREE** accepts FL_FOCUS events.
- **FL_CONTINUOUS_FREE** sets a timeout callback 100 times a second and provides an FL_STEP event. This has obvious detrimental effects on machine performance.
- **FL_ALL_FREE** same as FL_INPUT_FREE and FL_CONTINUOUS_FREE.
33.41 Fl_Free Class Reference

**33.41.2 Constructor & Destructor Documentation**

**33.41.2.1 Fl_Free()**

```cpp
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**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>t</td>
</tr>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
</tr>
<tr>
<td>in</td>
<td>hdl</td>
</tr>
</tbody>
</table>

The constructor takes both the type and the handle function. The handle function should be declared as follows:

```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.

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

**33.41.3 Member Function Documentation**

**33.41.3.1 draw()**

```cpp
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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

**33.41.3.2 handle()**

```cpp
int Fl_Free::handle ( int event ) [virtual]
```

Handles the specified event.

You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in | event | the kind of event received |

Return values

| 0  | if the event was not used or understood |
| 1  | if the event was used and can be deleted |

See also

`Fl_Event`

Reimplemented from `Fl_Widget`. The documentation for this class was generated from the following files:

- `Fl_Free.H`
- `forms_free.cxx`

### Fl_GIF_Image Class Reference

The `Fl_GIF_Image` class supports loading, caching, and drawing of Compuserve GIF images.

```cpp
#include <Fl_GIF_Image.H>
```

Inheritance diagram for `Fl_GIF_Image`:

```
Fl_GIF_Image
  \|-- Fl_Pixmap
      \|-- Fl_GIF_Frame
          \|-- Fl_Anim_GIF_Frame
```

#### Classes

- `struct GIF_FRAME`

#### Public Member Functions

- `Fl_GIF_Image` (const char ∗`filename`)
  
  This constructor loads a GIF image from the given file.

- `Fl_GIF_Image` (const char ∗`imagename`, const unsigned char ∗`data`)
  
  This constructor loads a GIF image from memory (deprecated).

- `Fl_GIF_Image` (const char ∗`imagename`, const unsigned char ∗`data`, const size_t `length`)
  
  This constructor loads a GIF image from memory.

#### Static Public Member Functions

- static bool `is_animated` (const char ∗`name_`)

Generated by Doxygen
Static Public Attributes

- static bool animate = false
  
  Sets how the shared image core routine should treat animated GIF files.

Protected Member Functions

- Fl_GIF_Image()
  
  The default constructor creates an empty GIF image.
- Fl_GIF_Image(const char *filename, bool anim)
- Fl_GIF_Image(const char *imagename, const unsigned char *data, const size_t length, bool anim)
- void load(const char *filename, bool anim)
  
  The protected load() methods are used by Fl_Anim_GIF_Image to request loading of animated GIF's.
- void load(const char *imagename, const unsigned char *data, const size_t length, bool anim)
- void load_gif_(class Fl_Image_Reader &rdr, bool anim=false)
- virtual void on_extension_data(GIF_FRAME &)
- virtual void on_frame_data(GIF_FRAME &)

Additional Inherited Members

33.42.1 Detailed Description

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF\textsuperscript{SM} images. The class loads the first image and supports transparency.

33.42.2 Constructor & Destructor Documentation

33.42.2.1 Fl_GIF_Image() [1/3]

Fl_GIF_Image::Fl_GIF_Image {
  const char * filename }

This constructor loads a GIF image from the given file.
If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

- **in filename** a full path and name pointing to a GIF image file.

See also

- Fl_GIF_Image::Fl_GIF_Image(const char *imagename, const unsigned char *data, const long length)

33.42.2.2 Fl_GIF_Image() [2/3]

Fl_GIF_Image::Fl_GIF_Image {
  const char * imagename,
  const unsigned char * data }

This constructor loads a GIF image from memory (deprecated).

**Deprecated** Please use Fl_GIF_Image(const char *imagename, const unsigned char *data, const size_t length) instead.
Note

Buffer overruns will not be checked.

This constructor should not be used because the caller can't supply the memory size and the image reader can't check for "end of memory" errors.

Note

A new constructor with parameter length is available since FLTK 1.4.0.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>imagename</th>
<th>A name given to this image or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>Pointer to the start of the GIF image in memory.</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)

33.42.3 Fl_GIF_Image() [3/3]

Fl_GIF_Image::Fl_GIF_Image (const char * imagename,
const unsigned char * data,
const size_t length )

This constructor loads a GIF image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. imagename can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation. The destructor frees all memory and server resources that are used by the image. The third parameter length is used to test for buffer overruns, i.e. truncated images. Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>imagedata</th>
<th>A name given to this image or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>Pointer to the start of the GIF image in memory.</td>
</tr>
<tr>
<td>in</td>
<td>length</td>
<td>Length of the GIF image in memory.</td>
</tr>
</tbody>
</table>

See also

- Fl_GIF_Image::Fl_GIF_Image(const char *filename)
- Fl_Shared_Image

Since

Since 1.4.0

33.42.3 Member Data Documentation
33.42.3.1 animate

```cpp
bool Fl_GIF_Image::animate = false [static]
```

Sets how the shared image core routine should treat animated GIF files. The default is to treat them as ordinary GIF's e.g. it creates a `Fl_GIF_Image` object. If this variable is set, then an animated GIF object `Fl_Anim_GIF_Image` is created.

The documentation for this class was generated from the following files:

- Fl_GIF_Image.H
- Fl_Anim_GIF_Image.cxx
- Fl_GIF_Image.cxx

33.43 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

33.44 Fl_Gl_Window Class Reference

The Fl_Gl_Window widget sets things up so OpenGL works.

```cpp
#include <Fl_Gl_Window.H>
```

Inheritance diagram for Fl_Gl_Window:

```
Fl_Widget
└─ Fl_Group
    └─ Fl_Window
        └─ Fl_Gl_Window
            └─ Fl_Glut_Window
```

Public Member Functions

- `Fl_Gl_Window` const ∗ *as_gl_window () const FL_OVERRIDE
- `Fl_Gl_Window` ∗ *as_gl_window () FL_OVERRIDE
  - Returns an `Fl_Gl_Window` pointer if this widget is an `Fl_Gl_Window`.
- `int can_do ()`
  - Returns non-zero if the hardware supports the current OpenGL mode.
- `int can_do_overlay ()`
  - Returns true if the hardware overlay is possible.
- `GLContext context () const`
  - Returns a pointer to the window's OpenGL rendering context.
• void context (GLContext, int destroy_flag=0)
  Sets a pointer to the GLContext that this window is using.

• char context_valid () const
  Will only be set if the OpenGL context is created or recreated.

• void context_valid (char v)
  See char Fl_Gl_Window::context_valid() const.

• Fl_Gl_Window (int W, int H, const char ∗=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 () FL_OVERRIDE
  Forces the window to be drawn, this window is also made current and calls draw().

• int handle (int) FL_OVERRIDE
  Handle some FLTK events as needed.

• void hide () FL_OVERRIDE
  Hides the window and destroys the OpenGL context.

• void hide_overlay ()
  Hides the window if it is not this window, does nothing in Windows.

• void invalidate ()
  The invalidate() method turns off valid() and is equivalent to calling value(0).

• void make_current ()
  The make_current() method selects the OpenGL context for the widget.

• void make_overlay_current ()
  Selects the OpenGL context for the widget's overlay.

• Fl_Mode mode () const
  Returns the current OpenGL capabilites of the window.

• int mode (const int ∗a)
  Set the OpenGL capabilites of the window using platform-specific data.

• int mode (int a)
  Set or change the OpenGL capabilites of the window.

• void ortho ()
  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.

• int pixel_h ()
  Gives the window height in OpenGL pixels.

• int pixel_w ()
  Gives the window width in OpenGL pixels.

• float pixels_per_unit ()
  The number of pixels per FLTK unit of length for the window.

• void redraw_overlay ()
  Causes draw_overlay() to be called at a later time.

• void resize (int, int, int, int) FL_OVERRIDE
  Changes the size or position of the widget.

• void show () FL_OVERRIDE
  Makes a widget visible.

• void show (int a, char ∗∗b)
  Same as Fl_Window::show(int a, char ∗∗b)

• void swap_buffers ()
  The swap_buffers() method swaps the back and front buffers.

• char valid () const
  Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.
• void **valid** (char v)
  See char Fl_Gl_Window::valid() const.
• **~Fl_Gl_Window** ()
  The destructor removes the widget and destroys the OpenGL context associated with it.

### Static Public Member Functions

• static int **can_do** (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

• void **draw** () FL_OVERRIDE
  Draws the Fl_Gl_Window.
• void **draw_begin** ()
  Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.
• void **draw_end** ()
  To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

### Friends

• class Fl_Gl_Window_Driver

### Additional Inherited Members

### 33.44.1 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.

### 33.44.2 Constructor & Destructor Documentation

Generated by Doxygen
33.44.2.1 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.

33.44.2.2 Fl_Gl_Window() [2/2]

Fl_Gl_Window::Fl_Gl_Window ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗ l = 0 ) [inline]

Creates a new Fl_Gl_Window widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

33.44.3 Member Function Documentation

33.44.3.1 as_gl_window() [1/2]

Fl_Gl_Window const ∗ Fl_Gl_Window::as_gl_window ( ) const [inline], [virtual]
Reimplemented from Fl_Widget.

33.44.3.2 as_gl_window() [2/2]

Fl_Gl_Window ∗ Fl_Gl_Window::as_gl_window ( ) [inline], [virtual]
Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Gl_Window.
If it returns non-NULL, then the widget in question is derived from Fl_Gl_Window.

Return values

| NULL | if this widget is not derived from Fl_Gl_Window. |

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_window()
Reimplemented from Fl_Widget.

33.44.3.3 can_do()

static int Fl_Gl_Window::can_do ( 
    const int ∗ m ) [inline], [static]
Returns non-zero if the hardware supports the given OpenGL mode.
See also

Fl_Gl_Window::mode(const int ∗a)
33.44.3.4  
**can_do_overlay()**

```cpp
def can_do_overlay() -> bool:
    # Implementation...
```

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.

33.44.3.5  
**context()** [1/2]

```cpp
def context() -> GLContext:
    # Implementation...
```

Returns a pointer to the window's OpenGL rendering context.

See also

```cpp
void context(GLContext c, int destroy_flag)
```

33.44.3.6  
**context()** [2/2]

```cpp
def context(GLContext v, int destroy_flag = 0):
    # Implementation...
```

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.

33.44.3.7  
**context_valid()**

```cpp
def context_valid() -> bool:
    # Implementation...
```

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.

33.44.3.8  
**draw()**

```cpp
def draw()
```

 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
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
if (!valid()) {
    glLoadIdentity();
}
```
void mywindow::draw() {
    // draw 3d graphics scene
    Fl_Gl_Window::draw();
    // -- or --
    draw_begin();
    Fl_Window::draw();
    // other 2d drawing calls, overlays, etc.
    draw_end();
}

Implements Fl_Widget.
Reimplemented in Fl_Glut_Window.

33.44.3.9 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

33.44.3.10 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

33.44.3.11 flush()

void Fl_Gl_Window::flush ( ) [virtual]
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.

33.44.3.12 handle()

int Fl_Gl_Window::handle ( int event ) [virtual]
Handle some FLTK events as needed.
Reimplemented from Fl_Widget.
Reimplemented in Fl_Glut_Window.

33.44.3.13 hide()

void Fl_Gl_Window::hide ( ) [virtual]
Hides the window and destroys the OpenGL context.
Reimplemented from Fl_Widget.
### 33.44.3.14 make_current()

```cpp
void Fl_Gl_Window::make_current() {
    The `make_current()` method selects the OpenGL context for the widget. It is called automatically prior to the `draw()` method being called and can also be used to implement feedback and/or selection within the `handle()` method.
}
```

### 33.44.3.15 make_overlay_current()

```cpp
void Fl_Gl_Window::make_overlay_current() {
    Selects the OpenGL context for the widget's overlay. This method is called automatically prior to the `draw_overlay()` method being called and can also be used to implement feedback and/or selection within the `handle()` method.
}
```

### 33.44.3.16 mode() 1/3

```cpp
Fl_Mode Fl_Gl_Window::mode() const [inline]
Returns the current OpenGL capabilities of the window. Don't use this if capabilities were set through `Fl_Gl_Window::mode(int *)`.
```

### 33.44.3.17 mode() 2/3

```cpp
int Fl_Gl_Window::mode(int *) [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.

### 33.44.3.18 mode() 3/3

```cpp
int Fl_Gl_Window::mode(int a) [inline]
Set or change the OpenGL capabilities of the window. The value can be any of the following OR'd together:

- `FL_RGB` - RGB color (not indexed)
- `FL_RGB8` - RGB color with at least 8 bits of each color
- `FL_INDEX` - Indexed mode
- `FL_SINGLE` - not double buffered
- `FL_DOUBLE` - double buffered
- `FL_ACCUM` - accumulation buffer
- `FL_ALPHA` - alpha channel in color
• FL_DEPTH - depth buffer
• FL_STENCIL - stencil buffer
• FL_MULTISAMPLE - multisample antialiasing
• FL_OPENGL3 - use OpenGL version 3.0 or more.

FL_RGB and FL_SINGLE have a value of zero, so they are "on" unless you give FL_INDEX or FL_DOUBLE. If the desired combination cannot be done, FLTK will try turning off FL_MULTISAMPLE. If this also fails the show() will call Fl::error() and not show the window.

You can change the mode while the window is displayed. This is most useful for turning double-buffering on and off. Under X this will cause the old X window to be destroyed and a new one to be created. If this is a top-level window this will unfortunately also cause the window to blink, raise to the top, and be de-iconized, and the xid() will change, possibly breaking other code. It is best to make the GL window a child of another window if you wish to do this!

mode() must not be called within draw() since it changes the current context.

The FL_OPENGL3 flag is required to access OpenGL version 3 or more under the X11 and MacOS platforms; it's optional under Windows and Wayland. See more details in Using OpenGL 3.0 (or higher versions).

Version
the FL_OPENGL3 flag appeared in version 1.3.4

### 33.44.3.19 ortho()

```cpp
void Fl_Gl_Window::ortho ( )
```

Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.

If you are drawing 2D images, your draw() method may want to call this if valid() is false.

### 33.44.3.20 pixel_h()

```cpp
int Fl_Gl_Window::pixel_h ( ) [inline]
```

Gives the window height in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::h() which is expressed in FLTK units, may differ from the window height in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use pixel_h() rather than h(). Method pixel_h() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version
1.3.4

### 33.44.3.21 pixel_w()

```cpp
int Fl_Gl_Window::pixel_w ( ) [inline]
```

Gives the window width in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::w() which is expressed in FLTK units, may differ from the window width in pixels. Calls to OpenGL functions expecting pixel values (e.g., glViewport) should therefore use pixel_w() rather than w(). Method pixel_w() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version
1.3.4
33.44.3.22 pixels_per_unit()

float Fl_Gl_Window::pixels_per_unit ()
The number of pixels per FLTK unit of length for the window.
This method dynamically adjusts its value when the GUI is rescaled or when the window is moved to/from displays of
distinct resolutions. This method is useful, e.g., to convert, in a window's handle() method, the FLTK units returned
by Fl::event_x() and Fl::event_y() to the pixel units used by the OpenGL source code.

Version
1.3.4

33.44.3.23 redraw_overlay()

void Fl_Gl_Window::redraw_overlay ( }
Causes draw_overlay() to be called at a later time.
Initially the overlay is clear. If you want the window to display something in the overlay when it first appears, you
must call this immediately after you show() your window.

33.44.3.24 resize()

void Fl_Gl_Window::resize ( 
    int x,
    int y,
    int w,
    int h ) [virtual]
Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does not
call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster
way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or
size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

| in  | x,y | new position relative to the parent window |
| in  | w,h | new size                                  |

See also

    position(int,int), size(int,int)

Reimplemented from Fl_Widget.

33.44.3.25 show()

void Fl_Gl_Window::show ( ) [virtual]
Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other
events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget
and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.
Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as
this will send false FL_SHOW or FL_HIDE events to the widget. redraw() is called if necessary on this or the parent.
See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.

### 33.44.3.26 swap_buffers()

```cpp
definition 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.

### 33.44.3.27 valid()

```cpp
definition Fl_Gl_Window::valid
```

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
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

### 33.45 Fl_Glut_Bitmap_Font Struct Reference

fltk glut font/size attributes used in the glutXXX functions

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

**Public Attributes**

- `Fl_Font font`
- `Fl_Fontsize size`

### 33.45.1 Detailed Description

fltk glut font/size attributes used in the glutXXX functions

The documentation for this struct was generated from the following file:

- glut.H

### 33.46 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

### 33.47 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

### 33.48 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

### 33.49 Fl_Glut_StrokeVertex Struct Reference

**Public Attributes**

- GLfloat X
- GLfloat Y

The documentation for this struct was generated from the following file:

- glut.H

### 33.50 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
   |
   v
Fl_Group
   |
   v
Fl_Window
   |
   v
Fl_Gl_Window
   |
   v
Fl_Glut_Window
```

Generated by Doxygen
Public Member Functions

- **Fl_Glut_Window** (int w, int h, const char *t=0)
  
  Creates a glut window, registers to the glut windows list.

- **Fl_Glut_Window** (int x, int y, int w, int h, const char *t=0)
  
  Creates a glut window, registers to the glut windows list.

- void **make_current**()

- **~Fl_Glut_Window**()
  
  Destroys the glut window, first unregister it from the glut windows list.

Public Attributes

- void (display)()

- void (entry)(int)

- void (keyboard)(uchar, int x, int y)

- int menu [3]

- void (motion)(int x, int y)

- void (mouse)(int b, int state, int x, int y)

- int number

- void (overlaydisplay)()

- void (passivemotion)(int x, int y)

- void (reshape)(int w, int h)

- void (special)(int, int x, int y)

- void (visibility)(int)

Protected Member Functions

- void **draw**() FL_OVERRIDE
  
  Draws the Fl_Gl_Window.

- void **draw_overlay**() FL_OVERRIDE
  
  You must implement this virtual function if you want to draw into the overlay.

- int **handle**(int) FL_OVERRIDE
  
  Handle some FLTK events as needed.

Additional Inherited Members

33.50.1 Detailed Description

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

33.50.2 Member Function Documentation

33.50.2.1 **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);
  // Draw white 'X'
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_LINE_STRIP);
  glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
  glBegin(GL_LINE_STRIP);
  glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

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.

### 33.50.2.2 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.

### 33.50.2.3 handle()

int Fl_Glut_Window::handle ( int event ) [protected], [virtual]
Handle some FLTK events as needed.
Reimplemented from Fl_Gl_Window.
The documentation for this class was generated from the following files:
- `glut.H`
- `glut_compatibility.cxx`

### 33.51 Fl_Group Class Reference

The Fl_Group class is the FLTK container widget.
#include <Fl_Group.H>

Inheritance diagram for Fl_Group:

```
Fl_Widget
   ▼
   Fl_Group

   | ▼
   Fl_Browser_
   Fl_Color_Chooser
   Fl_Flex
   Fl_Help_View
   Fl_Input_Choice
   Fl_Pack
   Fl_Scroll
   Fl_Spinner
   Fl_Table
   Fl_Tabs
   Fl_Text_Display
   Fl_Tile
   Fl_Tree
   Fl_Window
   Fl_Wizard
```

## Public Member Functions

- **FL_Widget** * _ddfdesign_kludge ()**
  
  This is for forms compatibility only.

- **void add (FL_Widget &)**
  
  The widget is removed from its current group (if any) and then added to the end of this group.

- **void add (FL_Widget &o)**
  
  See void Fl_Group::add(FL_Widget &w)

- **void add_resizable (FL_Widget &o)**
  
  Adds a widget to the group and makes it the resizable widget.

- **FL_Widget *const * array () const**
  
  Returns a pointer to the array of children.

- **FL_Group *const * as_group () const FL_OVERRIDE**

- **FL_Group * as_group () FL_OVERRIDE**
  
  Returns an FL_Group pointer if this widget is an FL_Group.

- **void begin ()**
  
  Sets the current group so you can build the widget tree by just constructing the widgets.

- **FL_Widget * child (int n) const**
  
  Returns array()[n].

- **int children () const**
  
  Returns how many child widgets the group has.

- **void clear ()**
Deletes all child widgets from memory recursively.
• unsigned int clip_children ()
  Returns the current clipping mode.
• void clip_children (int c)
  Controls whether the group widget clips the drawing of child widgets to its bounding box.
• virtual int delete_child (int n)
  Removes the widget at index from the group and deletes it.
• void end ()
  Exactly the same as current(this->parent()).
• int find (const Fl_Widget &o) const
  See int Fl_Group::find(const Fl_Widget *w) const.
• int find (const Fl_Widget *) const
  Searches the child array for the widget and returns the index.
• Fl_Group (int, int, int, int, const char *)=0)
  Creates a new Fl_Group widget using the given position, size, and label string.
• void focus (Fl_Widget *)
• void forms_end ()
  This is for forms compatibility only.
• int handle (int) FL_OVERRIDE
  Handles the specified event.
• void init_sizes ()
  Resets the internal array of widget sizes and positions.
• void insert (Fl_Widget &, int i)
  The widget is removed from its current group (if any) and then inserted into this group.
• void insert (Fl_Widget &, Fl_Widget *) before)
  This does insert(w, find(before)).
• void remove (Fl_Widget &)
  Removes a widget from the group but does not delete it.
• void remove (Fl_Widget *)
  Removes the widget o from the group.
• void remove (int index)
  Removes the widget at index from the group but does not delete it.
• Fl_Widget * resizable () const
  Returns the group’s resizable widget.
• void resizable (Fl_Widget &)
  Sets the group’s resizable widget.
• void resizable (Fl_Widget *)
  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
• void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl_Group widget and all of its children.
• virtual ~Fl_Group ()
  The destructor also deletes all the children.

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 () FL_OVERRIDE**
  
  Draws the widget.

- **void draw_child (Fl_Widget &widget) const**
  
  Forces a child to redraw.

- **void draw_children ()**
  
  Draws all children of the group.

- **void draw_outside_label (const Fl_Widget &widget) const**
  
  Parents normally call this to draw outside labels of child widgets.

- **virtual int on_insert (Fl_Widget *, int)**
  
  Allow derived groups to act when a widget is added as a child.

- **virtual int on_move (int, int)**
  
  Allow derived groups to act when a widget is moved within the group.

- **virtual void on_remove (int)**
  
  Allow derived groups to act when a child widget is removed from the group.

- **int * sizes ()**
  
  Returns the internal array of widget sizes and positions.

- **void update_child (Fl_Widget &widget) const**
  
  Draws a child only if it needs it.

Additional Inherited Members

33.51.1 Detailed Description

The **Fl_Group** class is the FLTK container widget.
It maintains an array of child widgets. These children can themselves be any widget including **Fl_Group**. The most important subclass of **Fl_Group** is **Fl_Window**, however groups can also be used to control radio buttons or to enforce resize behavior.
The tab and arrow keys are used to move the focus between widgets of this group, and to other groups. The only modifier grabbed is shift (for shift-tab), so that ctrl-tab, alt-up, and such are free for the app to use as shortcuts.
To remove a widget from the group and destroy it, in 1.3.x (and up) you can simply use:
```
delete some_widget;
```
...and this will trigger proper scheduling of the widget’s removal from its parent group.
If used as a child of **Fl_Tabs**, setting when(FL_WHEN_CLOSED) will enable the Close button in the corresponding tab. If the user clicks the Close button, the callback of this group will be called with the callback reason FL_REASON_CLOSED.

33.51.2 Constructor & Destructor Documentation

33.51.2.1 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.
33.51.2.2 \texttt{\textasciitilde Fl\_Group()}

\begin{verbatim}
Fl\_Group::\texttt{\textasciitilde Fl\_Group ( ) [virtual]}
The destructor also deletes all the children.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. It is allowed that the 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 \textit{before destroying} the group - otherwise the child widgets' destructors would be called twice!
\end{verbatim}

33.51.3 Member Function Documentation

33.51.3.1 \texttt{array()}

\begin{verbatim}
Fl\_Widget \*const \* Fl\_Group::\texttt{array ( ) const}
\end{verbatim}

Returns a pointer to the array of children.

\textbf{Note}

This pointer is only valid until the next time a child is added or removed.

33.51.3.2 \texttt{as\_group() [1/2]}

\begin{verbatim}
Fl\_Widget const \* Fl\_Group::\texttt{as\_group ( ) const [inline], [virtual]}
\end{verbatim}

Reimplemented from Fl\_Widget.

33.51.3.3 \texttt{as\_group() [2/2]}

\begin{verbatim}
Fl\_Group \* Fl\_Group::\texttt{as\_group ( ) [inline], [virtual]}
\end{verbatim}

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.

\textbf{Example:}

\begin{verbatim}
void my\_callback (Fl\_Widget \*w, void \*) {
    Fl\_Group \*g = w->as\_group();
    if (g)
        printf ("This group has %d children\n", g->children());
    else
        printf ("This widget is not a group!\n");
}
\end{verbatim}

\textbf{Return values}

\texttt{NULL} if this widget is not derived from Fl\_Group.

\textbf{Note}

This method is provided to avoid dynamic\_cast.

\textbf{See also}

Fl\_Widget::as\_window(), Fl\_Widget::as\_gl\_window()

Reimplemented from Fl\_Widget.
33.51.3.4 begin()

```cpp
void Fl_Group::begin ( )
```

Sets the current group so you can build the widget tree by just constructing the widgets.

`begin()` is automatically called by the constructor for `Fl_Group` (and thus for `Fl_Window` as well). `begin()` is exactly the same as `current(this)`. Don't forget to `end()` the group or window!

33.51.3.5 bounds()

```cpp
Fl_Rect * Fl_Group::bounds ( ) [protected]
```

Returns the internal array of widget sizes and positions.

If the `bounds()` array does not exist, it will be allocated and filled with the current widget sizes and positions. The `bounds()` array stores the initial positions of widgets as `Fl_Rect`'s. The size of the array is `children() + 2`.

- The first `Fl_Rect` is the group,
- the second is the resizable (clipped to the group),
- the rest are the children.

This is a convenient order for the resize algorithm.

If the group and/or the `resizable()` is a `Fl_Window` (or subclass) then the `x()` and `y()` coordinates of their respective `Fl_Rect`'s are zero.

**Note**

You should never need to use this `protected` method directly, unless you have special needs to rearrange the children of a `Fl_Group`. `Fl_Tile` uses this to rearrange its widget positions. The returned array should be considered read-only. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call `init_sizes()`.

Include `<FL/Fl Rect.H>` if you want to access the `bounds()` array in your derived class. `Fl Rect.H` is intentionally not included by `Fl Group.H` to avoid unnecessary dependencies.

Returns

Array of `Fl Rect`'s with widget positions and sizes. The returned array is only valid until `init_sizes()` is called or widgets are added to or removed from the group.

See also

`init_sizes()`

Since

FLTK 1.4.0

33.51.3.6 child()

```cpp
Fl_Widget * Fl_Group::child ( int n ) const [inline]
```

Returns `array()[n]`.

*No range checking is done!*

33.51.3.7 clear()

```cpp
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.
### 33.51.3.8 clip_children() [1/2]

```cpp
unsigned int Fl_Group::clip_children ( ) [inline]
```

Returns the current clipping mode.

Returns

true, if clipping is enabled, false otherwise.

See also

```cpp
void Fl_Group::clip_children(int c)
```

### 33.51.3.9 clip_children() [2/2]

```cpp
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.

### 33.51.3.10 current() [1/2]

```cpp
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).

### 33.51.3.11 current() [2/2]

```cpp
void Fl_Group::current ( 
    Fl_Group * g ) [static]
```

Sets the current group.

See also

```cpp
Fl_Group::current()
```

### 33.51.3.12 delete_child()

```cpp
int Fl_Group::delete_child ( 
    int index ) [virtual]
```

Removes the widget at index from the group and deletes it.
This method does nothing if index is out of bounds.
This method differs from the remove() method in that it deletes the widget from memory. Since this method is virtual it can be reimplemented in subclasses with additional requirements and consequences. See the documentation of subclasses.
Many subclasses don't need to reimplement this method.

Note

This method may refuse to remove and delete the widget if it is an essential part of the Fl_Group, for instance a scrollbar in an Fl_Scroll group. In this case the widget is neither removed nor deleted.

This method does not call init_sizes() or redraw(). This is left to user code if necessary.
Returns 0 if the widget was removed and deleted. Return values > 0 are reserved for use by FLTK core widgets. Return values < 0 are free to be used by user defined widgets.

Todo Reimplementation of Fl_Group::delete_child(int) in more FLTK subclasses. This is not yet complete.
Parameters

| in | index | index of child to be removed |

Returns

success (0) or error code

Return values

| 0 | success |
| 1 | index out of range |
| 2 | widget not allowed to be removed (see note) |
| >2 | reserved for FLTK use |

Since FLTK 1.4.0

Reimplemented in Fl_Scroll.

33.51.3.13 draw()

```cpp
void Fl_Group::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.


33.51.3.14 draw_child()

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

Forces a child to redraw.

This draws a child widget, if it is not clipped. The damage bits are cleared after drawing.

33.51.3.15 draw_children()

```cpp
void Fl_Group::draw_children ( ) [protected]
```

Draws all children of the group.

This is useful, if you derived a widget from Fl_Group and want to draw a special border or background. You can call `draw_children()` from the derived `draw()` method after drawing the box, border, or background.

33.51.3.16 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.
33.51 Fl_Group Class Reference

33.51.3.17 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.

33.51.3.18 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

- `Fl_Widget::take_focus();`

33.51.3.19 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**

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

- `Fl_Event`

Reimplemented from `Fl_Widget`. Reimplemented in `Fl_Table`, `Fl_Text_Display`, `Fl_Text_Editor`, `Fl_Tree`, `Fl_Spinner`, `Fl_Table_Row`, `Fl_Tile`, `Fl_Help_View`, `Fl_Scroll`, `Fl_Tabs`, `Fl_Window`, and `Fl_Glut_Window`.

33.51.3.20 init_sizes()

```cpp
void Fl_Group::init_sizes ()
```

Resets the internal array of widget sizes and positions. The `Fl_Group` widget keeps track of the original widget sizes and positions when resizing occurs so that if you resize a window back to its original size the widgets will be in the correct places. If you rearrange the widgets in your group, call this method to register the new arrangement with the `Fl_Group` that contains them. If you add or remove widgets, this will be done automatically.
Note

The internal array of widget sizes and positions will be allocated and filled when the next `resize()` occurs. For more information on the contents and structure of the `bounds()` array see `bounds()`.

See also

- `bounds()`
- `sizes()` (deprecated)

### 33.51.3.21 insert() [1/2]

```cpp
void Fl_Group::insert (Fl_Widget & o, int index)
```

The widget is removed from its current group (if any) and then inserted into this group. It is put at index n - or at the end, if n > children(). This can also be used to rearrange the widgets inside a group.

### 33.51.3.22 insert() [2/2]

```cpp
void Fl_Group::insert (Fl_Widget & o, Fl_Widget * before) [inline]
```

This does `insert(w, find(before))`. This will append the widget if `before` is not in the group.

### 33.51.3.23 on_insert()

```cpp
int Fl_Group::on_insert (Fl_Widget * candidate, int index) [protected], [virtual]
```

Allow derived groups to act when a widget is added as a child. Widgets derived from `Fl_Group` may store additional data for their children. Overriding this method will allow derived classes to generate these data structures just before the child is added.

This method usually returns the same index that was given in the parameters. By setting a new index, the position of other widgets in the child pointer array can be preserved (e.g. `Fl_Scroll` keeps its scroll bars as the last two children).

By returning -1, `Fl_Group::insert` will not add the child to `array_`. This is not recommended, but `Fl_Table` does something similar to forward children to a hidden group.

**Parameters**

<table>
<thead>
<tr>
<th>candidate</th>
<th>the candidate will be added to the child array_ after this method returns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>add the child at this position in the array_</td>
</tr>
</tbody>
</table>

**Returns**

- index to position the child as planned
- a new index to force the child to a different position
- -1 to keep the group from adding the candidate

Reimplemented in `Fl_Scroll`, and `Fl_Tabs`.

### 33.51.3.24 on_move()

```cpp
int Fl_Group::on_move (```

Generated by Doxygen
33.51 Fl_Group Class Reference

```cpp
int oldIndex,
int newIndex ) [protected], [virtual]
```

Allow derived groups to act when a widget is moved within the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived
classes to move these data structures just before the child itself is moved.
This method usually returns the new index that was given in the parameters. By setting a different destination index,
the position of other widgets in the child pointer array can be preserved.
By returning -1, Fl_Group::insert will not move the child.

**Parameters**

| oldIndex | the current index of the child that will be moved |
| newIndex | the new index of the child |

**Returns**

- newIndex to position the child as planned
- a different index to force the child to a different position
- -1 to keep the group from moving the child

Reimplemented in Fl_Scroll, and Fl_Tabs.

33.51.3.25 on_remove()

```cpp
void Fl_Group::on_remove ( int index ) [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived
classes to remove these data structures just before the child is removed.

**Parameters**

| index | remove the child at this position in the array_ |

Reimplemented in Fl_Flex, and Fl_Tabs.

33.51.3.26 remove() [1/3]

```cpp
void Fl_Group::remove ( Fl_Widget & o )
```

Removes a widget from the group but does not delete it.
This method does nothing if the widget is not a child of the group.
This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

**Note**

If you have the child's index anyway, use remove(int index) instead, because this doesn't need a child lookup
in the group's table of children. This can be much faster, if there are lots of children.

33.51.3.27 remove() [2/3]

```cpp
void Fl_Group::remove ( Fl_Widget * o ) [inline]
```

Removes the widget  from the group.

Generated by Doxygen
See also

```cpp
void remove(Fl_Widget&)
```

### 33.51.3.28 remove() [3/3]

```cpp
void Fl_Group::remove(
    int index)
```

Removes the widget at \texttt{index} from the group but does not delete it. This method does nothing if \texttt{index} is out of bounds. This method differs from the \texttt{clear()} method in that it only affects a single widget and does not delete it from memory.

Since

\texttt{FLTK 1.3.0}

### 33.51.3.29 resizable() [1/3]

```cpp
Fl_Widget * Fl_Group::resizable ( ) const [inline]
```

Returns the group’s resizable widget.
See void \texttt{Fl_Group::resizable(Fl_Widget *o)}

### 33.51.3.30 resizable() [2/3]

```cpp
void Fl_Group::resizable (
    Fl_Widget & o ) [inline]
```

Sets the group’s resizable widget.
See void \texttt{Fl_Group::resizable(Fl_Widget *o)}

### 33.51.3.31 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 \texttt{Fl_Window} and \texttt{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 \texttt{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.

\textbf{Note}

The resizable of a group \textbf{must be one of}

- NULL
- the group itself
- a direct child of the group.

If you set any other widget that is not a direct child of the group as its resizable then the behavior is undefined. This is \textbf{not} checked by \texttt{Fl_Group} for historical reasons.

In these examples the gray area is the resizable:
It is possible to achieve any type of resize behavior by using an invisible Fl_Box as the resizable and/or by using a hierarchy of Fl_Group widgets, each with their own resizing strategies. See the How Does Resizing Work? chapter for more examples and detailed explanation.

Note

The resizable() widget of a window can also affect the window's resizing behavior if Fl_Window::size_range() is not called. Please see Fl_Window::default_size_range() for more information on how the default size range is calculated.

See also

Fl_Window::size_range()
Fl_Window::default_size_range()

33.51.3.32 resize()

void Fl_Group::resize (int X, int Y, int W, int H) [virtual]

Resizes the Fl_Group widget and all of its children.

The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)
33.51.33 sizes()

int * Fl_Group::sizes ( ) [protected]

Returns the internal array of widget sizes and positions.
For backward compatibility with FLTK versions before 1.4.
The sizes() array stores the initial positions of widgets as (left, right, top, bottom) quads. The first quad is the group,
the second is the resizable (clipped to the group), and the rest are the children. If the group and/or the resizable() is a Fl_Window,
then the first (left) and third (top) entries of their respective quads (x,y) are zero.

Deprecated Deprecated since 1.4.0. Please use bounds() instead.

Note
This method will be removed in a future FLTK version (1.5.0 or higher).

Returns
Array of int's with widget positions and sizes. The returned array is only valid until init_sizes() is called or
widgets are added to or removed from the group.

Note
Since FLTK 1.4.0 the returned array is a read-only and re-ordered copy of the internal bounds() array. Do
not change its contents. If you need to rearrange children in a group, do so by resizing the children and call
init_sizes().

See also
bounds()
33.52 Fl_Help_Block Struct Reference

Public Attributes

- Fl_Color bgcolor
- uchar border
- const char * end
- int h
- int line [32]
- int ol
- int ol_num
- const char * start
- int w
- int x
- int y

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

33.53 Fl_Help_Dialog Class Reference

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.

Public Member Functions

- Fl_Help_Dialog ()
  The constructor creates the dialog pictured above.
- int h ()
  Returns the position and size of the help dialog.
- void hide ()
  Hides the Fl_Help_Dialog window.
- int load (const char * f)
  Loads the specified HTML file into the Fl_Help_View widget.
- void position (int xx, int yy)
  Set the screen position of the dialog.
- void resize (int xx, int yy, int ww, int hh)
  Change the position and size of the dialog.
- void show ()
  Shows the Fl_Help_Dialog window.
- void show (int argc, char * * argv)
  Shows the main Help dialog window Delegates call to encapsulated window void Fl_Window::show(int argc, char ** argv) instance method.
- Fl_Fontsize textsize ()
  Sets or gets the default text size for the help view.
- void textsize (Fl_Fontsize s)
  Sets or gets the default text size for the help view.
- void topline (const char * n)
  Sets the top line in the Fl_Help_View widget to the named or numbered line.
- void topline (int n)
  Sets the top line in the Fl_Help_View widget to the named or numbered line.
- const char * value () const
  The first form sets the current buffer to the string provided and reformats the text.
- void value (const char * f)
The first form sets the current buffer to the string provided and reformatsthe 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.

### 33.53.1 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 33.19 Fl_Help_Dialog](image)

### 33.53.2 Member Function Documentation

#### 33.53.2.1 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
33.53.2.2 show()

```c
void Fl_Help_Dialog::show()
```

Shows the Fl_Help_Dialog window.
Shows the main Help Dialog Window Delegates call to encapsulated window_ void Fl_Window::show() method.

33.53.2.3 textSize()

```c
void Fl_Help_Dialog::textsize(Fl_Fontsize s)
```

Sets or gets the default text size for the help view.
Sets the internal Fl_Help_View instance text size.
Delegates call to encapsulated view_ void Fl_Help_View::textsize(Fl_Fontsize s) instance method

33.53.2.4 value() [1/2]

```c
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.

33.53.2.5 value() [2/2]

```c
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

33.54 Fl_Help_Font_Stack Struct Reference

Public Member Functions

- `size_t count() const`
  
  Gets the current count of font style elements in the stack.

- `Fl_Help_Font_Stack()`
  
  font stack construction, initialize attributes.

- `void init(Fl_Font f, Fl_Fontsize s, Fl_Color c)`
  
  Pops from the stack the font style triplet and calls fl_font() & fl_color() adequately.

- `void pop(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`

### 33.55 Fl_Help_Font_Style Struct Reference

**Fl_Help_View** font stack element definition.

```c
#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.

### 33.55.1 Detailed Description

**Fl_Help_View** font stack element definition.

The documentation for this struct was generated from the following file:

- `Fl_Help_View.H`

### 33.56 Fl_Help_Link Struct Reference

Definition of a link for the html viewer.

```c
#include <Fl_Help_View.H>
```

Public Attributes

- char **filename** [192]
  Reference filename.
- int **h**
  Height of link text.
- char **name** [32]
  Link target (blank if none)
- int **w**
  Width of link text.
33.57 Fl_Help_Target Struct Reference

Fl_Help_Target structure.
#include <Fl_Help_Target.H>

Public Attributes
• char name [32]
  Target name.
• int y
  Y offset of target.

33.57.1 Detailed Description
Fl_Help_Target structure.
The documentation for this struct was generated from the following file:
  • Fl_Help_Target.H

33.58 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)

Generated by Doxygen
Finds the specified string \( s \) at starting position \( p \).

- **Fl_Help_View** (int xx, int yy, int ww, int hh, const char *l=0)
  
The constructor creates the Fl_Help_View widget at the specified position and size.

- int **handle** (int) FL_OVERRIDE
  
  Handles events in the widget.

- int **leftline** () const
  
  Gets the left position in pixels.

- void **leftline** (int)
  
  Scrolls the text to the indicated position, given a pixel column.

- void **link** (Fl_Help_Func *fn)
  
  This method assigns a callback function to use when a link is followed or a file is loaded (via Fl_Help_View::load()) that requires a different file or path.

- int **load** (const char *f)
  
  Loads the specified file.

- void **resize** (int, int, int, int) FL_OVERRIDE
  
  Resizes the help widget.

- int **scrollbar_size** () const
  
  Gets the current size of the scrollbars' troughs, in pixels.

- void **scrollbar_size** (int newSize)
  
  Sets the pixel size of the scrollbars' troughs to \( \text{newSize} \), 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 \texttt{draw () FL\_OVERRIDE}
  
  Draws the \texttt{Fl\_Help\_View} widget.

Additional Inherited Members

33.58.1 Detailed Description

The \texttt{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:

- \texttt{A}: HREF/NAME
- \texttt{B}
- \texttt{BODY}: BGCOLOR/TEXT/LINK
- \texttt{BR}
- \texttt{CENTER}
- \texttt{CODE}
- \texttt{DD}
- \texttt{DL}
- \texttt{DT}
- \texttt{EM}
- \texttt{FONT}: COLOR/SIZE/FACE=(helvetica/arial/sans/times/serif/symbol/courier)
- \texttt{H1/H2/H3/H4/H5/H6}
- \texttt{HEAD}
- \texttt{HR}
- \texttt{I}
- \texttt{IMG}: SRC/WIDTH/HEIGHT/ALT
- \texttt{KBD}
- \texttt{LI}
- \texttt{OL}
- \texttt{P}
- \texttt{PRE}
- \texttt{STRONG}
- \texttt{TABLE}: TH/TD/TR/BORDER/BGCOLOR/COLSPAN/ALIGN=CENTER|RIGHT|LEFT
- \texttt{TITLE}
- \texttt{TT}
- \texttt{U}
- \texttt{UL}
- \texttt{VAR}
Supported color names:
- black, red, green, yellow, blue, magenta, fuchsia, cyan, aqua, white, gray, grey, lime, maroon, navy, olive, purple, silver, teal.

Supported urls:
- Internal: file:
- External: http: ftp: https: ipp: mailto: news:

Quoted char names:
- Aacute aacute Acirc acirc acute AElig aelig Agrave agrave amp Aring aring Atilde atilde Auml auml
- brvbar bull
- Ccedil ccedil cedil cent copy curren
dagger deg divide
- Ecute eacute Ecirc ecirc Egrave egrave ETH eth Euml euml euro
- frac12 frac14 frac34
- gt
- Iacute ibackquote iexcl Igrave igrave iquest Iuml iuml
- laquo lt
- macr micro middot
- nbsp not Ntilde ntilde
- Oacute oacute Ocirc ocirc Ograve ograve ordf ordm Oslash oslash Otilde otilde Ouml ouml
- para permil plusmn pound
- quot
- raquo reg
- sect shy sup1 sup2 sup3 szlig
- THORN thorn times trade
- Uacute uacute Ucirc ucirc Ugrave ugrave uml Uuml uuml
- Yacute yacute
- yen Yuml yuml

Note
You can't effectively set the box() to FL_NO_BOX, this would result in FL_DOWN_BOX being used as the boxtype of the widget. This is unexpected but can't be changed for backwards compatibility. If you don't want a frame around the widget you can use FL_FLAT_BOX instead.

33.58.2 Constructor & Destructor Documentation

33.58.2.1 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.
### 33.58.3 Member Function Documentation

#### 33.58.3.1 draw()

```cpp
void Fl_Help_View::draw (
    void ) [protected], [virtual]
```

Draws the Fl_Help_View widget.
Reimplemented from Fl_Group.

#### 33.58.3.2 find()

```cpp
int Fl_Help_View::find (  
    const char * s,  
    int p = 0)
```

Finds the specified string `s` at starting position `p`.
The argument `p` and the return value are offsets in Fl_Help_View::value(), counting from 0. If `p` is out of range, 0 is used.
The string comparison is simple but honors some special cases:
- the specified string `s` must be in UTF-8 encoding
- HTML tags in value() are filtered (not compared as such, they never match)
- HTML entities like '&lt;' or '&x#20ac;' are converted to Unicode (UTF-8)
- ASCII characters (7-bit, < 0x80) are compared case insensitive
- every newline (LF, "\n") in value() is treated like a single space
- all other strings are compared as-is (byte by byte)

**Todo** complex HTML entities for Unicode code points > 0x80 are currently treated like one byte (not character!) and do not (yet) match correctly ("<" matches "<" but "€" doesn't match "€", and "ü" doesn't match "ü")

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>s</code></th>
<th>search string in UTF-8 encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>p</code></td>
<td>starting position for search (0,...). Default = 0</td>
</tr>
</tbody>
</table>

Returns
- the matching position or -1 if not found

#### 33.58.3.3 handle()

```cpp
int Fl_Help_View::handle (  
    int event ) [virtual]
```

Handles events in the widget.
Reimplemented from Fl_Group.

#### 33.58.3.4 leftline()

```cpp
void Fl_Help_View::leftline (  
    int left )
```

Generated by Doxygen
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) |

### 33.58.3.5 link()

```cpp
void Fl_Help_View::link (Fl_Help_Func * fn) [inline]
```

This method assigns a callback function to use when a link is followed or a file is loaded (via `Fl_Help_View::load()`) that requires a different file or path.

The callback function receives a pointer to the `Fl_Help_View` widget and the URI or full pathname for the file in question. It must return a pathname that can be opened as a local file or NULL:

```c
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.

### 33.58.3.6 load()

```cpp
int Fl_Help_View::load (const char * f)
```

Loads the specified file.

This method loads the specified file or URL. The filename may end in a #name style target.

If the URL starts with `ftp`, `http`, `https`, `ipp`, `mailto`, or `news`, followed by a colon, FLTK will use `fl_open_uri()` to show the requested page in an external browser.

In all other cases, the URL is interpreted as a filename. The file is read and displayed in this browser. Note that MSWindows style backslashes are not supported in the file name.

**Parameters**

| in | f | filename or URL |

**Returns**

0 on success, -1 on error

**See also**

`fl_open_uri()`

### 33.58.3.7 resize()

```cpp
void Fl_Help_View::resize (int xx, int yy, int ww, int hh) [virtual]
```

Resizes the help widget.

Reimplemented from `Fl_Group`.
33.58.3.8 scrollbar_size() [1/2]

```cpp
int Fl_Help_View::scrollbar_size ( ) const [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

33.58.3.9 scrollbar_size() [2/2]

```cpp
void Fl_Help_View::scrollbar_size ( int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>newSize</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the scrollbar size in pixels.</td>
</tr>
<tr>
<td></td>
<td>If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()</td>
</tr>
</tbody>
</table>

See also

Fl::scrollbar_size()

33.58.3.10 topline() [1/2]

```cpp
void Fl_Help_View::topline ( const char * n )
```

Scrolls the text to the indicated position, given a named destination.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>n target name</th>
</tr>
</thead>
</table>

33.58.3.11 topline() [2/2]

```cpp
void Fl_Help_View::topline ( int top )
```

Scrolls the text to the indicated position, given a pixel line.
If the given pixel value `top` is out of range, then the text is scrolled to the top or bottom of the document, resp.

Parameters

| in | top top line number in pixels (0 = start of document) |
### 33.58.3.12 value()

```cpp
void Fl_Help_View::value (const char* val)
```

Sets the current help text buffer to the string provided and reformats the text.
The provided character string \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:

- Fl_Help_View.H
- Fl_Help_View.cxx

### 33.59 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.

```cpp
#include <Fl_Hold_Browser.H>
```

Inheritance diagram for Fl_Hold_Browser:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Browser_
  ↓
Fl_Browser
  ↓
Fl_Hold_Browser
```

#### Public Member Functions

- **\texttt{Fl_Hold_Browser}** (int \texttt{X}, int \texttt{Y}, int \texttt{W}, int \texttt{H}, const char* \texttt{L=0})

  
  Creates a new \texttt{Fl_Hold_Browser} widget using the given position, size, and label string.

#### Additional Inherited Members

#### 33.59.1 Detailed Description

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.

![Figure 33.20 Fl_Hold_Browser](image)

**Figure 33.20 Fl_Hold_Browser**

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 \texttt{when()}. See \texttt{Fl_Browser} for methods to add and remove lines from the browser.
33.59.2 Constructor & Destructor Documentation

33.59.2.1 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

33.60 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

33.61 Fl_Hor_Nice_Slider Class Reference

Single thumb tab slider.  
#include <Fl_Hor_Nice_Slider.H>

Inheritance diagram for 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

33.61.1 Detailed Description

Single thumb tab slider.

![Figure 33.21 Fl_Hor_Nice_Slider with various Fl::scheme() values](image)

The documentation for this class was generated from the following files:

- Fl_Hor_Nice_Slider.H
- Fl_Slider.cxx

33.62 Fl_Hor_Slider Class Reference

Horizontal Slider class.

```cpp
#include <Fl_Hor_Slider.H>
```

Inheritance diagram for Fl_Hor_Slider:

![Inheritance diagram](image)

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

33.62.1 Detailed Description

Horizontal Slider class.

See also

class Fl_Slider.

The documentation for this class was generated from the following files:

- Fl_Hor_Slider.H
- Fl_Slider.cxx

33.63 Fl_Hor_Value_Slider Class Reference

Inheritance diagram for Fl_Hor_Value_Slider:

```
Fl_Widget
   Fl_Valuator
      Fl_Slider
         Fl_Value_Slider
              Fl_Hor_Value_Slider
```

Public Member Functions

- Fl_Hor_Value_Slider (int X, int Y, int W, int H, const char *l=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Hor_Value_Slider.H
- Fl_Value_Slider.cxx

33.64 Fl_ICO_Image Class Reference

The Fl_ICO_Image class supports loading, caching, and drawing of Windows icon (.ico) files.

```
#include <Fl_ICO_Image.H>
```

Inheritance diagram for Fl_ICO_Image:

```
Fl_Image
   Fl_RGB_Image
      Fl_BMP_Image
         Fl_ICO_Image
```

Generated by Doxygen
Classes

- struct IconDirEntry
  
  Windows ICONDIRENTRY structure

Public Member Functions

- Fl ICO Image (const char *filename, int id=-1, const unsigned char *data=NULL, const size_t datasize=0)
  
  Loads the named icon image from the given .ico filename or from memory.

- const IconDirEntry * icondirentry () const
  
  Returns the array of idcount() loaded IconDirEntry structures.

- int idcount () const
  
  Returns the number of icons of various resolutions present in the ICO object.

- ~Fl ICO Image ()
  
  Destructor.

Additional Inherited Members

33.64.1 Detailed Description

The Fl ICO Image class supports loading, caching, and drawing of Windows icon (.ico) files.

33.64.2 Constructor & Destructor Documentation

33.64.2.1 Fl ICO Image()

Fl ICO Image::Fl ICO Image (
  const char * filename,
  int id = -1,
  const unsigned char * data = NULL,
  const size_t datasize = 0 )

Loads the named icon image from the given .ico filename or from memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name of a .ico file, or of the in-memory image</td>
</tr>
<tr>
<td>id</td>
<td>When id is -1 (default), the highest-resolution icon is loaded; when id 0, load the icon with this ID; when id = -2, load all IconDirEntry structures but no image.</td>
</tr>
<tr>
<td>data</td>
<td>NULL, or in-memory icon data</td>
</tr>
<tr>
<td>datasize</td>
<td>Size in bytes of the data byte array (used when data is not NULL)</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl ICO Image.H
- Fl ICO Image.cxx

33.65 Fl Image Class Reference

Base class for image caching, scaling and drawing.

#include <Fl_Image.H>

Inheritance diagram for Fl Image:
Public Member Functions

- virtual class Fl_Shared_Image * as_shared_image ()
  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
  Creates a copy of the image in the same size.
- virtual Fl_Image * copy (int W, int H) const
  Creates a resized copy of the image.
- int count () const
  Returns the number of data values associated with the image.
- int d () const
  Returns the image depth.
- const char const * data () const
  Returns a pointer to the current image data array.
- int data_h () const
  Returns the height of the image data.
- int data_w () const
  Returns the width of the image data.
- virtual void desaturate ()
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
  Draws the image to the current drawing surface.
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)
  Draws the image to the current drawing surface with a bounding box.
- int fail () const
  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)
  The constructor creates an empty image with the specified width, height, and depth.
- int h () const
  Returns the current image drawing height in FLTK units.
- void inactive ()
  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- virtual void label (Fl_Menu_Item *m)
  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget *w)
This method is an obsolete way to set the image attribute of a widget or menu item.

- **int ld () const**
  Returns the current line data size in bytes.

- **virtual void release ()**
  Releases an Fl_Image - the same as 'delete this'.

- **virtual void scale (int width, int height, int proportional=1, int can_expand=0)**
  Sets the drawing size of the image.

- **virtual void uncache ()**
  If the image has been cached for display, delete the cache data.

- **int w () const**
  Returns the current image drawing width in FLTK units.

- **virtual ~Fl_Image ()**
  The destructor is a virtual method that frees all memory used by the image.

**Static Public Member Functions**

- **static Fl_Labeltype define_FL_IMAGE_LABEL ()**
- **static Fl_RGB_Scaling RGB_scaling ()**
  Returns the currently used RGB image scaling method.

- **static void RGB_scaling (Fl_RGB_Scaling)**
  Sets the RGB image scaling method used for copy(int, int).

- **static Fl_RGB_Scaling scaling_algorithm ()**
  Gets what algorithm is used when resizing a source image to draw it.

- **static void scaling_algorithm (Fl_RGB_Scaling algorithm)**
  Sets what algorithm is used when resizing a source image to draw it.

**Static Public Attributes**

- **static const int ERR_FILE_ACCESS = -2**
- **static const int ERR_FORMAT = -3**
- **static const int ERR_MEMORY_ACCESS = -4**
- **static const int ERR_NO_IMAGE = -1**
- **static bool register_images_done = false**
  True after fl_register_images() was called, false before.

**Protected Member Functions**

- **void d (int D)**
  Sets the current image depth.

- **void data (const char *const *p, int c)**
  Sets the current data pointer and count of pointers in the array.

- **void draw_empty (int X, int Y)**
  The protected method draw_empty() draws a box with an X in it.

- **int draw_scaled (int X, int Y, int W, int H)**
  Draw the image to the current drawing surface rescaled to a given width and height.

- **void h (int H)**
  Sets the height of the image data.

- **void ld (int LD)**
  Sets the current line data size in bytes.

- **void w (int W)**
  Sets the width of the image data.
Static Protected Member Functions

- static void labeltype (const Fl_Label ∗lo, int 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

33.65.1 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.

33.65.2 Constructor & Destructor Documentation

33.65.2.1 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.

33.65.3 Member Function Documentation

33.65.3.1 as_shared_image()

virtual class Fl_Shared_Imag e ∗ Fl_Image::as_shared_image ( ) [inline], [virtual]

Returns whether an image is an Fl_Shared_Imag e or not.

This virtual method returns a pointer to an Fl_Shared_Imag e if this object is an instance of Fl_Shared_Imag e 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_Imag e.

Since

1.4.0

Reimplemented in Fl_Shared_Imag e.
33.65.3.2 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_Anim_GIF_Image`, `Fl_RGB_Image`, `Fl_Pixmap`, `Fl_Shared_Image`, `Fl_SVG_Image`, and `Fl_Tiled_Image`.

33.65.3.3 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() 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'.

33.65.3.4 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`, `Fl_Anim_GIF_Image` and `Fl_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `Fl_GIF_Image` is an object of class `Fl_Pixmap`.

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 \texttt{Fl\_Image} or any subclass your overridden methods of \texttt{Fl\_Image::copy() const} and \texttt{Fl\_Image::copy(int, int) const} \textbf{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 \texttt{Fl\_Anim\_GIF\_Image}, \texttt{Fl\_Bitmap}, \texttt{Fl\_RGB\_Image}, \texttt{Fl\_Pixmap}, \texttt{Fl\_Shared\_Image}, \texttt{Fl\_SVG\_Image}, and \texttt{Fl\_Tiled\_Image}.

33.65.3.5 \texttt{count()}

\begin{verbatim}
int Fl\_Image::count ( ) const [inline]
\end{verbatim}

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

\texttt{data()}

33.65.3.6 \texttt{d()}

\begin{verbatim}
int Fl\_Image::d ( ) const [inline]
\end{verbatim}

Returns the image depth. The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

33.65.3.7 \texttt{data()} [1/2]

\begin{verbatim}
const char *const * Fl\_Image::data ( ) const [inline]
\end{verbatim}

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 \texttt{count()} method to find the size of the data array. You must not dereference the \texttt{data()} pointer if \texttt{count()} equals zero.

Note

\texttt{data()} may return NULL.

Example:

\texttt{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 \texttt{data\_w()}, \texttt{data\_h()}, \texttt{d()} and \texttt{ld()} and is basically \texttt{data\_w()} * \texttt{data\_h()} * \texttt{d()} but there are exceptions if \texttt{ld()} is non-zero: see description of \texttt{ld()}. Since FLTK 1.4.0 \texttt{w()} and \texttt{h()} are no longer significant for the image data size if \texttt{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

\texttt{count()}, \texttt{w()}, \texttt{h()}, \texttt{data\_w()}, \texttt{data\_h()}, \texttt{d()}, \texttt{ld()}
33.65.3.8 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()

33.65.3.9 desaturate()

void Fl_Image::desaturate () [virtual]

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented in Fl_Anim_GIF_Image, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

33.65.3.10 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. |

33.65.3.11 draw() [2/2]

void Fl_Image::draw (int X, int Y, int W, int H, int cx = 0, int cy = 0) [virtual]

Draws the image to the current drawing surface with a bounding box.
Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.
In other words: fl_push_clip(X, Y, W, H) is applied, the image is drawn with its upper-left corner at X-cx, Y-cy and its own width and height; fl_pop_clip() is applied.
Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

33.65.3.12 draw_empty()

void Fl_Image::draw_empty (int X,
33.65 Fl_Image Class Reference

```cpp
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.

### 33.65.3.13 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**

<table>
<thead>
<tr>
<th>X, Y</th>
<th>position of the image's top-left</th>
</tr>
</thead>
<tbody>
<tr>
<td>W, H</td>
<td>width and height for the drawn image</td>
</tr>
</tbody>
</table>

**Returns**

1

**Deprecated** Only for API compatibility with FLTK 1.3.4.

### 33.65.3.14 fail()

```cpp
int Fl_Image::fail ( ) const
```
Returns a value that is not 0 if there is currently no image available.

**Example use:**
```
// [
Fl_Box box(X, Y, W, H);
Fl_JPEG_Image jpg("/tmp/foo.jpg");
switch (jpg.fail()) {
    case Fl_Image::ERR_NO_IMAGE:
    case Fl_Image::ERR_FILE_ACCESS:
        fl_alert("/tmp/foo.jpg: %s", strerror(errno)); // shows actual os error to user
        exit(1);
    case Fl_Image::ERR_FORMAT:
        fl_alert("/tmp/foo.jpg: couldn't decode image");
        exit(1);
    box.image(jpg);
}
```

**Returns**

Image load failure if non-zero

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the image was loaded successfully</td>
</tr>
<tr>
<td>ERR_NO_IMAGE</td>
<td>no image was found</td>
</tr>
<tr>
<td>ERR_FILE_ACCESS</td>
<td>there was a file access related error (errno should be set)</td>
</tr>
<tr>
<td>ERR_FORMAT</td>
<td>image decoding failed</td>
</tr>
<tr>
<td>ERR_MEMORY_ACCESS</td>
<td>image decoder tried to access memory outside of given memory block</td>
</tr>
</tbody>
</table>
33.65.3.15  h() [1/2]

    int Fl_Image::h ( ) const [inline]

Returns the current image drawing height in FLTK units.
The values of h() and data_h() are identical unless scale() has been called after which they may differ.

33.65.3.16  h() [2/2]

    void Fl_Image::h ( int H ) [inline], [protected]

Sets the height of the image data.
This protected function sets both image heights: the height of the image data returned by data_h() and the image
drawing height in FLTK units returned by h().

33.65.3.17  inactive()

    void Fl_Image::inactive ( ) [inline]

The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears
grayed out.
An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

    Note
    The RGB color of FL_BACKGROUND_COLOR may change when the connection to the display is made. See
    fl_open_display().

33.65.3.18  label() [1/2]

    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 Fl_Menu_Item::image() instead.

Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

33.65.3.19  label() [2/2]

    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 Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

33.65.3.20  ld() [1/2]

    int Fl_Image::ld ( ) const [inline]

Returns the current line data size in bytes.
See also
    ld(int)
33.65.3.21 ld() [2/2]

void Fl_Image::ld ( int LD ) [inline], [protected]
Sets the current line data size in bytes.
Color images may contain extra data (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 data_w() * d() bytes.
If LD is non-zero, then it must be positive and larger than data_w() * d() to account for the extra data per line.

33.65.3.22 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 operator delete since this is not appropriate for Fl_Shared_Image objects.
The virtual method release() handles this properly.
Since 1.4.0 in the base class Fl_Image and virtual in Fl_Shared_Image
Reimplemented in Fl_Shared_Image.

33.65.3.23 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.

33.65.3.24 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 |

Generated by Doxygen
**Note**

This function generally changes the values returned by the \( w() \) and \( h() \) member functions. In contrast, the values returned by \( \text{data}_w() \) and \( \text{data}_h() \) remain unchanged.

**Version**

1.4 (1.3.4 and FL_ABI_VERSION for \( \text{Fl}_\_\text{Shared}_\_\text{Image} \) only)

Example code: scale an image to fit in a box

```cpp
Fl_Box *b = ... // a box
Fl_Image *img = new Fl_PNG_Image("/path/to/picture.png"); // read a picture file
// 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
```

### 33.65.3.25 scaling_algorithm()

```cpp
static void Fl_Image::scaling_algorithm (Fl_RGB_Scaling algorithm) [inline], [static]
```

Sets what algorithm is used when resizing a source image to draw it. The default algorithm is \( \text{FL}_\_\text{RGB}_{-}\_\text{SCALING}_{-}\_\text{BILINEAR} \). Drawing an \( \text{Fl}_\_\text{Image} \) 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 \( \text{scale()} \). This function controls what method is used when the image to be resized is an \( \text{Fl}_\_\text{RGB}_\_\text{Image} \).

**Version**

1.4

### 33.65.3.26 uncache()

```cpp
void Fl_Image::uncache () [virtual]
```

If the image has been cached for display, delete the cache data. This allows you to change the data used for the image and then redraw it without recreating an image object. Reimplemented in \( \text{Fl}_\_\text{Anim}_\_\text{GIF}_\_\text{Image} \), \( \text{Fl}_\_\text{Bitmap} \), \( \text{Fl}_\_\text{RGB}_\_\text{Image} \), \( \text{Fl}_\_\text{Pixmap} \), and \( \text{Fl}_\_\text{Shared}_\_\text{Image} \).

### 33.65.3.27 w() [1/2]

```cpp
int Fl_Image::w () const [inline]
```

Returns the current image drawing width in FLTK units. The values of \( w() \) and \( \text{data}_w() \) are identical unless \( \text{scaled}() \) has been called after which they may differ.

### 33.65.3.28 w() [2/2]

```cpp
void Fl_Image::w (int W) [inline], [protected]
```

Sets the width of the image data. This protected function sets both image widths: the width of the image data returned by \( \text{data}_w() \) and the image drawing width in FLTK units returned by \( w() \).

The documentation for this class was generated from the following files:

- \( \text{Fl}_\_\text{Image}.\text{H} \)
- \( \text{Fl}_\_\text{Image}.\text{cxx} \)

### 33.66 Fl_Image_Reader Class Reference

**Public Member Functions**

- \( \text{int} \ \text{error()} \ \text{const} \)
The documentation for this class was generated from the following files:

- Fl_Image_Reader.h
- Fl_Image_Reader.cxx
Protected Member Functions

- void translate (int x, int y) FL_OVERRIDE
  Translates the current graphics origin accounting for the current rotation.
- void untranslate () FL_OVERRIDE
  Undoes the effect of a previous translate() call.

Friends

- class Fl_Graphics_Driver

Additional Inherited Members

33.67.1 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_Imag* 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;
```

33.67.2 Constructor & Destructor Documentation

33.67.2.1 Fl_Image_Surface()

Fl_Image_Surface::Fl_Image_Surface {
  int w,
  int h,
  int high_res = 0,
  Fl_Offscreen off = 0 )

Constructor with optional high resolution.

Parameters

| w,h | Width and height of the resulting image. The value of the high_res parameter controls whether w and h are interpreted as pixels or FLTK units. |
| high_res | If zero, the created image surface is sized at w 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. |
| off | If not null, the image surface is constructed around a pre-existing Fl_Offscreen. The caller is responsible for both construction and destruction of this Fl_Offscreen object. It is mostly intended for internal use by FLTK. |
33.67.3 Member Function Documentation

33.67.3.1 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.

33.67.3.2 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.

33.67.3.3 is_current()

```
bool Fl_Image_Surface::is_current ( ) [virtual]
```

Is this surface the current drawing surface? Reimplemented from Fl_Surface_Device.

33.67.3.4 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.

33.67.3.5 origin() [1/2]

```
void Fl_Image_Surface::origin ( int * x, int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**

| out | x, y | If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.

33.67.3.6 origin() [2/2]

```
void Fl_Image_Surface::origin ( (Generated by Doxygen)
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`.

### 33.67.3.7 printable_rect()

```c
int Fl_Image_Surface::printable_rect ( 
    int * w, 
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**

- 0 if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`.

### 33.67.3.8 rescale()

```c
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

### 33.67.3.9 set_current()

```c
void Fl_Image_Surface::set_current ( 
    void ) [virtual]
```

Make this surface the current drawing surface. This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current()` / `Fl_Surface_Device::pop_current()`.

**Note**

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
• make the display current again with `Fl_Display_Device::display_device() -> set_current();`. Don’t do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

Reimplemented from `Fl_Surface_Device`.

### 33.67.3.10 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`.

### 33.67.3.11 untranslate()

```cpp
void Fl_Image_Surface::untranslate (  
    void );  [protected],  [virtual]
```

Undoes the effect of a previous `translate()` call.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following files:

• `Fl_Image_Surface.H`
• `Fl_Image_Surface.cxx`

### 33.68 Fl_Input Class Reference

This is the FLTK text input widget.

```cpp
#include <Fl_Input.H>
```

Inheritance diagram for Fl_Input:

```
Fl_Widget
   |   |
   v   v
Fl_Widget
   |   |
   v   v
Fl_Input
```

#### Public Member Functions

- `Fl_Input (int, int, int, int, const char * = 0)`

  Creates a new `Fl_Input` widget using the given position, size, and label string.

- `int handle (int) FL_OVERRIDE`

  Handles the specified event.

#### Protected Member Functions

- `void draw () FL_OVERRIDE`

  Draws the widget.

- `int handle_key ()`

  Handles a keystroke.
Friends

- `class Fl_Cocoa_Screen_Driver`
- `class Fl_Screen_Driver`

Additional Inherited Members

### 33.68.1 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>Table 33.260 Keyboard and mouse bindings.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mouse button 1</strong></td>
</tr>
<tr>
<td><strong>Mouse button 2</strong></td>
</tr>
<tr>
<td><strong>Mouse button 3</strong></td>
</tr>
<tr>
<td><strong>Backspace</strong></td>
</tr>
<tr>
<td><strong>Delete</strong></td>
</tr>
<tr>
<td><strong>Enter</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 33.261 Platform specific keyboard bindings.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows/Linux</strong></td>
</tr>
<tr>
<td>^A</td>
</tr>
<tr>
<td>^C</td>
</tr>
<tr>
<td>^I</td>
</tr>
<tr>
<td>^J</td>
</tr>
<tr>
<td>^L</td>
</tr>
<tr>
<td>^M</td>
</tr>
<tr>
<td>^V, Shift-Insert</td>
</tr>
<tr>
<td>^X, Shift-Delete</td>
</tr>
<tr>
<td>^Z</td>
</tr>
</tbody>
</table>
33.68 Fl_Input Class Reference

### Shift-^Z
- **Shift-Command-Z**: Redo.
  - Currently same behavior as ^Z. Reserved for future multilevel undo/redo.

### Arrow Keys
- **Arrow Keys**: Standard cursor movement.
  - Can be combined with Shift to extend selection.

### Home
- **Command-Up, Command-Left**: Move to start of line.
  - Can be combined with Shift to extend selection.

### End
- **Command-Down, Command-Right**: Move to end of line.
  - Can be combined with Shift to extend selection.

### Ctrl-Home
- **Command-Up, Command-PgUp, Ctrl-Left**: Move to top of document/field.
  - In single line input, moves to start of line. In multiline input, moves to start of top line. Can be combined with Shift to extend selection.

### Ctrl-End
- **Command-End, Command-PgDn, Ctrl-Right**: Move to bottom of document/field.
  - In single line input, moves to end of line. In multiline input, moves to end of last line. Can be combined with Shift to extend selection.

### Ctrl-Left
- **Alt-Left**: Word left.
  - Can be combined with Shift to extend selection.

### Ctrl-Right
- **Alt-Right**: Word right.
  - Can be combined with Shift to extend selection.

### Ctrl-Backspace
- **Alt-Backspace**: Delete word left.

### Ctrl-Delete
- **Alt-Delete**: Delete word right.

### 33.68.2 Constructor & Destructor Documentation

#### 33.68.2.1 Fl_Input()

```cpp
class Fl_Input
{
public:
    Fl_Input();
    Fl_Input(int x, int y, int w, int h);
    Fl_Input(int x, int y, int w, int h, const char *l = 0);

private:
    Fl_Input(const Fl_Input &); // Copy constructor.
};

Fl_Input::Fl_Input()
{
    // Implementation...
}
```

Creates a new Fl_Input widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX.

### 33.68.3 Member Function Documentation

#### 33.68.3.1 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
```

Generated by Doxygen
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

### 33.68.3.2 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**

<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.
Reimplemented in Fl_Spinner::Fl_Spinner_Input, and Fl_Secret_Input.

### 33.68.3.3 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

### 33.69 Fl_Input_ Class Reference

This class provides a low-overhead text input field.

```cpp
#include <Fl_Input_.H>
```

Inheritance diagram for Fl_Input_:
Public Member Functions

- int **append**(const char *t, int l=0, char keep_selection=0)
  
  Append text at the end.

- bool **can_redo**() const
  
  Check if there is a redo action available.

- bool **can_undo**() const
  
  Check if the last operation can be undone.

- int **copy**(int clipboard)
  
  Put the current selection into the clipboard.

- int **copy_cuts**()
  
  Copies the yank buffer to the clipboard.

- **Fl_Color cursor_color**() const
  
  Gets the color of the cursor.

- void **cursor_color**(Fl_Color n)
  
  Sets the color of the cursor.

- int **cut**()
  
  Deletes the current selection.

- int **cut**(int a, int b)
  
  Deletes all characters between index a and b.

- int **cut**(int n)
  
  Deletes the next n bytes rounded to characters before or after the cursor.

- **Fl_Input_(int, int, int, int, const char *=0)**
  
  Creates a new Fl_Input_ widget.

- unsigned int **index**(int i) const
  
  Returns the character at index i.

- int **input_type**() const
  
  Gets the input field type.

- void **input_type**(int t)
  
  Sets the input field type.

- int **insert**(const char *t, int l=0)
  
  Inserts text at the cursor position.

- int **insert_position**() const
  
  Gets the position of the text cursor.

- int **insert_position**(int p)
  
  Sets the cursor position and mark.

- int **insert_position**(int p, int m)
  
  Sets the index for the cursor and mark.

- int **mark**() const
  
  Gets the current selection mark.

- int **mark**(int m)
  
  Sets the current selection mark.

- int **maximum_size**() const
  
  Gets the maximum length of the input field in characters.

- void **maximum_size**(int m)
Sets the maximum length of the input field in characters.

- `int position () const`
- `int position (int p)`
- `int position (int p, int m)`
- `int readonly () const`
  Gets the read-only state of the input field.
- `void readonly (int b)`
  Sets the read-only state of the input field.
- `int redo ()`
  Redo previous undo operation.
- `int replace (int b, int e, const char ∗text, int ilen=0)`
  Deletes text from b to e and inserts the new string text.
- `void resize (int, int, int, int) FL_OVERRIDE`
  Changes the size of the widget.
- `int shortcut () const`
  Return the shortcut key associated with this widget.
- `void shortcut (int s)`
  Sets the shortcut key associated with this widget.
- `int size () const`
  Returns the number of bytes in value().
- `void size (int W, int H)`
  Sets the width and height of this widget.
- `int static_value (const char ∗)`
  Changes the widget text.
- `int static_value (const char ∗, int)`
  Changes the widget text.
- `int tab_nav () const`
  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- `void tab_nav (int val)`
  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- `Fl_Color textcolor () const`
  Gets the color of the text in the input field.
- `void textcolor (Fl_Color n)`
  Sets the color of the text in the input field.
- `Fl_Font textfont () const`
  Gets the font of the text in the input field.
- `void textfont (Fl_Font s)`
  Sets the font of the text in the input field.
- `Fl_Fontsize textsize () const`
  Gets the size of the text in the input field.
- `void textsize (Fl_Fontsize s)`
  Sets the size of the text in the input field.
- `int undo ()`
  Undoes previous changes to the text buffer.
- `const char ∗value () const`
  Returns the text displayed in the widget.
- `int value (const char ∗)`
  Changes the widget text.
- `int value (const char ∗, int)`
  Changes the widget text.
- `int 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

- int apply_undo()
  Apply the current undo/redo operation.

- void drawtext (int, int, int, int)
  Draws the text in the passed bounding box.

- void drawtext (int, int, int, int, bool draw_active)
  Draws the text in the passed bounding box.

- void handle_mouse (int, int, int, int, int keepmark=0)
  Handles mouse clicks and mouse moves.

- int handletextarea (int e, int, int, int)
  Handles all kinds of text field related events.

- int line_end (int i) const
  Finds the end of a line.

- int line_start (int i) const
  Finds the start of a line.

- int linesPerPage ()

- void maybe_do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

- int up_down_position (int, int keepmark=0)
  Moves the cursor to the column given by up_down_pos.

- int word_end (int i) const
  Finds the end of a word.

- int word_start (int i) const
  Finds the start of a word.

- int xscroll () const

- int yscroll () const

- void yscroll (int yOffset)

Additional Inherited Members

33.69.1 Detailed Description

This class provides a low-overhead text input field. This is a virtual base class below Fl_Input. It has all the same interfaces, but lacks the handle() and draw() method. You may want to subclass it if you are one of those people who likes to change how the editing keys work. It may also be useful for adding scrollbars to the input field.

This can act like any of the subclasses of Fl_Input, by setting type() to one of the following values:

- #define FL_NORMAL_INPUT  0
- #define FL_FLOAT_INPUT   1
- #define FL_INT_INPUT     2
- #define FL_MULTILINE_INPUT 4
- #define FL_SECRET_INPUT  5
- #define FL_INPUT_TYPE    7
- #define FL_INPUT_READONLY 8
- #define FL_NORMAL_OUTPUT (FL_NORMAL_INPUT | FL_INPUT_READONLY)
- #define FL_MULTILINE_OUTPUT (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
- #define FL_INPUT_WRAP 16
- #define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_WRAP)
- #define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)

All variables that represent an index into a text buffer are byte-oriented, not character oriented, counting from 0 (at or before the first character) to size() (at the end of the buffer, after the last byte). Since UTF-8 characters can be up to six bytes long, simply incrementing such an index will not reliably advance to the next character in the text buffer. Indices and pointers into the text buffer should always point at a 7 bit ASCII character or the beginning of a UTF-8 character sequence. Behavior for false UTF-8 sequences and pointers into the middle of a sequence are undefined.
33.69.2 Constructor & Destructor Documentation

33.69.2.1 Fl_Input()

Fl_Input_::Fl_Input_ ( int X, int Y, int W, int H, const char ∗ l = 0 )

Creates a new Fl_Input_ widget.
This function creates a new Fl_Input_ widget and adds it to the current Fl_Group. The value() is set to NULL. The default boxtype is FL_DOWN_BOX.

Parameters

| X, Y, W, H | the dimensions of the new widget |
| l           | an optional label text          |

33.69.2.2 ∼Fl_Input()

Fl_Input_::∼Fl_Input_ ( )

Destroys the widget.
The destructor clears all allocated buffers and removes the widget from the parent Fl_Group.

33.69.3 Member Function Documentation

33.69.3.1 append()

int Fl_Input_::append ( const char ∗ t, int l = 0, char keep_selection = 0 )

Append text at the end.
This function appends the string in t to the end of the text. It does not moves the new position or mark.

Parameters

| in t | text that will be appended |
| l | length of text, or 0 if the string is terminated by nul. |
| keep_selection | if this is 1, the current text selection will remain, if 0, the cursor will move to the end of the inserted text. |

Returns

0 if no text was appended
33.69.3.2  apply_undo()

int Fl_Input_::apply_undo ( ) [protected]
Apply the current undo/redo operation.
It's up to undo() and redo() to push and pop actions to and from the lists.
Returns
1 if the current action changed any text.

See also
undo(), redo()

33.69.3.3  can_redo()

bool Fl_Input_::can_redo ( ) const
Check if there is a redo action available.
Returns
true if the widget can redo the last undo action

33.69.3.4  can_undo()

bool Fl_Input_::can_undo ( ) const
Check if the last operation can be undone.
Returns
true if the widget can undo the last change

33.69.3.5  copy()

int Fl_Input_::copy ( int clipboard )
Put the current selection into the clipboard.
This function copies the current selection between mark() and position() into the specified clipboard. This does not replace the old clipboard contents if position() and mark() are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.
Parameters
clipboard the clipboard destination 0 or 1
Returns
0 if no text is selected, 1 if the selection was copied
See also
Fl::copy(const char *, int, int)

33.69.3.6  copy_cuts()

int Fl_Input_::copy_cuts ( )
Copies the yank buffer to the clipboard. This method copies all the previous contiguous cuts from the undo information to the clipboard. This function implements the \^K shortcut key.

Returns
0 if the operation did not change the clipboard

See also
\texttt{copy(int)}, \texttt{cut()}

### 33.69.3.7 cursor_color() [1/2]

\texttt{Fl\_Color Fl\_Input\_::cursor\_color ( ) const [inline]}

Gets the color of the cursor.

Returns
the current cursor color

### 33.69.3.8 cursor_color() [2/2]

\texttt{void Fl\_Input\_::cursor\_color ( Fl\_Color n ) [inline]}

Sets the color of the cursor.
The default color for the cursor is \texttt{FL\_BLACK}.

Parameters
- \texttt{n} the new cursor color

### 33.69.3.9 cut() [1/3]

\texttt{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 \texttt{copy()} first or \texttt{copy_cuts()} after this call.

Returns
0 if no data was copied

### 33.69.3.10 cut() [2/3]

\texttt{int Fl\_Input\_::cut ( int a, int b ) [inline]}

Deletes all characters between index \texttt{a} and \texttt{b}. This function deletes the currently selected text without storing it in the clipboard. To use the clipboard, you may call \texttt{copy()} first or \texttt{copy_cuts()} after this call.

Parameters
- \texttt{a,b} range of bytes rounded to full characters and clamped to the buffer
33.69.3.11 cut() [3/3]

```cpp
int Fl_Input_::cut ( int n ) [inline]
```

Deletes the next \( n \) bytes rounded to characters before or after the cursor.
This function deletes the currently selected text \textit{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

33.69.3.12 drawtext() [1/2]

```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 |

33.69.3.13 drawtext() [2/2]

```cpp
void Fl_Input_::drawtext ( int X, int Y, int W, int H, bool draw_active ) [protected]
```

Draws the text in the passed bounding box.
This version of `drawtext` allows the user to control whether the widget is drawn as active, i.e. with the text cursor, or inactive. This is useful for compound widgets where the input should be shown as active when actually the container widget is the active one.
A caller should not draw the widget with `active` set if another text widget may indeed be the active widget.

**Parameters**

| \( X,Y,W,H \) | area that must be redrawn |
| `draw_active` | if set, the cursor will be drawn, even if the widget is not active |
See also

   Fl_Input_::drawtext(int X, int Y, int W, int H)

### 33.69.3.14 handle_mouse()

```c
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

### 33.69.3.15 handletext()

```c
int Fl_Input_::handletext (  
   int event,  
   int X,  
   int Y,  
   int W,  
   int H ) [protected]
```

Handles all kinds of text field related events.
This is called by derived classes.

**Todo** Add comment and parameters

### 33.69.3.16 index()

```c
unsigned int Fl_Input_::index (  
   int i ) const
```

Returns the character at index i.
This function returns the UTF-8 character at i as a ucs4 character code.

**Parameters**

| in | i | index into the value field |

**Returns**

the character at index i

### 33.69.3.17 input_type() [1/2]

```c
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.

**Returns**

the current input type
33.69.3.18  input_type() [2/2]

void Fl_Input_::input_type (  
    int t  ) [inline]

Sets the input field type.  
A redraw() is required to reformat the input field.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>new input type</th>
</tr>
</thead>
</table>

33.69.3.19  insert()

int Fl_Input_::insert (  
    const char * t,  
    int l = 0  ) [inline]

Inserts text at the cursor position.  
This function inserts the string in t at the cursor position() and moves the new position and mark to the end of the inserted text.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>text that will be inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>l</td>
<td>length of text, or 0 if the string is terminated by null.</td>
</tr>
</tbody>
</table>

Returns

0 if no text was inserted

33.69.3.20  insert_position() [1/3]

int Fl_Input_::insert_position ( ) const [inline]

Gets the position of the text cursor.

Returns

the cursor position as an index in the range 0..size()

See also

insert_position(int, int)

33.69.3.21  insert_position() [2/3]

int Fl_Input_::insert_position (  
    int p  ) [inline]

Sets the cursor position and mark.  
position(n) is the same as position(n, n).

Parameters

<table>
<thead>
<tr>
<th>p</th>
<th>new index for cursor and mark</th>
</tr>
</thead>
</table>
Returns

0 if no positions changed

See also

insert_position(int, int), insert_position(), mark(int)

33.69.3.22 insert_position() [3/3]

int Fl_Input_::insert_position (
    int p,
    int m )

Sets the index for the cursor and mark.
The input widget maintains two pointers into the string. The position (p) is where the cursor is. The mark (m) is the other end of the selected text. If they are equal then there is no selection. Changing this does not affect the clipboard (use copy() to do that). Changing these values causes a redraw(). The new values are bounds checked.

Parameters

| p | index for the cursor position |
| m | index for the mark |

Returns

0 if no positions changed

See also

position(int), position(), mark(int)

33.69.3.23 line_end()

int Fl_Input_::line_end ( const [protected]
    int i )

Finds the end of a line.
This call calculates the end of a line based on the given index i.

Parameters

| in | i | starting index for the search |

Returns

end of the line

33.69.3.24 line_start()

int Fl_Input_::line_start ( const [protected]
    int i )

Finds the start of a line.
This call calculates the start of a line based on the given index i.
Parameters

\[
\begin{array}{c|c}
\text{in} & \text{i} \\
\hline
\text{starting index for the search} & \\
\end{array}
\]

Returns

\[
\text{start of the line}
\]

33.69.3.25 \textbf{mark} [1/2]

int Fl_Input_::mark () const [inline]

Gets the current selection mark.

Returns

\[
\text{index into the text}
\]

33.69.3.26 \textbf{mark} [2/2]

int Fl_Input_::mark (int \text{m}) [inline]

Sets the current selection mark.
mark(n) is the same as insertion(insertation(),n).

Parameters

\[
\begin{array}{c|c}
\text{m} & \text{new index of the mark} \\
\end{array}
\]

Returns

\[
0 \text{ if the mark did not change}
\]

See also

\[
\text{insert_position(), insert_position(int, int)}
\]

33.69.3.27 \textbf{maximum_size} [1/2]

int Fl_Input_::maximum_size () const [inline]

Gets the maximum length of the input field in characters.

See also

\[
\text{maximum_size(int)}
\]

33.69.3.28 \textbf{maximum_size} [2/2]

void Fl_Input_::maximum_size (int \text{m}) [inline]

Sets the maximum length of the input field in characters.
This limits the number of \textbf{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).
33.69.3.29  position() [1/3]

int Fl_Input_::position ( ) const [inline]

*Deprecated* "in 1.4.0 - use insert_position() instead"

33.69.3.30  position() [2/3]

int Fl_Input_::position ( int p ) [inline]

*Deprecated* "in 1.4.0 - use insert_position(p) instead"

33.69.3.31  position() [3/3]

int Fl_Input_::position ( int p, int m ) [inline]

*Deprecated* "in 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

33.69.3.32  readonly() [1/2]

int Fl_Input_::readonly ( ) const [inline]

Gets the read-only state of the input field.

Returns:
non-zero if this widget is read-only

33.69.3.33  readonly() [2/2]

void Fl_Input_::readonly ( int b ) [inline]

Sets the read-only state of the input field.

Parameters:

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>if b is 0, the text in this widget can be edited by the user</td>
<td></td>
</tr>
</tbody>
</table>

33.69.3.34  redo()

int Fl_Input_::redo ( )

Redo previous undo operation.
This call reapsplies previously executed undo operations.
Returns
non-zero if any change was made.

33.69.3.35  replace()

int Fl_Input_::replace {
    int b,
    int e,
    const char * text,
    int ilen = 0
}

Deletes text from \( b \) to \( e \) and inserts the new string \( \text{text} \).
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 \( \text{text} \) 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 \( \text{text} \) 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>( b )</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>( e )</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>( \text{text} )</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ilen</strong></td>
</tr>
</tbody>
</table>

Returns
0 if nothing changed

Note
If \( \text{text} \) 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).

33.69.3.36  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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>( X,Y,W,H )</strong></td>
</tr>
</tbody>
</table>
See also

```cpp
Fl_Widget::resize(int, int, int, int)
```
Reimplemented from `Fl_Widget`.

### 33.69.3.37 shortcut() [1/2]

```cpp
int Fl_Input_::shortcut () const [inline]
```
Return the shortcut key associated with this widget.

Returns

- shortcut keystroke

See also

```cpp
Fl_Button::shortcut()
```

### 33.69.3.38 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

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>new shortcut keystroke</th>
</tr>
</thead>
</table>

See also

```cpp
Fl_Button::shortcut()
```

### 33.69.3.39 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

### 33.69.3.40 size() [2/2]

```cpp
void Fl_Input_::size (int W, int H) [inline]
```
Sets the width and height of this widget.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>W,H</th>
<th>new width and height</th>
</tr>
</thead>
</table>
See also

`Fl_Widget::size(int, int)`

### 33.69.3.41 static_value() [1/2]

```cpp
int Fl_Input_::static_value (const char ∗ str)
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the `Fl_Input` is destroyed or `value()` is called again.

**Parameters**

- `in str` the new text

**Returns**

- non-zero if the new value is different than the current one

### 33.69.3.42 static_value() [2/2]

```cpp
int Fl_Input_::static_value (const char ∗ str, int len)
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is not copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the `Fl_Input` is destroyed or `value()` is called again.

You can use the `len` parameter to directly set the length if you know it already or want to put null characters in the text.

**Parameters**

- `in str` the new text
- `in len` the length of the new text

**Returns**

- non-zero if the new value is different than the current one

### 33.69.3.43 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`. 

33.69.3.44 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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If val is 1, Tab advances focus (default).</td>
</tr>
<tr>
<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`. 

33.69.3.45 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)`

33.69.3.46 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()`
### 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

### 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.

Parameters

- **in s** the new text font

### textsize() [1/2]

```cpp
Fl_Fontsize Fl_Input_::textsize ( ) const [inline]
```

Gets the size of the text in the input field.

Returns

the text height in pixels

### textsize() [2/2]

```cpp
void Fl_Input_::textsize ( Fl_Fontsize s ) [inline]
```

Sets the size of the text in the input field.
The text height defaults to FL_NORMAL_SIZE.

Parameters

- **in s** the new font height in pixel units

### undo()

```cpp
int Fl_Input_::undo ( )
```

Undoes previous changes to the text buffer.
This call undoes a number of previous calls to replace().

Returns

non-zero if any change was made.

### up_down_position()

```cpp
int Fl_Input_::up_down_position ( )
```
int i,
int keepmark = 0 ) [protected]

Moves the cursor to the column given by up_down_pos.
This function is helpful when implementing up and down cursor movement. It moves the cursor from the beginning
of a line to the column indicated by the global variable up_down_pos in pixel units.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>index into the beginning of a line of text</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>keepmark</td>
<td>if set, move only the cursor, but not the mark</td>
</tr>
</tbody>
</table>

Returns

index to new cursor position

33.69.3.53 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*)

33.69.3.54 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()

33.69.3.55 value() [3/3]

int Fl_Input_::value ( const char * str,
int len )

Generated by Doxygen
Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal
buffer. Passing NULL is the same as "". You can use the length parameter to directly set the length if you know it already or want to put null characters
in the text.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>str</td>
<td>the new text</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>the length of the new text</td>
</tr>
</tbody>
</table>

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char* str), Fl_Input_::value()

33.69.3.56 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i</td>
<td>starting index for the search</td>
</tr>
</tbody>
</table>

Returns

end of the word

33.69.3.57 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i</td>
<td>starting index for the search</td>
</tr>
</tbody>
</table>

Returns

start of the word, or previous word
33.69.3.58 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.

33.69.3.59 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

33.70 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) FL_OVERRIDE
  
  Resizes the Fl_Input_Choice widget.

• void **set_changed** ()
  
  Sets the changed() state of both input and menu button widgets to the specified value.

• Fl_Color **textcolor** () const
  
  Gets the Fl_Input text field's text color.

• void **textcolor** (Fl_Color c)
  
  Sets the Fl_Input text field's text color to 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

#### 33.70.1 Detailed Description

A combination of the input widget and a menu button.

The user can either type into the input area, or use the menu button chooser on the right to choose an item which loads the input area with the selected text.

The application can directly access both the internal `FL_Input` and `FL_Menu_Button` widgets respectively using the `input()` and `menubutton()` accessor methods.

The default behavior is to invoke the `FL_Input_Choice::callback()` if the user changes the input field's contents, either by typing, pasting, or clicking a different item in the choice menu.

The callback can determine if an item was picked vs. typing into the input field by checking the value of `menubutton() -> changed()`, which will be:

- 1: the user picked a different item in the choice menu
- 0: the user typed or pasted directly into the input field

#### Example Use of `FL_Input_Choice`

```cpp
#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    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:

```cpp
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Input_Choice.H>

class MyInputChoice : public FL_Input_Choice {
protected:
```
33.70 Fl_Input_Choice Class Reference

```cpp
virtual int inp_x() const { return x() + Fl::box_dx(box()); } // 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();
}
```

33.70.2 Constructor & Destructor Documentation

### Fl_Input_Choice()

```cpp
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.

33.70.3 Member Function Documentation

#### add()

```cpp
void Fl_Input_Choice::add (  
    const char * s ) [inline]
```

Adds an item to the menu. When any item is selected, the Fl_Input_Choice callback() is invoked, which can do something with the selected item.

You can access the more complex Fl_Menu_Button::add() methods (setting item-specific callbacks, userdata, etc), via menubutton(). Example:

```cpp
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Fonts");
Fl_Menu_Button *mb = choice->menubutton(); // use Fl_Input_Choice’s Fl_Menu_Button
mb->add("Helvetica", 0, MyFont_CB, (void*)mydata); // use Fl_Menu_Button’s add() methods
mb->add("Courier", 0, MyFont_CB, (void*)mydata);
mb->add("More..", 0, FontDialog_CB, (void*)mydata);
```

#### inp_x()

```cpp
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.

#### input()

```cpp
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.

Generated by Doxygen
33.70.3.4 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.

33.70.3.5 menubutton()

Fl_Menu_Button ∗ Fl_Input_Choice::menubutton ( ) [inline]
Returns a pointer to the internal Fl_Menu_Button widget.
This can be used to access any of the methods of the menu button, e.g.
Fl_Input_Choice ∗ choice = new Fl_Input_Choice(100,10,120,25,"Choice:);
[...
// Print all the items in the choice menu
for ( int t=0; t<choice->menubutton()->size(); t++ ) {
    const Fl_Menu_Item &item = choice->menubutton()->menu()[t];
    printf("item %d -- label=%s\n", t, item.label() ? item.label() : "(Null)" );
}

33.70.3.6 resize()

void Fl_Input_Choice::resize ( int X, int Y, int W, int H ) [virtual]
Resizes the Fl_Input_Choice widget.
Reimplemented from Fl_Group.

33.70.3.7 update_menubutton()

int Fl_Input_Choice::update_menubutton ( )
Updates the menubutton with the string value in Fl_Input.
If the string value currently in Fl_Input matches one of the menu items in menubutton(), that menu item will become the current item selected.
Call this method after setting value(const char *) if you need the menubutton() to be synchronized with the Fl_Input field.
// Add items
choice->add(".25");
choice->add(".50");
choice->add("1.0");
choice->add("2.0");
choice->add("4.0");
choice->value("1.0"); // sets Fl_Input to "1.0"
choice->update_menubutton(); // cause menubutton to reflect this value too
// (returns 1 if match was found, 0 if not)
// Verify menubutton(‘s value.
printf("menu button choice index=%d, value=%s\n",
    choice->menubutton()->value(), // would be -1 if update not done
    choice->menubutton()->text() ); // would be NULL if update not done

Returns
1 if a matching menuitem was found and value set, 0 if not.

Version
1.4.0
33.70.3.8 value() [1/2]

void Fl_Input_Choice::value ( const char * val ) [inline]

Sets the Fl_Input text field's contents to val.
Note it is possible to set the value() to one that is not in the menubutton's list of choices.
Setting the value() does NOT affect the menubutton's selection. If that's needed, call update_menubutton() after setting value().

See also

void value(int val), update_menubutton()

33.70.3.9 value() [2/2]

void Fl_Input_Choice::value ( int val )

Chooses item# val in the menu, and sets the Fl_Input text field to that value.
Any previous text is cleared.

See also

void value(const char *val)

The documentation for this class was generated from the following files:

- Fl_Input_Choice.H
- Fl_Input_Choice.cxx

33.71 Fl_Int_Input Class Reference

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xabcdef).
#include <Fl_Int_Input.H>

Inheritance diagram for Fl_Int_Input:

```
FL_Widget
  FL_Input_
    FL_Input
      FL_Int_Input
```

Public Member Functions

- Fl_Int_Input (int X, int Y, int W, int H, const char *l=0)
  Creates a new Fl_Int_Input widget using the given position, size, and label string.

Additional Inherited Members

33.71.1 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 0xabcdef).
33.71.2 Constructor & Destructor Documentation

33.71.2.1 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

33.72 Fl_Int_Vector Class Reference

An STL-ish vector without templates.  
#include <Fl_Int_Vector.H>

Public Member Functions

- int back () const  
    Return the last element in the array.
- bool empty () const  
    Checks if array has no elements.
- Fl_Int_Vector ()  
    Create an empty vector of integers.
- Fl_Int_Vector (Fl_Int_Vector &o)  
    Copy constructor.
- Fl_Int_Vector & operator= (Fl_Int_Vector &o)  
    Assignment operator.
- int & operator[ ] (int x)  
    Access the specified integer element at index position x as a reference.
- int operator[ ] (int x) const  
    Access the specified integer element at index position x.
- int pop_back ()  
    Removes the last element the last element and returns its value.
- void push_back (int val)  
    Appends val to the array, enlarging the array by one.
- unsigned int size () const  
    Return the number of integer elements in the array.
- void size (unsigned int count)  
    Set the size of the array to count.
- ~Fl_Int_Vector ()  
    Destructor - frees the internal array and destroys the class.
33.72.1  Detailed Description

An STL-ish vector without templates.
Handles dynamic memory management of an integer array, and allows array elements to be accessed with zero
based indexing: v[0], v[1],...

Common use:
#include <stdio.h>
#include <FL/Fl_Int_Vector.H>
int main() {
    Fl_Int_Vector v;
    // Create an array of values 11,22,33:
    v.push_back(11); // add first element
    v.push_back(22); // add second element
    v.push_back(33); // add third element
    // Assignment by subscript
    v[1] = 222; // changes 2nd element from 22 to 222
    // Loop through printing the values
    for ( unsigned int i=0; i<v.size(); i++ )
        printf("%d ", v[i]); // access the elements
    printf("\n");
    // Clear the array
    v.size(0);
}

Todo  
• Add other std::vector methods like erase(), etc.
• Make memory blocking size flexible, and add related methods like capacity(), reserve(), shrink_to_fit(),
etc.
• Add non-std methods that are nevertheless needed, e.g. insert(index,val), delete(index), delete(start,
end), swap(a_idx,b_idx)

33.72.2  Member Function Documentation

33.72.2.1  back()

int Fl_Int_Vector::back ( ) const [inline]
Return the last element in the array.
Warning

You must not call back() if the array is empty, i.e. if (size() == 0).
Todo  Internals should maybe assert(size_ != 0)

33.72.2.2  empty()

bool Fl_Int_Vector::empty ( ) const [inline]
Checks if array has no elements.
Same as a test for (size() == 0).

33.72.2.3  operator=()

Fl_Int_Vector & Fl_Int_Vector::operator= ( 
    Fl_Int_Vector & o ) [inline]
Assignment operator.
Similar to the copy constructor, creates a separate copy of the source array, freeing any previous contents in the
current integer array.

33.72.2.4  operator[](1/2)

int & Fl_Int_Vector::operator[] ( 
    int x ) [inline]
Access the specified integer element at index position x as a reference.
This allows assignment by index through the returned reference, e.g. arr[1] = 222; where arr[1] ends up being a
reference to ptr[1], and then 222 is assigned to that ref.
Warning

No range checking is done on \( x \), which must be less than \( \text{size}() \).

33.72.5  \texttt{operator[]} (2/2)

\begin{verbatim}
int Fl_Int_Vector::operator[] ( int x ) const [inline]
Access the specified integer element at index position \( x \).
\end{verbatim}

Warning

No range checking is done on \( x \), which must be less than \( \text{size}() \).

33.72.6  \texttt{pop_back()}

\begin{verbatim}
int Fl_Int_Vector::pop_back () [inline]
Removes the last element the last element and returns its value.
\end{verbatim}

Warning

You must not call \texttt{pop_back()} if the array is empty, i.e. if \( \text{size}() == 0 \).

Todo  Internals should maybe assert(size_ != 0)

33.72.7  \texttt{size()}

\begin{verbatim}
void Fl_Int_Vector::size ( unsigned int count )
Set the size of the array to \( \text{count} \).
Setting size to zero clears the array and frees any memory it used.
Shrinking truncates the array and frees memory of truncated elements. Enlarging creates new elements that are
zero in value.
The documentation for this class was generated from the following files:
\begin{itemize}
  \item Fl_Int_Vector.H
  \item Fl_Int_Vector.cxx
\end{itemize}
\end{verbatim}

33.73  \texttt{Fl_JPEG_Image} Class Reference

The \texttt{Fl_JPEG_Image} class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG)
File Interchange Format (JFIF) images.

\texttt{\#include <Fl_JPEG_Image.H>}

Inheritance diagram for \texttt{Fl_JPEG_Image}:

\begin{verbatim}
Fl_Image

Fl_RGB_Image

Fl_JPEG_Image
\end{verbatim}
Public Member Functions

- `Fl_JPEG_Image` (const char *filename)
  The constructor loads the JPEG image from the given jpeg filename.

- `Fl_JPEG_Image` (const char *name, const unsigned char *data, int data_length=-1)
  The constructor loads the JPEG image from memory.

Protected Member Functions

- void `load_jpg_` (const char *filename, const char *sharename, const unsigned char *data, int data_length=-1)

Additional Inherited Members

33.73.1 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.

33.73.2 Constructor & Destructor Documentation

33.73.2.1 `Fl_JPEG_Image()` [1/2]

```cpp
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

- `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)`

33.73.2.2 `Fl_JPEG_Image()` [2/2]

```cpp
Fl_JPEG_Image::Fl_JPEG_Image {
    const char * name,
    const unsigned char * data,
    int data_length = -1
}
```

The constructor loads the JPEG image from memory.

Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. `name_png` can be NULL. If a name is given, the image is added to the list of shared images (see: `Fl_Shared_Image`) and will be available by that name.
The inherited destructor frees all memory and server resources that are used by the image.
Use `Fl_image::fail()` to check if `Fl_JPEG_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image...
could not be loaded for another reason. If the image has loaded correctly, \texttt{w()}, \texttt{h()}, and \texttt{d()} should return values greater than zero.
Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>A unique name or NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A pointer to the memory location of the JPEG image</td>
</tr>
<tr>
<td>data_length</td>
<td>optional length of data. This will protect memory outside of the data array from illegal read operations</td>
</tr>
</tbody>
</table>

See also

* Fl_JPEG_Image::Fl_JPEG_Image(const char *filename)
* Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_JPEG_Image.H
- Fl_JPEG_Image.cxx

### 33.74 Fl_Kdialog_Native_File_Chooser_Driver Class Reference

Inheritance diagram for Fl_Kdialog_Native_File_Chooser_Driver:

```
Fl_Kdialog_Native_File_Chooser_Driver
  |               |
  v               v
Fl_NATIVE_FILE_CHOOSER_FLTK_DRIVER
```

Friends

- class Fl_Native_File_Chooser
- class Fl_Zenity_Native_File_Chooser_Driver

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser_Kdialog.H
- Fl_Native_File_Chooser_Kdialog.cxx

### 33.75 Fl_Label Struct Reference

This struct stores all information for a text or mixed graphics label.

```cpp
#include <Fl_Widget.H>
```

Public Member Functions

- void **draw**(int, int, int, int, **Fl_Align**) const
  
  Draws the label aligned to the given box.

- void **measure**(int &w, int &h) const
  
  Measures the size of the label.
Public Attributes

- **Fl_Align align_**
  
  alignment of label

- **Fl_Color color**
  
  text color

- **Fl_Image * deimage**
  
  optional image for a deactivated label

- **Fl_Font font**
  
  label font used in text

- **Fl_Image * image**
  
  optional image for an active label

- **Fl_Fontsize size**
  
  size of label font

- **uchar type**
  
  type of label.

- **const char * value**
  
  label text

33.75.1 Detailed Description

This struct stores all information for a text or mixed graphics label.

**Todo** There is an aspiration that the `Fl_Label` type will become a widget by itself. That way we will be avoiding a lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in place in this release.

33.75.2 Member Function Documentation

33.75.2.1 draw()

```c
void Fl_Label::draw ( int X, int Y, int W, int H, Fl_Align align ) const
```

Draws the label aligned to the given box.
Draws a label with arbitrary alignment in an arbitrary box.

33.75.2.2 measure()

```c
void Fl_Label::measure ( int & W, int & H ) const
```

Measures the size of the label.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>: this is the requested size for the label text plus image; on return, this will contain the size needed to fit the label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
33.75.3 Member Data Documentation

33.75.3.1 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

33.76 Fl_Light_Button Class Reference

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.

#include <Fl_Light_Button.H>

Inheritance diagram for Fl_Light_Button:

```
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.

- int handle (int) FL_OVERRIDE
  
  Handles the specified event.

Protected Member Functions

- void draw () FL_OVERRIDE
  
  Draws the widget.

Additional Inherited Members

33.76.1 Detailed Description

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.

The shape of the "light" is initially set to FL_DOWN_BOX. The color of the light when on is controlled with selection_color(), which defaults to FL_YELLOW.

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

Generated by Doxygen
33.76.2 Constructor & Destructor Documentation

33.76.2.1 Fl_Light_Button()

Fl_Light_Button::Fl_Light_Button (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0  
)  
Creates a new Fl_Light_Button widget using the given position, size, and label string.  
The 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.

33.76.3 Member Function Documentation

33.76.3.1 draw()

void Fl_Light_Button::draw ( ) [protected], [virtual]
Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()
Reimplemented from Fl_Button.

33.76.3.2 handle()

int Fl_Light_Button::handle (  
    int event ) [virtual]
Handles the specified event.  
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

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_Button.

The documentation for this class was generated from the following files:

- Fl_Light_Button.H
- Fl_Light_Button.cxx

## 33.77 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

## 33.78 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
  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.

33.78.1  Member Function Documentation

33.78.1.1  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.

33.78.2  Member Data Documentation

33.78.2.1  print

c 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

33.79  Fl_Menu_ Class Reference

Base class of all widgets that have a menu in FLTK.
#include <Fl_Menu_.H>
Inheritance diagram for Fl_Menu_:
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 *, int=0)
  Adds a new menu item.

- int add (const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  See int Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void* user_data=0, int flags=0)

- void clear ()
  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

- int clear_submenu (int index)
  Clears the specified submenu pointed to by index of all menu items.

- void copy (const Fl_Menu_Item *, void *user_data=0)
  Sets the menu array pointer with a copy of m that will be automatically deleted.

- Fl_Boxtype down_box () const
  This box type is used to surround the currently-selected items in the menus.

- void down_box (Fl_Boxtype b)
  Sets the box type used to surround the currently-selected items in the menus.

- Fl_Color down_color () const
  For back compatibility, same as selection_color()

- void down_color (unsigned c)
  For back compatibility, same as selection_color()

- int find_index (const char *) const
  Find the menu item index for a given menu pathname, such as "Edit/Copy".

- int find_index (const Fl_Menu_Item *item) const
  Find the index into the menu array for a given item.

- int find_index (Fl_Callback *) const
  Find the index into the menu array for a given callback cb.

- const Fl_Menu_Item * find_item (const char *name) const
  Find the menu item for a given menu pathname, such as "Edit/Copy".

- const Fl_Menu_Item * find_item (Fl_Callback *)
  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 *, int=0, int=0)
  Inserts a new menu item at the specified index position.

- int insert (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)
  See int Fl_Menu_::insert(const char* label, int shortcut, Fl_Callback*, void* user_data=0, int flags=0)
• **int item_pathname** (char ∗name, int namelen, const Fl_Menu_Item ∗finditem=0) const
  
  Get the menu ‘pathname’ for the specified menuitem.

• **const Fl_Menu_Item ∗ menu () const**
  
  Returns a pointer to the array of Fl_Menu_Items.

• **void menu (const Fl_Menu_Item ∗m)**
  
  Sets the menu array pointer directly.

• **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**
  
  Gets 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

33.79.1 Detailed Description

Base class of all widgets that have a menu in FLTK. Currently FLTK provides you with Fl_Menu_Button, Fl_Menu_Bar, and Fl_Choice. The class contains a pointer to an array of structures of type Fl_Menu_Item. The array may either be supplied directly by the user program, or it may be "private": a dynamically allocated array managed by the Fl_Menu_. When the user clicks a menu item, 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)

33.79.2 Constructor & Destructor Documentation

33.79.2.1 Fl_Menu_( )

Fl_Menu_::Fl_Menu_ (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0 )

Creates a new Fl_Menu_ widget using the given position, size, and label string. menu() is initialized to null.

33.79.3 Member Function Documentation

33.79.3.1 add() [1/2]

int Fl_Menu_::add (  
    const char * str )

This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
The passed string is split at any '|' characters and then add(s,0,0,0) 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

### 33.79.3.2 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>label</th>
<th>The text label for the menu item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortcut</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>callback</td>
<td>Optional callback invoked when user clicks the item. Default 0 if none.</td>
</tr>
<tr>
<td>userdata</td>
<td>Optional user data passed as an argument to the callback. Default 0 if none.</td>
</tr>
<tr>
<td>flags</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_oldShortcut(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_RADIO` // Item is a radio button (one checkbox of many can be on)
- `FL_MENU_INVISIBLE` // Item will not show up (shortcut will work)
- `FLSubMenu_POINTER` // Indicates user_data() is a pointer to another menu array
- `FLSubMenu` // 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 and results of STR#2344

33.79.3.3 clear()

```cpp
void Fl_Menu_::clear ()
```

Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.

Menus must not be cleared during a callback to the same menu.
33.79.3.4  clear_submenu()

```c
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:**

```c
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)`

33.79.3.5  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)`.

33.79.3.6  down_box() [1/2]

```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.

33.79.3.7  down_box() [2/2]

```c
void Fl_Menu_::down_box (  
    Fl_Boxtype b  )  [inline]
```

Sets the box type used to surround the currently-selected items in the menus.
### find_index() [1/3]

```cpp
int Fl_Menu_::find_index (const char * pathname) const
```

Find the menu item index for a given menu `pathname`, such as "Edit/Copy".

This method finds a menu item's index position for the given menu `pathname`, also traversing submenus, but **not** submenu pointers (FL_SUBMENU_POINTER).

To get the menu item pointer for a `pathname`, use `find_item()`

**Parameters**

- `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()`

### find_index() [2/3]

```cpp
int Fl_Menu_::find_index (const Fl_Menu_Item * item) const
```

Find the index into the menu array for a given `item`.

A way to convert a menu item pointer into an index.

**Current implementation is fast and not expensive.**

```c
define index 12;
const Fl_Menu_Item *item = mymenu->menu() + index;
```

**Parameters**

- `item` : The item to be found

**Returns**

The index of the item, or -1 if not found.

**See also**

- `menu()`

### find_index() [3/3]

```cpp
int Fl_Menu_::find_index (Fl_Callback * cb) const
```

Find the index into the menu array for a given callback `cb`.

This method finds a menu item's index position, also traversing submenus, but **not** submenu pointers (FL_SUBMENU_POINTER). This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.
Parameters

\texttt{cb} Find the first item with this callback

Returns

The index of the item with the specific callback, or -1 if not found

See also

\texttt{find_index(const char*)}

\subsection*{33.79.3.11 \texttt{find_item()} [1/2]}

\begin{verbatim}
const Fl_Menu_Item * Fl_Menu_::find_item (const char * pathname )
\end{verbatim}

Find the menu item for a given menu \texttt{pathname}, such as "Edit/Copy".
This method finds a menu item in the menu array, also traversing submenus, but not submenu pointers (FL_\texttt{-SUBMENU_POINTER}).
To get the menu item's index, use \texttt{find_index(const char*)}

\textbf{Example:}

\begin{verbatim}
Fl_Menu_Bar *menubar = new Fl_Menu_Bar(...);
menubar->add("File/Open");
menubar->add("File/Save");
menubar->add("Edit/Copy");
// [...]
Fl_Menu_Item *item;
if ( ( item = (Fl_Menu_Item*)menubar->find_item("File/Open") ) != NULL ) {
  item->labelcolor(FL_RED);
}
if ( ( item = (Fl_Menu_Item*)menubar->find_item("Edit/Copy") ) != NULL ) {
  item->labelcolor(FL_GREEN);
}
\end{verbatim}

Parameters

\texttt{pathname} The path and name of the menu item

Returns

The item found, or NULL if not found

See also

\texttt{find_index(const char*)}, \texttt{find_item(Fl_Callback*)}, \texttt{item_pathname()}

\subsection*{33.79.3.12 \texttt{find_item()} [2/2]}

\begin{verbatim}
const Fl_Menu_Item * Fl_Menu_::find_item (Fl_Callback * cb )
\end{verbatim}

Find the menu item for the given callback \texttt{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

\texttt{in \texttt{cb}} find the first item with this callback

Generated by Doxygen
Returns

The item found, or NULL if not found

See also

find_item(const char*)

33.79.3.13 global()

```cpp
void Fl_Menu_::global ()
```

Make the shortcuts for this menu work no matter what window has the focus when you type it. This is done by using `Fl::add_handler()`. This `Fl_Menu_` widget does not have to be visible (i.e., the window it is in can be hidden, or it does not have to be put in a window at all).

Currently there can be only one `global()` menu. Setting a new one will replace the old one. There is no way to remove the `global()` setting (so don't destroy the widget!)

33.79.3.14 insert()

```cpp
int Fl_Menu_::insert ( int index, const char * label, int shortcut, Fl_Callback * callback, void * userdata = 0, int flags = 0 )
```

Inserts a new menu item at the specified `index` position.

If `index` is -1, the menu item is appended; same behavior as `add()`.

To properly insert a menu item, `label` must be the name of the item (e.g., "Quit"), and not a 'menu pathname' (e.g., "File/Quit"). If a menu pathname is specified, the value of `index` is ignored; the new item's position defined by the pathname.

For more details, see `add()`. Except for the `index` parameter, `add()` has more detailed information on parameters and behavior, and is functionally equivalent.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `index`   | The menu array's index position where the new item is inserted. If -1, behavior is the same as `add()`.
| `label`   | The text label for the menu item. If the label is a menu pathname, `index` is ignored, and the pathname indicates the position of the new item.
| `shortcut` | Optional keyboard shortcut. Can be an int (FL_CTRL+'a') or a string ("^a"). Default is 0. |
| `callback` | Optional callback invoked when user clicks the item. Default 0 if none. |
| `userdata` | Optional user data passed as an argument to the callback. Default 0 if none. |
| `flags`   | Optional flags that control the type of menu item; see `add()` for more info. Default is 0 for none. |

Returns

The index into the `menu()` array, where the entry was added.

See also

`add()`
33.79.3.15 item_pathname()

```cpp
int Fl_Menu_::item_pathname (char * name, int namelen, const Fl_Menu_Item * finditem = 0) const
```

Get the menu 'pathname' for the specified menuitem. If finditem==NULL, mvalue() is used (the most recently picked menuitem).

**Example:**

```cpp
Fl_Menu_Bar *menubar = 0;
void my_menu_callback(Fl_Widget*,void*) {
    char name[80];
    if ( menubar->item_pathname(name, sizeof(name)-1) == 0 ) { // recently picked item
        if ( strcmp(name, "File/&Open") == 0 ) { .. } // open invoked
        if ( strcmp(name, "File/&Save") == 0 ) { .. } // save invoked
        if ( strcmp(name, "Edit/&Copy") == 0 ) { .. } // copy invoked
    }
}
int main() {
    [..]
    menubar = new Fl_Menu_Bar(..);
    menubar->add("File/&Open", 0, my_menu_callback);
    menubar->add("File/&Save", 0, my_menu_callback);
    menubar->add("Edit/&Copy", 0, my_menu_callback);
    [..]
}
```

**Returns**

- 0 : OK (name has menuitem’s pathname)
- -1 : item not found (name="")
- -2 : ‘name’ not large enough (name="")

**See also**

find_item()

33.79.3.16 menu() [1/2]

```cpp
const Fl_Menu_Item * Fl_Menu_::menu ( ) const [inline]
```

Returns a pointer to the array of Fl_Menu_Items.

This will either be the value passed to menu(value) or the private copy or an internal (temporary) location (see note below).

**Note**

**Implementation details - may be changed in the future.** All modifications of the menu array are done by copying the entire menu array to an internal storage for optimization of memory allocations, for instance when using add() or insert(). While this is done, menu() returns the pointer to this internal location. The entire menu will be copied back to private storage when needed, i.e. when another Fl_Menu_ is modified. You can force this reallocation after you’re done with all menu modifications by calling Fl_Menu_::menu_end() to make sure menu() returns a permanent pointer to private storage (until the menu is modified again). Note also that some menu methods (e.g. Fl_Menu_Button::popup()) call menu_end() internally to ensure a consistent menu array while the menu is open.

**See also**

size() – returns the size of the Fl_Menu_Item array.

menu_end() – finish menu modifications (optional)

**Example:** How to walk the array:

```cpp
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
", t, item.label() ? item.label() : "(Null)", (item.flags & FL_MENU_VALUE) ? "set" : "clear", (item.flags & FL_SUBMENU) ? "Submenu" : "Item"); // see if item is a submenu or actual item
}
```
33.79.3.17  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.

33.79.3.18  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()

33.79.3.19  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.

33.79.3.20  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.

33.79.3.21  mvalue()

const Fl_Menu_Item * Fl_Menu_::mvalue ( ) const [inline]

Returns a pointer to the last menu item that was picked.
33.79.3.22 picked()

```cpp
const Fl_Menu_Item * Fl_Menu_::picked (const Fl_Menu_Item * v)
```

When user picks a menu item, call this. It will do the callback. Unfortunately this also casts away const for the checkboxes, but this was necessary so non-checkbox menus can really be declared const...

33.79.3.23 remove()

```cpp
void Fl_Menu_::remove (int i)
```

Deletes item i from the menu. If the menu array was directly set with menu(x) then copy() is done to make a private array. No items must be removed from a menu during a callback to the same menu.

Parameters

| i | index into menu array |

33.79.3.24 replace()

```cpp
void Fl_Menu_::replace (int i, const char * str)
```

Changes the text of item i. This is the only way to get slash into an add()'ed menu item. If the menu array was directly set with menu(x) then copy() is done to make a private array.

Parameters

| i | index into menu array |
| str | new label for menu item at index i |

33.79.3.25 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).

33.79.3.26 testShortcut()

```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.
33.79.3.27  text() [1/2]

const char * Fl_Menu_::text() const [inline]
Returns the title of the last item chosen.

33.79.3.28  text() [2/2]

const char * Fl_Menu_::text(int i) const [inline]
Returns the title of item i.

33.79.3.29  textcolor()

Fl_Color Fl_Menu_::textcolor() const [inline]
Get the current color of menu item labels.

33.79.3.30  textfont() [1/2]

Fl_Font Fl_Menu_::textfont() const [inline]
Gets the current font of menu item labels.

33.79.3.31  textfont() [2/2]

void Fl_Menu_::textfont(Fl_Font c) [inline]
Sets the current font of menu item labels.

33.79.3.32  textsize() [1/2]

Fl_Fontsize Fl_Menu_::textsize() const [inline]
Gets the font size of menu item labels.

33.79.3.33  textsize() [2/2]

void Fl_Menu_::textsize(Fl_Fontsize c) [inline]
Sets the font size of menu item labels.

33.79.3.34  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.
33.79.3.35 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.

33.79.3.36 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

33.80 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

#include <Fl_Menu_Bar.H>

Inheritance diagram for Fl_Menu_Bar:

```
Fl_Widget
   |
   V
Fl_Menu_
   |
   V
Fl_Menu_Bar
   |
   V
Fl_Sys_Menu_Bar
```

Public Member Functions

- **Fl_Menu_Bar** (int X, int Y, int W, int H, const char * l=0)

  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.

- int **handle** (int) FL_OVERRIDE

  Handles the specified event.

- virtual void **update** ()

  Updates the menu bar after any change to its items.

Protected Member Functions

- void **draw** () FL_OVERRIDE

  Draws the widget.

Friends

- class **Fl_Sys_Menu_Bar_Driver**
**Additional Inherited Members**

### 33.80.1 Detailed Description

This widget provides a standard menubar interface. Usually you will put this widget along the top edge of your window. The height of the widget should be 30 for the menu titles to draw correctly with the default font.

The items on the bar and the menus they bring up are defined by a single `Fl_Menu_Item` array. Because a `Fl_Menu_Item` array defines a hierarchy, the top level menu defines the items in the menubar, while the submenus define the pull-down menus. Sub-sub menus and lower pop up to the right of the submenus.

![Figure 33.24 menubar](image)

If there is an item in the top menu that is not a title of a submenu, then it acts like a "button" in the menubar. Clicking on it will pick it.

When the user clicks a menu item, `value()` is set to that item and then:

- The item's callback is done if one has been set; the `Fl_Menu_Bar` is passed as the `Fl_Widget` argument, along with any userdata configured for the callback.
- If the item does not have a callback, the `Fl_Menu_Bar`'s callback is done instead, along with any userdata configured for the callback. The callback can determine which item was picked using `value()`, `mvalue()`, `item_pathname()`, etc.

Submenus will also pop up in response to shortcuts indicated by putting a `&` character in the name field of the menu item. If you put a `&` character in a top-level "button" then the shortcut picks it. The `&` character in submenus is ignored until the menu is popped up. Typing the `shortcut()` of any of the menu items will cause callbacks exactly the same as when you pick the item with the mouse.

### 33.80.2 Constructor & Destructor Documentation

#### 33.80.2.1 Fl_Menu_Bar()

```cpp
Fl_Menu_Bar::Fl_Menu_Bar ( 
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0
)
```

Creates a new `Fl_Menu_Bar` widget using the given position, size, and label string. The default boxtype is `FL_UP_BOX`.

The constructor sets `menu()` to NULL. See `Fl_Menu_` for the methods to set or change the menu. `labellsize()`, `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.

### 33.80.3 Member Function Documentation
33.80.3.1 draw()

```cpp
def Fl_Menu_Bar::draw () [protected], [virtual]
```
Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own* draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

Reimplemented in Fl_Sys_Menu_Bar.

33.80.3.2 handle()

```cpp
int Fl_Menu_Bar::handle ( ) [virtual]
```
Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**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.

33.80.3.3 update()

```cpp
virtual void Fl_Menu_Bar::update ( ) [inline], [virtual]
```
Updates the menu bar after any change to its items.

This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.

Reimplemented in Fl_Sys_Menu_Bar.

The documentation for this class was generated from the following files:

- Fl_Menu_Bar.H
- Fl_Menu_Bar.hxx

33.81 Fl_Menu_Button Class Reference

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.

```cpp
#include <Fl_Menu_Button.H>
```

Inheritance diagram for 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) FL_OVERRIDE` handles the specified event.
- `const Fl_Menu_Item *popup()` Act exactly as though the user clicked the button or typed the shortcut key.

Protected Member Functions

- `void draw() FL_OVERRIDE` Draws the widget.

Static Protected Attributes

- `static Fl_Menu_Button *pressed_menu_button_ = NULL`

Additional Inherited Members

33.81.1 Detailed Description

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of `Fl_Menu_Item` objects.
Normally any mouse button will pop up a menu and it is lined up below the button as shown in the picture. However an \texttt{Fl\_Menu\_Button} may also control a pop-up menu. This is done by setting the \texttt{type()}. If \texttt{type()} is zero a normal menu button is produced. If it is nonzero then this is a pop-up menu. The bits in \texttt{type()} indicate what mouse buttons pop up the menu (see \texttt{Fl\_Menu\_Button::popup\_buttons}).

The menu will also pop up in response to shortcuts indicated by putting a ‘&’ character in the \texttt{label()}. Typing the \texttt{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, \texttt{value()} is set to that item and then:

- The item's callback is done if one has been set; the \texttt{Fl\_Menu\_Button} is passed as the \texttt{Fl\_Widget*} argument, along with any userdata configured for the callback.

- If the item does not have a callback, the \texttt{Fl\_Menu\_Button}'s callback is done instead, along with any userdata configured for it. The callback can determine which item was picked using \texttt{value()}, \texttt{mvalue()}, \texttt{item\_pathname()}, etc.

### 33.81.2 Member Enumeration Documentation

#### 33.81.2.1 popup\_buttons

\begin{verbatim}
enum Fl_Menu_Button::popup_buttons

indicate what mouse buttons pop up the menu.
Values for \texttt{type()} used to indicate what mouse buttons pop up the menu. \texttt{Fl\_Menu\_Button::POPUP3} is usually what you want.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPUP1</td>
<td>pops up with the mouse 1st button.</td>
</tr>
<tr>
<td>POPUP2</td>
<td>pops up with the mouse 2nd button.</td>
</tr>
<tr>
<td>POPUP12</td>
<td>pops up with the mouse 1st or 2nd buttons.</td>
</tr>
<tr>
<td>POPUP3</td>
<td>pops up with the mouse 3rd button.</td>
</tr>
<tr>
<td>POPUP13</td>
<td>pops up with the mouse 1st or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP23</td>
<td>pops up with the mouse 2nd or 3rd buttons.</td>
</tr>
<tr>
<td>POPUP123</td>
<td>pops up with any mouse button.</td>
</tr>
</tbody>
</table>
\end{verbatim}
33.81.3 Constructor & Destructor Documentation

33.81.3.1 Fl_Menu_Button()

Fl_Menu_Button::Fl_Menu_Button (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * l = 0 )  

Creates a new Fl_Menu_Button widget using the given position, size, and label string.  
The default boxtype is FL_UP_BOX.  
The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

33.81.4 Member Function Documentation

33.81.4.1 draw()

void Fl_Menu_Button::draw ( ) [protected], [virtual]  

Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as  
soon as possible, call redraw() instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded  
scrollbar, you can do it (because draw() is virtual) like this:  
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()  
Implements Fl_Widget.

33.81.4.2 handle()

int Fl_Menu_Button::handle (  
  int event ) [virtual]  

Handles the specified event.  
You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-  
circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event  
Reimplemented from Fl_Widget.
33.81.4.3 popup()

const Fl_Menu_Item * Fl_Menu_Button::popup()

Act exactly as though the user clicked the button or typed the shortcut key.
The menu appears, it waits for the user to pick an item, and if they pick one it sets value() and does the callback or sets changed() as described above. The menu item is returned or NULL if the user dismisses the menu.

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

33.82 Fl_Menu_Item Struct Reference

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

#include <Fl_Menu_Item.H>

Public Member Functions

• void activate()
  Allows a menu item to be picked.

• int active() const
  Gets whether or not the item can be picked.

• int activevisible() const
  Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.

• int add(const char *, int shortcut, Fl_Callback *, void *, 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 label type.
• void labeltype (Fl_Labeltype a)
  Sets the menu item's labeltype.
• int measure (int *h, const Fl_Menu_Item * const)
  Measures width of label, including effect of & characters.
• 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 i=1) const
  Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items.
• const Fl_Menu_Item * popup (int X, int Y, const char *title=0, const Fl_Menu_Item *picked=0, const Fl_Menu_Item *=0) const
  This method is called by widgets that want to display menus.
• const Fl_Menu_Item * pulldown (int X, int Y, int W, int H, const Fl_Menu_Item *picked=0, const Fl_Menu_Item *=0, const Fl_Menu_Item *=0, int menubar=0) const
  Pulldown() is similar to popup(), but a rectangle is provided to position the menu.
• int radio () const
  Returns true if this item is a radio item.
• void set ()
  Turns the check or radio item "on" for the menu item.
• void setonly (Fl_Menu_Item const *first=NULL)
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.
• int shortcut () const
  Gets what key combination shortcut will trigger the menu item.
• void shortcut (int s)
  Sets exactly what key combination will trigger the menu item.
• void show ()
  Makes an item visible in the menu.
• int size () const
  Size of the menu starting from this menu item.
• int submenu () const
  Returns true if either FL_SUBMENU or FL_SUBMENU_POINTER is on in the flags.
• const Fl_Menu_Item * test_shortcut () const
  This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.
• void uncheck ()
  Back compatibility only.
• void * user_data () const
  Gets the user_data() argument that is sent to the callback function.
• void user_data (void *v)
  Sets the user_data() argument that is sent to the callback function.
• int value () const
  Returns the current value of the check or radio item.
• void value (int v)
  Sets the current value of the check or radio item.
• int visible () const
  Gets the visibility of an item.
Public Attributes

- **Fl_Callback * callback_**
  - menu item callback
- **int flags**
  - menu item flags like FL_MENU_TOGGLE, FL_MENU_RADIO
- **Fl_Color labelcolor_**
  - menu item text color
- **Fl_Font labelfont_**
  - which font for this menu item text
- **Fl_Fontsize labelsize_**
  - size of menu item text
- **uchar labeltype_**
  - how the menu item text looks like
- **int shortcut_**
  - menu item shortcut
- **const char* text**
  - menu item text, returned by label()
- **void* user_data_**
  - menu item user_data for the menu’s callback

33.82.1 Detailed Description

The `Fl_Menu_Item` structure defines a single menu item that is used by the `Fl_Menu_` class.

```c
struct Fl_Menu_Item {
    const char* text; // label()
    int shortcut_;       // On/off state for checkbox/radio buttons
    Fl_Callback* callback_; // Callback function
    void* user_data_;    // User data for menu item
    uchar labeltype_;     // How the menu item text looks
    uchar labelfont_;     // Which font for this menu item text
    uchar labelsize_;     // Size of menu item text
    uchar labelcolor_;    // Menu item text color
};
```

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"},
    {"inactive", FL_ALT+'i', 0, 0, FL_MENU_INACTIVE|FL_MENU_DIVIDER},
    {"invisible", FL_ALT+'i', 0, 0, FL_MENU_INVISIBLE},
    {"check", FL_ALT+' ', 0, 0, FL_MENU_TOGGLE|FL_MENU_VALUE},
    {"box", FL_ALT+' ', 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.

### 33.82.2 Member Function Documentation

#### 33.82.2.1 add()

```cpp
int Fl_Menu_Item::add (const char * mytext, int sc, Fl_Callback * cb, void * data = 0, int myflags = 0)
```

Add 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)

33.82.2.2 argument() [1/2]

long Fl_Menu_Item::argument ( ) const [inline]
Gets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the stored user-
data() argument to long and returns it as a long value.

33.82.2.3 argument() [2/2]

void Fl_Menu_Item::argument (long v) [inline]
Sets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the given argument
v to void* and stores it in the menu item's userdata() member. This may not be portable to some machines.

33.82.2.4 callback() [1/5]

Fl_Callback_p Fl_Menu_Item::callback ( ) const [inline]
Returns the callback function that is set for the menu item.
Each item has space for a callback function and an argument for that function. Due to back compatibility, the
Fl_Menu_Item itself is not passed to the callback, instead you have to get it by calling ((Fl_Menu_+w)->mvalue() where w is the widget argument.

33.82.2.5 callback() [2/5]

void Fl_Menu_Item::callback (Fl_Callback * c) [inline]
Sets the menu item's callback function.
This method does not set the userdata() argument.

See also
Fl_Callback_p Fl_MenuItem::callback() const

33.82.2.6 callback() [3/5]

void Fl_Menu_Item::callback (Fl_Callback * c, void * p) [inline]
Sets the menu item's callback function and userdata() argument.

See also
Fl_Callback_p Fl_MenuItem::callback() const
33.82.2.7 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
```

33.82.2.8 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
```

33.82.2.9 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()
```

33.82.2.10 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.

33.82.2.11 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()
```

33.82.2.12 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.
33.82.2.13  do_callback() [1/3]

void Fl_Menu_Item::do_callback (  
    Fl_Widget * o ) const [inline]

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.  
The callback is called with the stored user_data() as its second argument.  You must first check that callback() is  
non-zero before calling this.

33.82.2.14  do_callback() [2/3]

void Fl_Menu_Item::do_callback (  
    Fl_Widget * o,  
    long arg ) const [inline]

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.  
This call overrides the callback's second argument with the given value arg.  long arg is cast to void* when calling  
the callback.  You must first check that callback() is non-zero before calling this.

33.82.2.15  do_callback() [3/3]

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.

33.82.2.16  find_shortcut()

const Fl_Menu_Item * Fl_Menu_Item::find_shortcut (  
    int * ip = 0,  
    const bool require_alt = false ) const

Search only the top level menu for a shortcut.  
Either &x in the label or the shortcut fields are used.  
This tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value.

Parameters

<table>
<thead>
<tr>
<th>ip</th>
<th>returns the index of the item, if ip is not NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>require_alt</td>
<td>if true: match only if Alt key is pressed.</td>
</tr>
</tbody>
</table>

Returns

found Fl_Menu_Item or NULL

33.82.2.17  image() [1/2]

void Fl_Menu_Item::image (  
    Fl_Image & image ) [inline]

Compatibility API for FLUID, same as image.label(this).

Note

This method is intended for internal use by fluid and may not do what you expect.

33.82.2.18  image() [2/2]

void Fl_Menu_Item::image (  
    Fl_Image * image ) [inline]
Compatibility API for FLUID, same as image->label(this).

Note
This method is intended for internal use by fluid and may not do what you expect.

### 33.82.2.19 insert()

```c
int Fl_Menu_Item::insert (
    int index,
    const char * mytext,
    int sc,
    Fl_Callback * cb,
    void * data = 0,
    int myflags = 0
)
```

Inserts an item at position `index`.
If `index` is -1, the item is added the same way as `Fl_Menu_Item::add()`.
If `mytext` contains any un-escaped front slashes (/), it's assumed a menu pathname is being specified, and the value of `index` will be ignored.
In all other aspects, the behavior of `insert()` is the same as `add()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</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 Fl_Menu_Item</td>
</tr>
</tbody>
</table>

**Returns**

the index into the menu() array, where the entry was added

### 33.82.2.20 label()

```c
const char * Fl_Menu_Item::label ( ) const [inline]
```

Returns the title of the item.
A NULL here indicates the end of the menu (or of a submenu). A '&' in the item will print an underscore under the next letter, and if the menu is popped up that letter will be a "shortcut" to pick that item. To get a real '&' put two in a row.

### 33.82.2.21 labelcolor() [1/2]

```c
Fl_Color Fl_Menu_Item::labelcolor ( ) const [inline]
```

Gets the menu item's label color.
This color is passed to the labelfont routine, and is typically the color of the label text. This defaults to FL_BLACK. If this color is not black fltk will not use overlay bitplanes to draw the menu - this is so that images put in the menu draw correctly.

### 33.82.2.22 labelcolor() [2/2]

```c
void Fl_Menu_Item::labelcolor ( Fl_Color a ) [inline]
```

Sets the menu item's label color.
See also

\texttt{Fl\_Color Fl\_Menu\_Item::labelcolor() const}

33.82.2.23 \textbf{labelfont()} [1/2]

\texttt{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 \texttt{Fl::set\_font()} can define new fonts.

33.82.2.24 \textbf{labelfont()} [2/2]

\texttt{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 \texttt{Fl::set\_font()} can define new fonts.

33.82.2.25 \textbf{labeltype()} [1/2]

\texttt{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 \texttt{label()} pointer as another form of data such as a bitmap. The value \texttt{FL\_NORMAL\_LABEL} prints the label as text.

33.82.2.26 \textbf{labeltype()} [2/2]

\texttt{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 \texttt{label()} pointer as another form of data such as a bitmap. The value \texttt{FL\_NORMAL\_LABEL} prints the label as text.

33.82.2.27 \textbf{measure()}

\texttt{int Fl\_Menu\_Item::measure ( int * hp, const Fl\_Menu\_Item * m ) const}

Measures width of label, including effect of & characters. Optionally, can get height if hp is not NULL.

33.82.2.28 \textbf{next()} [1/2]

\texttt{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.

33.82.2.29 \textbf{next()} [2/2]

\texttt{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.
33.82 Fl_Menu_Item Struct Reference 707

33.82.2.30 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

33.82.2.31 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.

33.82.2.32 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.

33.82.2.33 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.

### 33.82.2.34 `setonly()`

```cpp
void Fl_Menu_Item::setonly (Fl_Menu_Item const * first = NULL)
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

**Note**

This method is dangerous if radio items are first in the menu. Make sure that `first` is set correctly or use `Fl_Menu_Item::setonly(Fl_Menu_Item*)` instead.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>first</code></td>
<td>start of menu array or NULL (default) if the radio group is not the first item</td>
</tr>
</tbody>
</table>

### 33.82.2.35 `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).

### 33.82.2.36 `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
```

### 33.82.2.37 `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.
33.82.2.38  test_shortcut()

    const Fl_Menu_Item * Fl_Menu_Item::test_shortcut ( ) const

This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event. If
the current event matches one of the items shortcut, that item is returned. If the keystroke does not
match any shortcuts then NULL is returned. This only matches the shortcut() fields, not the letters in the title preceeded by '

33.82.2.39  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()

33.82.2.40  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

33.83  Fl_Menu_Window Class Reference

The Fl_Menu_Window widget is a window type used for menus.

```
#include <Fl_Menu_Window.H>
```

Inheritance diagram for Fl_Menu_Window:

```
Fl_Widget
  |
  v
Fl_Group
  |
  v
Fl_Window
    |
    v
Fl_Single_Window
      |
      v
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

33.83.1 Detailed Description

The Fl_Menu_Window widget is a window type used for menus. By default, the window is drawn in the hardware overlay planes if they are available so that the menu don't force the rest of the window to redraw. The documentation for this class was generated from the following files:

- Fl_Menu_Window.H
- Fl_Menu_Window.cxx

33.84 Fl_Multi_Browser Class Reference

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

#include <Fl_Multi_Browser.H>

Inheritance diagram for Fl_Multi_Browser:

```
Fl_Widget
  |
  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

33.84.1 Detailed Description

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

Figure 33.27 Fl_Multi_Browser
The user interface is Macintosh style: clicking an item turns off all the others and selects that one, dragging selects all the items the mouse moves over, and ctrl + click (Cmd+click on the Mac OS platform) toggles the items. Shift + click extends the selection until the clicked item. This is different from how forms did it. Normally the callback is done when the user releases the mouse, but you can change this with when(). See Fl_Browser for methods to add and remove lines from the browser.

### 33.84.2 Constructor & Destructor Documentation

#### 33.84.2.1 Fl_Multi_Browser()

```cpp
Fl_Multi_Browser::Fl_Multi_Browser (int X, int Y, int W, int H, const char * L = 0)
```

Creates a new Fl_Multi_Browser widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL_MULTI_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.

#### 33.85 Fl_Multi_Label Struct Reference

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.

```cpp
#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.

### 33.85.1 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.
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**

```c
FL_Pixmap *image = new FL_Pixmap(..); // image for menu item; any Fl_Image based widget
FL_Menu_Bar *menu = new FL_Menu_Bar(..); // can be any Fl_Menu oriented widget (Fl_Choice, Fl_Menu_Button..)

// Create a menu item
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;
ml->typea = FL_IMAGE_LABEL;
ml->labela = (const char*)image; // any Fl_Image widget: FL_Pixmap, FL_PNG_Image, etc..
ml->typeb = FL_NORMAL_LABEL;
ml->labelb = item->label();
ml->label(item);
```

See also `Fl_Label` and `Fl_Labeltype` and examples/howto-menu-with-images.cxx

### 33.85.2 Member Data Documentation

#### 33.85.2.1 labela

`const char* Fl_Multi_Label::labela`

Holds the "leftmost" of the two elements in the composite label. Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

#### 33.85.2.2 labelb

`const char* Fl_Multi_Label::labelb`

Holds the "rightmost" of the two elements in the composite label. Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

#### 33.85.2.3 typea

`uchar Fl_Multi_Label::typea`

Holds the "type" of labela. Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.

#### 33.85.2.4 typeb

`uchar Fl_Multi_Label::typeb`

Holds the "type" of labelb.
Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together. The documentation for this struct was generated from the following files:

- Fl_Multi_Label.H
- Fl_Multi_Label.hxx

### 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.

```cpp
#include <Fl_Multiline_Input.H>
```

Inheritance diagram for Fl_Multiline_Input:

```
Fl_Widget
  |
  v
Fl_Input_
  |
  v
Fl_Input
  |
  v
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**

#### 33.86.1 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.

#### 33.86.2 Constructor & Destructor Documentation

**33.86.2.1 Fl_Multiline_Input()**

```cpp
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

### 33.87 Fl_Multiline_Output Class Reference

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

```c
#include <Fl_Multiline_Output.H>
```

Inheritance diagram for Fl_Multiline_Output:

```
    Fl_Widget
     |        |
    Fl_Input_
     |        |
    Fl_Input
     |        |
    Fl_Output
        
    Fl_Multiline_Output
```

#### Public Member Functions

- **Fl_Multiline_Output** (int X, int Y, int W, int H, const char *l = 0)

  Creates a new Fl_Multiline_Output widget using the given position, size, and label string.

#### Additional Inherited Members

### 33.87.1 Detailed Description

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

It also displays tab characters as whitespace to the next column.

Note that this widget does not support scrollbars, or per-character color control.

If you are presenting large amounts of read-only text and need scrollbars, or full color control of characters, then use Fl_Text_Display. If you want to display HTML text, use Fl_Help_View.

A caret cursor (^) shows the keyboard navigation mark for keyboard selection of the output text, e.g. Arrow Keys to move the cursor, Shift + Arrow Keys to create a text selection, and ^C to copy the selected text to the paste buffer.

The caret cursor can be disabled by disabling the widget's "visible focus" using clear_visible_focus(), inherited from the Fl_Widget base class. Doing this also disables the widget's keyboard navigation.

### 33.87.2 Constructor & Destructor Documentation

#### 33.87.2.1 Fl_Multiline_Output()

```c
Fl_Multiline_Output::Fl_Multiline_Output ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char * l = 0 )
```
33.88 Fl_Native_File_Chooser Class Reference

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

33.88 Fl_Native_File_Chooser Class Reference

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

Public Types

- enum Option {
  NO_OPTIONS = 0x0000 , SAVEAS_CONFIRM = 0x0001 , NEW_FOLDER = 0x0002 , PREVIEW = 0x0004 ,
  USE_FILTER_EXT = 0x0008 }
- enum Type {
  BROWSE_FILE = 0 , BROWSE_DIRECTORY , BROWSE_MULTI_FILE , BROWSE_MULTI_DIRECTORY ,
  BROWSE_SAVE_FILE , BROWSE_SAVE_DIRECTORY }

Public Member Functions

- int count () const
  Returns the number of filenames (or directory names) the user selected.
- const char * directory () const
  Returns the current preset directory() value.
- void directory (const char * val)
  Preset the directory the browser will show when opened.
- const char * errmsg () const
  Returns a system dependent error message for the last method that failed.
- const char * filename () const
  Return the filename the user chose.
- const char * filename (int i) const
  Return one of the filenames the user selected.
- const char * filter () const
  Returns the filter string last set.
- void filter (const char * f)
  Sets the filename filters used for browsing.
- int filter_value () const
  Returns which filter value was last selected by the user.
- void filter_value (int i)
  Sets which filter will be initially selected.
- int filters () const
  Gets how many filters were available, not including "All Files".
- 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.

33.88.1 Detailed Description
This class lets an FLTK application easily and consistently access the operating system's native file chooser. Some operating systems have very complex and specific file choosers that many users want access to specifically, instead of FLTK's default file chooser(s).
In cases where there is no native file browser, FLTK's own file browser is used instead.
To use this widget, use the following include in your code:
#include <FL/Fl_Native_File_Chooser.H>
The following example shows how to pick a single file:
// Create and post the local native file chooser
#include <FL/Fl_Native_File_Chooser.H>
[...
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE_FILE);
fnfc.filter("Text\n\nT*.txt\n\nC Files\n\t.*.{cxx,h,c}");
fnfc.directory="/var/tmp"; // default directory to use
// Show native chooser
switch ( fnfc.show() ) {
   case -1: printf("ERROR: %s\n", fnfc.errmsg()); break; // ERROR
   case 1: printf("CANCEL\n"); break; // CANCEL
   default: printf("PICKED: %s\n", fnfc.filename()); break; // FILE CHOSEN
}

The Fl_Native_File_Chooser widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the fl_fopen() or fl_open() utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).
Platform Specific Caveats
• Under X11/Wayland, what dialog opens is determined as follows:
  1. If command zenithy is available at run-time and if Fl::option(OPTION_FNFC_USES_ZENITY) is not turned off, the zenithy-based dialog opens. This is expected to be more appropriate than other dialog forms for sandboxed apps.
  2. Else if the app runs under the KDE desktop and if command kdiallog is available at run-time and if the library was not built with cmake -DOPTION_USE_KDIALOG=off, the kdiallog-based dialog opens.
  3. Else if the GTK library is available at run-time on the computer and if Fl::option(OPTION_FNFC_USES_GTK) is not turned off, the GTK-styled dialog opens. Call fl_register_images() to add a "Preview" button to this dialog. Use the static public attributes of class Fl_File_Chooser to localize the browser.
4. Otherwise, FLTK's own dialog produced by the Fl_File_Chooser class opens. Call fl_register_images() to add a "Preview" button to it. It's best if you also call Fl_File_Icon::load_system_icons() at the start of main(), to enable the nicer looking file browser widgets. Use the static public attributes of class Fl_File_Chooser to localize the browser.

- Some operating systems support certain OS specific options; see Fl_Native_File_Chooser::options() for a list.

![Image of different platforms]

Figure 33.29 The Fl_Native_File_Chooser on different platforms

### 33.88.2 Member Enumeration Documentation

#### 33.88.2.1 Option

```cpp
enum Fl_Native_File_Chooser::Option
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_OPTIONS</td>
<td>no options enabled</td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Show native 'Save As' overwrite confirm dialog.</td>
</tr>
<tr>
<td>NEW_FOLDER</td>
<td>Show 'New Folder' icon (if supported)</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>enable preview mode (if supported)</td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter pilots the output file extension (if supported)</td>
</tr>
</tbody>
</table>

Generated by Doxygen
33.88.2.2 Type

```cpp
enum Fl_Native_File_Chooser::Type
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWSE_FILE</td>
<td>browse files (lets user choose one file)</td>
</tr>
<tr>
<td>BROWSE_DIRECTORY</td>
<td>browse directories (lets user choose one directory)</td>
</tr>
<tr>
<td>BROWSE_MULTI_FILE</td>
<td>browse files (lets user choose multiple files)</td>
</tr>
<tr>
<td>BROWSE_MULTI_DIRECTORY</td>
<td>browse directories (lets user choose multiple directories)</td>
</tr>
<tr>
<td>BROWSE_SAVE_FILE</td>
<td>browse to save a file</td>
</tr>
<tr>
<td>BROWSE_SAVE_DIRECTORY</td>
<td>browse to save a directory</td>
</tr>
</tbody>
</table>

33.88.3 Constructor & Destructor Documentation

33.88.3.1 Fl_Native_File_Chooser()

```cpp
Fl_Native_File_Chooser::Fl_Native_File_Chooser ( int val = BROWSE_FILE )
```

The constructor. Internally allocates the native widgets. Optional `val` presets the type of browser this will be, which can also be changed with `type()`.

33.88.3.2 ~Fl_Native_File_Chooser()

```cpp
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ( )
```

Destructor. Deallocates any resources allocated to this widget.

33.88.4 Member Function Documentation

33.88.4.1 count()

```cpp
int Fl_Native_File_Chooser::count ( ) const
```

Returns the number of filenames (or directory names) the user selected.

Example:
```cpp
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for (int n=0; n<fnfc->count(); n++) {
        printf("%d) '%s'
", n, fnfc->filename(n));
    }
}
```

33.88.4.2 directory()

```cpp
void Fl_Native_File_Chooser::directory ( const char * val )
```

Preset the directory the browser will show when opened. If `val` is NULL, or no directory is specified, the chooser will attempt to use the last non-cancelled folder.

33.88.4.3 errmsg()

```cpp
const char * Fl_Native_File_Chooser::errmsg ( ) const
```

Returns a system dependent error message for the last method that failed.
This message should at least be flagged to the user in a dialog box, or to some kind of error log. Contents will be valid only for methods that document `errmsg()` will have info on failures.

33.88.4.4 filename() [1/2]

```cpp
const char * Fl_Native_File_Chooser::filename () const
```

Return the filename the user chose.
Use this if only expecting a single filename. If more than one filename is expected, use filename(int) instead. Return value may be `""` if no filename was chosen (eg. user cancelled).

33.88.4.5 filename() [2/2]

```cpp
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:**
```cpp
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for (int n=0; n<fnfc->count(); n++) {
        printf("%d) '%s'
", n, fnfc->filename(n));
    }
}
```

33.88.4.6 filter() [1/2]

```cpp
const char * Fl_Native_File_Chooser::filter ( ) const
```

Returns the filter string last set.
Can be NULL if no filter was set.

33.88.4.7 filter() [2/2]

```cpp
void Fl_Native_File_Chooser::filter ( const char * f )
```

Sets the filename filters used for browsing.
The default is NULL, which browses all files.
The filter string can be any of:

- A single wildcard (eg. "*.txt")
- Multiple wildcards (eg. "*.{cxx,h,H}")
- A descriptive name followed by a "\t" and a wildcard (eg. "Text Files\t*.txt")
- A list of separate wildcards with a "\n" between each (eg. "*.{cxx,H}\n*.txt")
- A list of descriptive names and wildcards (eg. "C++ Files\t*.{cxx,H}\nTxt Files\t*.txt")

The format of each filter is a wildcard, or an optional user description followed by \t' and the wildcard.
On most platforms, each filter is available to the user via a pulldown menu in the file chooser. The 'All Files' option is always available to the user.

33.88.4.8 filter_value() [1/2]

```cpp
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.
33.88.4.9  filter_value() [2/2]

void Fl_Native_File_Chooser::filter_value (  
    int i  
)  
Sets which filter will be initially selected.  
The first filter is indexed as 0. If filter_value()==filters(), then "All Files" was chosen. If filter_value()>filters(), then a custom filter was set.

33.88.4.10  options()

void Fl_Native_File_Chooser::options (  
    int o  
)  
Sets the platform specific chooser options to val.  
val is expected to be one or more Fl_Native_File_Chooser::Option flags ORed together. Some platforms have OS-specific functions that can be enabled/disabled via this method.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Min</th>
<th>Mac</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_FOLDER</td>
<td>Shows the 'New Folder' button.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREVIEW</td>
<td>Enables the 'Preview' mode by default.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAVEAS_CONFIRM</td>
<td>Confirm dialog if BROWSE_SAVE_FILE file exists.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USE_FILTER_EXT</td>
<td>Chooser filter pilots the output file extension.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

33.88.4.11  preset_file()

void Fl_Native_File_Chooser::preset_file (  
    const char ∗ f  
)  
Sets the default filename for the chooser.  
Use directory() to set the default directory. Mainly used to preset the filename for save dialogs, and on most platforms can be used for opening files as well.

33.88.4.12  show()

int Fl_Native_File_Chooser::show ( )  
Post the chooser’s dialog.  
Blocks until dialog has been completed or cancelled.  

Returns  

• 0 – user picked a file  
• 1 – user cancelled  
• -1 – failed; errmsg() has reason

33.88.4.13  title() [1/2]

const char ∗ Fl_Native_File_Chooser::title ( ) const  
Get the title of the file chooser’s dialog window.  
Return value may be NULL if no title was set.

33.88.4.14  title() [2/2]

void Fl_Native_File_Chooser::title (  
    const char ∗ t  
)  
Set the title of the file chooser’s dialog window.  
Can be NULL if no title desired. The default title varies according to the platform, so you are advised to set the title explicitly.  
The documentation for this class was generated from the following files:

• Fl_Native_File_Chooser.H  
• Fl_Native_File_Chooser.cxx  
• Fl_Native_File_Chooser_GTK.cxx
33.89 Fl_Nice_Slider Class Reference

Inheritance diagram for Fl_Nice_Slider:

```
Fl_Widget
   ↓
Fl_Valuator
   ↓
Fl_Slider
   ↓
Fl_Nice_Slider
```

Public Member Functions

- Fl_Nice_Slider (int X, int Y, int W, int H, const char ∗L=0)

Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Nice_Slider.H
- Fl_Slider.cxx

---

33.90 Fl_Output Class Reference

This widget displays a piece of text.

```cpp
#include <Fl_Output.H>
```

Inheritance diagram for Fl_Output:

```
Fl_Widget
   ↓
Fl_Input_
   ↓
Fl_Input
   ↓
Fl_Output
   ↓
Fl_Multiline_Output
```

Public Member Functions

- Fl_Output (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Output widget using the given position, size, and label string.

Additional Inherited Members

33.90.1 Detailed Description

This widget displays a piece of text.
When you set the value(), Fl_Output does a strcpy() to its own storage, which is useful for program-generated values. The user may select portions of the text using the mouse and paste the contents into other fields or programs.

There is a single subclass, Fl_Multiline_Output, which allows you to display multiple lines of text. Fl_Multiline_Output does not provide scroll bars. If a more complete text editing widget is needed, use Fl_Text_Display instead. The text may contain any characters except \0, and will correctly display anything, using ^X notation for unprintable control characters and \nnn notation for unprintable characters with the high bit set. It assumes the font can draw any characters in the ISO-Latin1 character set.

### 33.90.2 Constructor & Destructor Documentation

#### 33.90.2.1 Fl_Output()

Fl_Output::Fl_Output (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char * l = 0  
)  
Creates a new Fl_Output widget using the given position, size, and label string. The default boxtype is FL_DOWN_BOX. Inherited destructor destroys the widget and any value associated with it. The documentation for this class was generated from the following files:

- Fl_Output.H  
- Fl_Input.cxx

### 33.91 Fl_Overlay_Window Class Reference

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

```c
#include <Fl_Overlay_Window.H>
```

Inheritance diagram for Fl_Overlay_Window:
Public Member Functions

- **Fl_Overlay_Window** ∗ as_overlay_window () FL_OVERRIDE
  
  Return non-null if this is an Fl_Overlay_Window object.

- int can_do_overlay ()

  Returns non-zero if there’s hardware overlay support.

- virtual void draw_overlay ()=0

  You must subclass Fl_Overlay_Window and provide this method.

- void flush () FL_OVERRIDE

  Forces the window to be drawn, this window is also made current and calls draw().

- void hide () FL_OVERRIDE

  Makes a widget invisible.

- void redraw_overlay ()

  Call this to indicate that the overlay data has changed and needs to be redrawn.

- void resize (int, int, int, int) FL_OVERRIDE

  Changes the size or position of the widget.

- void show () FL_OVERRIDE

  Makes a widget visible.

- void show (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)

- ~Fl_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

33.91.1 Detailed Description

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

The overlay is designed to be a rapidly-changing but simple graphic such as a mouse selection box. Fl_Overlay_Window uses the overlay planes provided by your graphics hardware if they are available.

If no hardware support is found the overlay is simulated by drawing directly into the on-screen copy of the double-buffered window, and "erased" by copying the backbuffer over it again. This means the overlay will blink if you change the image in the window.
33.91.2 Constructor & Destructor Documentation

33.91.2.1 Fl_Overlay_Window()

Fl_Overlay_Window::Fl_Overlay_Window ( 
    int X, 
    int Y, 
    int W, 
    int H, 
    const char ∗ l = 0 ) [protected]

Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string. 
If the positions (x,y) are not given, then the window manager will choose them.

33.91.3 Member Function Documentation

33.91.3.1 as_overlay_window()

Fl_Overlay_Window ∗ Fl_Overlay_Window::as_overlay_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Overlay_Window object.
Reimplemented from Fl_Window.

33.91.3.2 draw_overlay()

virtual void Fl_Overlay_Window::draw_overlay ( ) [pure virtual]
You must subclass Fl_Overlay_Window and provide this method.
It is just like a draw() method, except it draws the overlay. The overlay will have already been "cleared" when this is 
called. You can use any of the routines described in <FL/fl_draw.H>.

33.91.3.3 flush()

void Fl_Overlay_Window::flush ( ) [virtual]
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Double_Window.

33.91.3.4 hide()

void Fl_Overlay_Window::hide ( ) [virtual]
Makes a widget invisible.
See also
    show(), visible(), visible_r()
Reimplemented from Fl_Double_Window.

33.91.3.5 redraw_overlay()

void Fl_Overlay_Window::redraw_overlay ( )
Call this to indicate that the overlay data has changed and needs to be redrawn.
The overlay will be clear until the first time this is called, so if you want an initial display you must call this after calling show().
33.91.3.6 resize()

```cpp
void Fl_Overlay_Window::resize (  
    int x,  
    int y,  
    int w,  
    int h ) [virtual]
```

Changes the size or position of the widget. This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the `redraw()` method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.

Some window managers under X11 call `resize()` a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.

`position(X, Y)` is a shortcut for `resize(X, Y, w(), h())`, and `size(W, H)` is a shortcut for `resize(x(), y(), W, H)`.  

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x, y</th>
<th>new position relative to the parent window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>w, h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`

Reimplemented from `Fl_Double_Window`.

33.91.3.7 show()

```cpp
void Fl_Overlay_Window::show ( ) [virtual]
```

Makes a widget visible. An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like `FL_SHOW`.

The `visible()` method returns true if the widget is set to be visible. The `visible_r()` method returns true if the widget and all of its parents are visible. A widget is only visible if `visible()` is true on it and all of its parents.

Changing it will send `FL_SHOW` or `FL_HIDE` events to the widget. Do not change it if the parent is not visible, as this will send false `FL_SHOW` or `FL_HIDE` events to the widget. `redraw()` is called if necessary on this or the parent.

See also

- `hide()`, `visible()`, `visible_r()`

Reimplemented from `Fl_Double_Window`.

The documentation for this class was generated from the following files:

- `Fl_Overlay_Window.H`
- `Fl_Overlay_Window.cxx`

33.92 Fl_Pack Class Reference

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

```cpp
#include <Fl_Pack.H>
```

Inheritance diagram for Fl_Pack:

```
Fl_Widget
   `--- Fl_Group
        `--- Fl_Pack
```

Generated by Doxygen
Public Types

• enum { VERTICAL = 0, HORIZONTAL = 1 }

Public Member Functions

• Fl_Pack (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Pack widget using the given position, size, and label string.
• uchar horizontal () const
  Returns non-zero if Fl_Pack alignment is horizontal.
• void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Override Fl_Group resize behavior.
• int spacing () const
  Gets the number of extra pixels of blank space that are added between the children.
• void spacing (int i)
  Sets the number of extra pixels of blank space that are added between the children.

Protected Member Functions

• void draw () FL_OVERRIDE
  Draws the widget.

Additional Inherited Members

33.92.1 Detailed Description

This widget was designed to add the functionality of compressing and aligning widgets. If type() is Fl_Pack::HORIZONTAL all the children are resized to the height of the Fl_Pack, and are moved next to each other horizontally. If type() is not Fl_Pack::HORIZONTAL then the children are resized to the width and are stacked below each other. Then the Fl_Pack resizes itself to surround the child widgets.

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()

33.92.2 Constructor & Destructor Documentation

33.92.2.1 Fl_Pack()

Fl_Pack::Fl_Pack ( int X, int Y, int W, int H,
int H,
    const char * L = 0 )
Creates a new Fl_Pack widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The default type() is Fl_Pack::VERTICAL.
The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep
a pointer to all the children in the user code. A kludge has been done so the Fl_Pack and all of its children can be
automatic (local) variables, but you must declare the Fl_Pack first, so that it is destroyed last.

Parameters

| in | X, Y | X and Y coordinates (position) |
| in | W, H | width and height, respectively |
| in | L    | label (optional)               |

33.92.3 Member Function Documentation

33.92.3.1 draw()

void Fl_Pack::draw ( ) [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrew as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```
Reimplemented from Fl_Group.

33.92.3.2 horizontal()

uchar Fl_Pack::horizontal ( ) const [inline]
Returns non-zero if Fl_Pack alignment is horizontal.

Returns

non-zero if Fl_Pack alignment is horizontal (Fl_Pack::HORIZONTAL)

Note

Currently the return value is the same as Fl_Group::type(), but this may change in the future. Do not set any
other values than the following with Fl_Pack::type():

- Fl_Pack::VERTICAL (Default)
- Fl_Pack::HORIZONTAL

See class Fl_Pack documentation for details.

33.92.3.3 resize()

void Fl_Pack::resize ( 
    int X,
    int Y,
    int W,
    int H ) [virtual]
Override Fl_Group resize behavior.
Resizing an Fl_Pack will not resize any of its children, but trigger a redraw, which in turn recalculates the dimensions of all children.

Parameters

| in X,Y,W,H | new position and size of the Fl_Pack widget |

Reimplemented from Fl_Group.
The documentation for this class was generated from the following files:

- Fl_Pack.H
- Fl_Pack.cxx

## 33.93 Fl_Paged_Device Class Reference

Represents page-structured drawing surfaces.

```c
#include <Fl_Paged_Device.H>
```

Inheritance diagram for Fl_Paged_Device:

```
Fl_Surface_Device
  └── Fl_Paged_Device
      ├── Fl_Widget_Surface
      │    └── Fl_Surface_Device
      │        └── Fl_Paged_Device
      ├── Fl_PostScript_File_Device
      │    └── Fl_Printer
```

### Classes

- struct page_format
  
  width, height and name of a page format

### Public Types

- enum Page_Format {
  A0 = 0, A1, A2, A3,
  A4, A5, A6, A7,
  A8, A9, B0, B1,
  B2, B3, B4, B5,
  B6, B7, B8, B9,
  B10, C5E, DLE, EXECUTIVE,
  FOLIO, LEDGER, LEGAL, LETTER,
  TABLOID, ENVELOPE, MEDIA = 0x1000 }

  Possible page formats.

- enum Page_Layout { PORTRAIT = 0, LANDSCAPE = 0x100, REVERSED = 0x200, ORIENTATION = 0x300 }

  Possible page layouts.

### Public Member Functions

- 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**

33.93.1 Detailed Description

Represents page-structured drawing surfaces.
This class has no public constructor: don't instantiate it; use Fl_Printer or Fl_PostScript_File_Device instead.

33.93.2 Member Enumeration Documentation

33.93.2.1 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>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A0 format.
33.93.2  Page_Layout

enum Fl_Paged_Device::Page_Layout

Possible page layouts.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>

33.93.3  Member Function Documentation

33.93.3.1  begin_job()

int Fl_Paged_Device::begin_job (  
    int pagecount = 0,  
    int * frompage = NULL,  
    int * toppage = NULL,  
    char ** perr_message = NULL ) [virtual]
Begins a print job.

Parameters

| in  | pagecount | the total number of pages of the job (or 0 if you don't know the number of pages) |
| out | frompage  | if non-null, *frompage is set to the first page the user wants printed           |
| out | topage    | if non-null, *topage is set to the last page the user wants printed              |
| out | perr_message | 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. |

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

33.93.3.2 begin_page()

int Fl_Paged_Device::begin_page (   
    void  )  [virtual]

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

33.93.3.3 end_job()

void Fl_Paged_Device::end_job (    
    void  )  [virtual]

To be called at the end of a print job.
Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

33.93.3.4 end_page()

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

d 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().

Generated by Doxygen
Returns

0 if OK, non-zero if any error.

Reimplemented in `Fl_PDF_File_Device` and `Fl_Printer`.

### 33.93.3.5 margins()

```cpp
void Fl_Paged_Device::margins (  
  int * left,  
  int * top,  
  int * right,  
  int * bottom ) [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by `scale()` calls.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>left</th>
<th>If non-null, <code>*left</code> is set to the left margin size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>top</td>
<td>If non-null, <code>*top</code> is set to the top margin size.</td>
</tr>
<tr>
<td>out</td>
<td>right</td>
<td>If non-null, <code>*right</code> is set to the right margin size.</td>
</tr>
<tr>
<td>out</td>
<td>bottom</td>
<td>If non-null, <code>*bottom</code> is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented in `Fl_PDF_File_Device` and `Fl_Printer`.

### 33.93.3.6 rotate()

```cpp
void Fl_Paged_Device::rotate (  
  float angle ) [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive `rotate()` calls don’t combine their effects.

**Parameters**

| angle | Rotation angle in counter-clockwise degrees. |

Reimplemented in `Fl_PDF_File_Device` and `Fl_Printer`.

### 33.93.3.7 scale()

```cpp
void Fl_Paged_Device::scale (  
  float scale_x,  
  float scale_y = 0. ) [virtual]
```

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a `scale()` call, do a `printable_rect()` call to get the new dimensions of the printable page area. Successive `scale()` calls don’t combine their effects.

**Parameters**

<table>
<thead>
<tr>
<th>scale</th>
<th>Horizontal dimensions of plot are multiplied by this quantity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</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>

Generated by Doxygen
Reimplemented in Fl_PostScript_File_Device, and Fl_Printer.

### 33.93.3.8 start_job()

```cpp
def start_job (int pagecount = 0, int * frompage = NULL, int * topage = NULL, char ** perr_message = NULL) [inline]
```

Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).
For API compatibility with FLTK 1.3.x

### 33.93.3.9 start_page()

```cpp
def start_page () [inline]
```

Synonym of begin_page().
For API compatibility with FLTK 1.3.x

The documentation for this class was generated from the following files:

- Fl_Paged_Device.H
- Fl_Paged_Device.hxx

### 33.94 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
  \|-- Fl_Image
    \|-- Fl_GIF_Image
    \|-- Fl_XPM_Image
    \|-- Fl_Anim_GIF_Image
```

**Public Member Functions**

- **void color_average (Fl_Color c, float i)** FL_OVERRIDE
  
  The color_average() method averages the colors in the image with the provided FLTK color value.

- **Fl_Image * copy ()** const

  Creates a resized copy of the image.

- **void desaturate ()** FL_OVERRIDE
  
  The desaturate() method converts an image to grayscale.

- **void draw (int X, int Y)**

  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.

Generated by Doxygen
• Fl_Pixmap (const uchar ∗ const ∗ D)
The constructors create a new pixmap from the specified XPM data.

• Fl_Pixmap (uchar ∗ const ∗ D)
The constructors create a new pixmap from the specified XPM data.

• void label (Fl_Menu_Item ∗ m) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a menu item.

• void label (Fl_Widget ∗ w) FL_OVERRIDE
  This method is an obsolete way to set the image attribute of a widget or menu item.

• void uncache () FL_OVERRIDE
  If the image has been cached for display, delete the cache data.

• virtual ∼Fl_Pixmap ()
The destructor frees all memory and server resources that are used by the pixmap.

Public Attributes
• int alloc_data

Protected Member Functions
• void measure ()

Friends
• class Fl_Graphics_Driver

Additional Inherited Members

33.94.1 Detailed Description
The Fl_Pixmap class supports caching and drawing of colormap ( pixmap) images, including transparency.

33.94.2 Constructor & Destructor Documentation

33.94.2.1 Fl_Pixmap() [1/4]
Fl_Pixmap::Fl_Pixmap (char ∗ const ∗ D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.

33.94.2.2 Fl_Pixmap() [2/4]
Fl_Pixmap::Fl_Pixmap (uchar ∗ const ∗ D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.

33.94.2.3 Fl_Pixmap() [3/4]
Fl_Pixmap::Fl_Pixmap (const char ∗ const ∗ D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.
33.94 Fl_Pixmap Class Reference

33.94.2.4 Fl_Pixmap() [4/4]

Fl_Pixmap::Fl_Pixmap (const uchar ∗ ∗ D) [inline], [explicit]
The constructors create a new pixmap from the specified XPM data.

33.94.3 Member Function Documentation

33.94.3.1 color_average()

void Fl_Pixmap::color_average (Fl_Color c, float i) [virtual]
The color_average() method averages the colors in the image with the 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.

33.94.3.2 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:

- \( w() == \text{data}_w() == W \)
- \( h() == \text{data}_h() == H \)

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, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Parameters

\[ \text{in} \quad W, H \quad \text{Requested width and height of the new image} \]

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 inheription 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.
33.94.3.3 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.

33.94.3.4 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.

33.94.3.5 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.

33.94.3.6 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.

33.94.3.7 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
33.95 Fl_Plugin Class Reference

Fl_Plugin allows link-time and run-time integration of binary modules.  
#include <Fl_Plugin.H>

Inheritance diagram for Fl_Plugin:

```
Fl_Plugin

Fl_Device_Plugin
```

Public Member Functions

- **Fl_Plugin (const char *klass, const char *name)**
  
  Create a plugin.

- virtual **~Fl_Plugin ()**

  Clear the plugin and remove it from the database.

33.95.1 Detailed Description

Fl_Plugin allows link-time and run-time integration of binary modules. Fl_Plugin and Fl_Plugin_Manager provide a small and simple solution for linking C++ classes at run-time, or optionally linking modules at compile time without the need to change the main application. Fl_Plugin_Manager uses static initialisation to create the plugin interface early during startup. Plugins are stored in a temporary database, organized in classes.

Plugins should derive a new class from Fl_Plugin as a base:
```
class My_Plugin : public Fl_Plugin {
public:
  My_Plugin() : Fl_Plugin("effects", "blur") { }
  void do_something(...) { }
};
My_Plugin blur_plugin();
```

Plugins can be put into modules and either linked before distribution, or loaded from dynamically linkable files. An Fl_Plugin_Manager is used to list and access all currently loaded plugins.
```
Fl_Plugin_Manager mgr("effects");
int i, n = mgr.plugins();
for (i=0; i<n; i++) {
  My_Plugin *pin = (My_Plugin*)mgr.plugin(i);
  pin->do_something();
}
```

33.95.2 Constructor & Destructor Documentation

33.95.2.1 Fl_Plugin()

Fl_Plugin::Fl_Plugin (
    const char * klass,
    const char * name )

Create a plugin.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>klass</th>
<th>plugins are grouped in classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>every plugin should have a unique name</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Plugin.H
33.96  Fl_Plugin_Manager Class Reference

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

```cpp
#include <Fl_Plugin.H>
```

Inheritance diagram for Fl_Plugin_Manager:

```
Fl_Preferences

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Plugin_Manager</td>
</tr>
</tbody>
</table>
```

### 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 class.

- `~Fl_Plugin_Manager()`
  
  Remove the plugin manager.

### Static Public Member Functions

- `static int load` (const char *filename)
  
  Load a module from disk.

- `static int loadAll` (const char *filepath, const char *pattern=0)
  
  Use this function to load a whole directory full of modules.

- `static void removePlugin` (Fl_Preferences::ID id)
  
  Remove any plugin.

### Additional Inherited Members

33.96.1  Detailed Description

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

See also

Fl_Plugin

33.96.2  Constructor & Destructor Documentation

33.96.2.1  `~Fl_Plugin_Manager()`

Fl_Plugin_Manager::~Fl_Plugin_Manager()

Remove the plugin manager.

Calling this does not remove the database itself or any plugins. It just removes the reference to the database.
33.96.3 Member Function Documentation

33.96.3.1 addPlugin()

Fl_Preferences::ID Fl_Plugin_Manager::addPlugin (  
    const char * name,  
    Fl_Plugin * plugin )

This function adds a new plugin to the database.  
There is no need to call this function explicitly. Every Fl_Plugin constructor will call this function at initialization time.

33.96.3.2 load()

int Fl_Plugin_Manager::load (  
    const char * filename ) [static]

Load a module from disk.  
A module must be a dynamically linkable file for the given operating system. When loading a module, its +init function will be called which in turn calls the constructor of all statically initialized Fl_Plugin classes and adds them to the database.

33.96.3.3 removePlugin()

void Fl_Plugin_Manager::removePlugin (  
    Fl_Preferences::ID id ) [static]

Remove any plugin.  
There is no need to call this function explicitly. Every Fl_Plugin destructor will call this function at destruction time. The documentation for this class was generated from the following files:

- Fl_Plugin.H
- Fl_Preferences.cxx

33.97 Fl_PNG_Image Class Reference

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

# include <Fl_PNG_Image.H>

Inheritance diagram for Fl_PNG_Image:

```
Fl_PNG_Image
   |        |
   |        |
   v        v
Fl_RGB_Image
   |        |
   |        |
   v        v
Fl_Image
```

Public Member Functions

- **Fl_PNG_Image** (const char *filename)

  The constructor loads the named PNG image from the given png filename.

- **Fl_PNG_Image** (const char *name_png, const unsigned char *buffer, int datasize)

  Constructor that reads a PNG image from memory.

Friends

- class Fl_ICO_Image

Generated by Doxygen
**Additional Inherited Members**

**33.97.1 Detailed Description**

The `Fl_PNG_Image` class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files. The class loads colormapped and full-color images and handles color- and alpha-based transparency.

**33.97.2 Constructor & Destructor Documentation**

**33.97.2.1 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**

| in | filename | Name of PNG file to read |

**33.97.2.2 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 |
| maxsize  | Size in bytes of the memory buffer containing the PNG image |

The documentation for this class was generated from the following files:

- `Fl_PNG_Image.H`
- `Fl_PNG_Image.cxx`

**33.98 Fl_PNM_Image Class Reference**

The `Fl_PNM_Image` class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.

```cpp
#include <Fl_PNM_Image.H>
```

Inheritance diagram for Fl_PNM_Image:
Public Member Functions

- **Fl_PNM_Image** (const char *filename)
  
  *The constructor loads the named PNM image.*

Additional Inherited Members

33.98.1 Detailed Description

The **Fl_PNM_Image** class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.

The class loads bitmap, grayscale, and full-color images in both ASCII and binary formats.

33.98.2 Constructor & Destructor Documentation

33.98.2.1 **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**

- **filename** a full path and name pointing to a valid jpeg file.

The documentation for this class was generated from the following files:

- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

33.99 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) FL_OVERRIDE
  
  Handles the specified event.

- int **value** (double, double)
  
  Returns the current position in x and y.

- void **xbounds** (double, double)
  
  Sets the X axis bounds.

- double **xmaximum** () const
  
  Gets the X axis maximum.

- void **xmaximum** (double a)
  
  Same as xbounds(xminimum(), a)

- double **xminimum** () const
  
  Gets the X axis minimum.

- void **xminimum** (double a)
  
  Same as xbounds(a, xmaximum())

- void **xstep** (double a)
  
  Sets the stepping value for the X axis.

- double **xvalue** () const
  
  Gets the X axis coordinate.

- int **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** () FL_OVERRIDE
  
  Draws the widget.

- void **draw** (int, int, int, int)

- int **handle** (int, int, int, int)
Additional Inherited Members

33.99.1 Detailed Description

This class is provided for Forms compatibility. It provides 2D input. It would be useful if this could be put atop another widget so that the crosshairs are on top, but this is not implemented. The color of the crosshairs is `selection_color()`.

![Figure 33.31 Fl_Positioner](image)

33.99.2 Constructor & Destructor Documentation

33.99.2.1 Fl_Positioner()

```cpp
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`.

33.99.3 Member Function Documentation

33.99.3.1 draw()

```cpp
void Fl_Positioner::draw ( ) [protected], [virtual]
```

Draws the widget. Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead. Override this function to draw your own widgets. If you ever need to call another widget's draw method from within your own `draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements `Fl_Widget`.

33.99.3.2 handle()

```cpp
int Fl_Positioner::handle ( 
    int event ) [virtual]
```

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
Parameters

| in | event | the kind of event received |

Return values

| 0 | if the event was not used or understood |
| 1 | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Positioner.H
- Fl_Positioner.cxx

33.100 Fl_PostScript_File_Device Class Reference

To send graphical output to a PostScript file.

#include <Fl_PostScript.H>

Inheritance diagram for Fl_PostScript_File_Device:

```
Fl_Surface_Device
  |
  v
Fl_Widget_Surface
  |
  v
Fl_Paged_Device
  |
  v
FlPostScript_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) FL_OVERRIDE
  
  Begins the session where all graphics requests will go to FILE pointer.
- int begin_job (int pagecount, int *from, int *to, char **perr_message) FL_OVERRIDE
  
  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) FL_OVERRIDE
  
  Begins the session where all graphics requests will go to a local PostScript file.
- int begin_page (void) FL_OVERRIDE
  
  Begins a new printed page.
- void close_command (Fl_PostScript_Close_Command cmd)
  
  Sets the function end_job() calls to close the file()
- void end_current () FL_OVERRIDE
  
  FLTK calls this each time a surface ceases to be the current drawing surface.
- void end_job (void) FL_OVERRIDE
  
  Completes all PostScript output.
- `int end_page (void) FL_OVERRIDE`
  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) FL_OVERRIDE`
  Computes the dimensions of margins that lie between the printable page area and the full page.

- `void origin (int *x, int *y) FL_OVERRIDE`
  Computes the coordinates of the current origin of graphics functions.

- `void origin (int x, int y) FL_OVERRIDE`
  Sets the position of the origin of graphics in the drawable part of the drawing surface.

- `int printable_rect (int *w, int *h) FL_OVERRIDE`
  Computes the width and height of the drawable area of the drawing surface.

- `void rotate (float angle) FL_OVERRIDE`
  Rotates the graphics operations relatively to paper.

- `void scale (float scale_x, float scale_y=0.) FL_OVERRIDE`
  Changes the scaling of page coordinates.

- `void set_current () FL_OVERRIDE`
  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) FL_OVERRIDE`
  Translates the current graphics origin accounting for the current rotation.

- `void untranslate (void) FL_OVERRIDE`
  Undoes the effect of a previous `translate()` call.

- `~Fl_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

#### 33.100.1 Detailed Description

To send graphical output to a PostScript file.

This class is used exactly as the `Fl_Printer` class except for the `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_Imag e 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>
<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>U+201A</td>
<td>quotesingle</td>
<td>trademark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>U+02C6</td>
<td>circumflex</td>
<td>U+201C</td>
<td>quotedblleft</td>
<td>partialleft</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>U+02C7</td>
<td>caron</td>
<td>U+201D</td>
<td>quotedblright</td>
<td>partialright</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>°</td>
<td>U+02D8</td>
<td>breve</td>
<td>U+201E</td>
<td>quotedblbase</td>
<td>summation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>′</td>
<td>U+02D9</td>
<td>dotaccent</td>
<td>U+2020</td>
<td>dagger</td>
<td>U+221A</td>
<td>radical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>‾</td>
<td>U+02DA</td>
<td>ring</td>
<td>U+2021</td>
<td>daggerdbl</td>
<td>U+221E</td>
<td>infinity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>‼</td>
<td>U+02DB</td>
<td>ogonek</td>
<td>U+2022</td>
<td>bullet</td>
<td>U+2260</td>
<td>notequal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>U+02DC</td>
<td>tilde</td>
<td>U+2026</td>
<td>ellipsis</td>
<td>U+2264</td>
<td>lessequal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>″</td>
<td>U+02DD</td>
<td>hungarumlaut</td>
<td>U+2030</td>
<td>perthousand</td>
<td>U+2265</td>
<td>greaterequal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>‒</td>
<td>U+2013</td>
<td>endash</td>
<td>U+2039</td>
<td>guillemotleft</td>
<td>U+25CA</td>
<td>lozenge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>U+2014</td>
<td>emdash</td>
<td>U+203A</td>
<td>guillemotright</td>
<td>fi</td>
<td>U+FB01</td>
<td>fi</td>
<td></td>
</tr>
<tr>
<td>′</td>
<td>U+2018</td>
<td>quotellip</td>
<td>/</td>
<td>U+2044</td>
<td>fraction</td>
<td>fl</td>
<td>U+FB02</td>
<td>fi</td>
</tr>
<tr>
<td>′</td>
<td>U+2019</td>
<td>quoteright</td>
<td>€</td>
<td>U+20AC</td>
<td>Euro</td>
<td>U+F8FF</td>
<td>apple (Mac OS only)</td>
<td></td>
</tr>
</tbody>
</table>

### 33.100.2 Member Function Documentation

#### 33.100.2.1 begin_job() [1/3]

```c
int Fl_PostScript_File_Device::begin_job ( FILE * ps_output,
   int pagecount = 0,
   enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
   enum Fl_Paged_Device::Page_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>ps_output</th>
<th>A writable FILE pointer that will receive PostScript output and that should not be closed until after end_job() has been called.</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

always 0.
33.100.2.2 begin_job() [2/3]

```cpp
int Fl_PostScript_File_Device::begin_job (  
    int pagecount,  
    int * from,  
    int * to,  
    char ** perr_message ) [virtual]
```

Don't use with this class.
Reimplemented from Fl_Paged_Device.

33.100.2.3 begin_job() [3/3]

```cpp
int Fl_PostScript_File_Device::begin_job (  
    int pagecount = 0,  
    enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,  
    enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to a local PostScript file.
Opens a file dialog 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.

33.100.2.4 begin_page()

```cpp
int Fl_PostScript_File_Device::begin_page (  
    void ) [virtual]
```

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
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.

33.100.2.5 end_current()

```cpp
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.

33.100.2.6 end_job()

void Fl_PostScript_File_Device::end_job {
    void     [virtual]
Completes all PostScript output.
This also closes with fclose() the underlying file() unless close_command() was used to set another function.
Reimplemented from Fl_Paged_Device.

33.100.2.7 end_page()

int Fl_PostScript_File_Device::end_page {
    void     [virtual]
To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().
Note
end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page()
open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to
Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns
0 if OK, non-zero if any error.
Reimplemented from Fl_Paged_Device.

33.100.2.8 margins()

void Fl_PostScript_File_Device::margins {
    int     * left,
    int     * top,
    int     * right,
    int     * bottom   [virtual]
Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters
<table>
<thead>
<tr>
<th>out</th>
<th>left</th>
<th>If non-null, *left is set to the left margin size.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>top</td>
<td>If non-null, *top is set to the top margin size.</td>
</tr>
<tr>
<td>out</td>
<td>right</td>
<td>If non-null, *right is set to the right margin size.</td>
</tr>
<tr>
<td>out</td>
<td>bottom</td>
<td>If non-null, *bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

33.100.2.9 origin() [1/2]

void Fl_PostScript_File_Device::origin {

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.

### 33.100.2.10 origin() [2/2]

```cpp
def 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.

### 33.100.2.11 printable_rect()

```cpp
def 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.

### 33.100.2.12 rotate()

```cpp
def 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.
33.100.2.13  scale()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale-_x</td>
<td>Horizontal dimensions of plot are multiplied by this quantity.</td>
</tr>
<tr>
<td>scale-_y</td>
<td>Same as above, vertically. The value 0. is equivalent to setting scale_y = scale_x. Thus, scale(factor); is equivalent to scale(factor, factor);</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

33.100.2.14  set_current()

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.

33.100.2.15  start_job() [1/2]

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

33.100.2.16  start_job() [2/2]

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

33.100.2.17 translate()

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.

33.100.2.18 untranslate()

void Fl_PostScript_File_Device::untranslate ( 
    void ) [virtual]

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following file:
   • Fl_PostScript.H

33.101 Fl_Preferences Class Reference

Fl_Preferences store user settings between application starts.
#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, Fl_String &value, const Fl_String &defaultValue)**
  
  Reads an entry from the group.

- **char get (const char * entry, float &value, float defaultValue)**
  
  Reads an entry from the group.

- **char get (const char * entry, int &value, int defaultValue)**
  
  Reads an entry from the group.
• char get (const char *entry, void *value, const void *defaultValue, int defaultSize)
  Reads an entry from the group.
• char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int size)
  Reads a binary entry from the group, encoded in hexadecimal blocks.
• char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)
  Reads a binary entry from the group, encoded in hexadecimal blocks.
• char get_userdata_path (char *path, int pathlen)
  Creates a path that is related to the preference file and that is usable for additional application data.
• const char *group (int num_group)
  Returns the name of the Nth (num_group) group.
• char group_exists (const char *key)
  Returns non-zero if a group with this name exists.
• int groups ()
  Returns the number of groups that are contained within a group.
• ID id ()
  Return an ID that can later be reused to open more references to this dataset.
• const char *name ()
  Return the name of this entry.
• const char *path ()
  Return the full path to this entry.
• char set (const char *entry, const char *value)
  Sets an entry (name/value pair).
• char set (const char *entry, const Fl_String &value)
  Sets an entry (name/value pair).
• char set (const char *entry, const void *value, int size)
  Sets an entry (name/value pair).
• char set (const char *entry, double value)
  Sets an entry (name/value pair).
• char set (const char *entry, double value, int precision)
  Sets an entry (name/value pair).
• char set (const char *entry, float value)
  Sets an entry (name/value pair).
• char set (const char *entry, float value, int precision)
  Sets an entry (name/value pair).
• char set (const char *entry, int value)
  Sets an entry (name/value pair).
• int size (const char *entry)
  Returns the size of the value part of an entry.
• virtual ~Fl_Preferences ()
  The destructor removes allocated resources.

Static Public Member Functions

• static unsigned int file_access ()
  Return the current file access permissions for the FLTK preferences system.
• static void file_access (unsigned int flags)
  Tell the FLTK preferences system which files in the file system it may read, create, or write.
• static Root filename (char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application)
  Determine the file name and path to preferences that would be openended 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, or 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

33.101.1 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(Root root, const char ∗vendor, const char ∗application)

As a special case, Fl_Preferences can be memory mapped and not be associated with a file on disk.

See also

Fl_Preferences::Fl_Preferences(Fl_Preferences ∗parent, const char ∗group) for more details on memory mapped preferences.

Note

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see Fl_Preferences::Fl_Preferences(Root, const char ∗vendor, const char ∗application) 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.

33.101.2 Member TypeDef Documentation

33.101.2.1 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.
33.101.3 Member Enumeration Documentation

33.101.3.1 Root

debug Fl_Preferences::Root
Define the scope of the preferences.

<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>

33.101.4 Constructor & Destructor Documentation

33.101.4.1 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

<table>
<thead>
<tr>
<th>in</th>
<th>root</th>
<th>can be <code>USER_L</code> or <code>SYSTEM_L</code> for user specific or system wide preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>in</td>
<td>application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>

See also

`Fl_Preferences(Fl_Preferences *parent, const char *group)` with `parent` set to `NULL`.

### Fl_Preferences() [2/7]

```cpp
Fl_Preferences::Fl_Preferences (  
  const char * path,  
  const char * vendor,  
  const char * application )
```

Use this constructor to create or read a 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

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>path to the directory that contains the preference file</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vendor</td>
<td>unique text describing the company or author of this file, must be a valid filepath segment</td>
</tr>
<tr>
<td>in</td>
<td>application</td>
<td>unique text describing the application, must be a valid filepath segment</td>
</tr>
</tbody>
</table>
33.101.4.3 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>

33.101.4.4 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

<table>
<thead>
<tr>
<th>in</th>
<th>parent</th>
<th>the parameter parent is a pointer to the parent group. If parent is NULL, the new preferences item refers to an application internal database (&quot;runtime prefs&quot;) 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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>group</td>
<td>a group name that is used as a key into the database</td>
</tr>
</tbody>
</table>

See also

Fl_Preferences( Fl_Preferences&, const char *group )

33.101.4.5 Fl_Preferences() [5/7]

Fl_Preferences::Fl_Preferences ( 

   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>
Fl_Preferences & parent,
   int groupIndex)

Open a child group using a given index.
Use the groupIndex argument to find the group that you would like to access. If the given index is invalid
(negative or too high), a new group is created with a UUID as a name.
The index needs to be fixed. It is currently backward. Index 0 points to the last member in the 'list' of preferences.

Parameters

<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>groupIndex</td>
<td>zero based index into child groups</td>
</tr>
</tbody>
</table>

33.101.4.6 Fl_Preferences() [6/7]

Fl_Preferences::Fl_Preferences (  
   Fl_Preferences * parent,
   int groupIndex )

See also

   Fl_Preferences( Fl_Preferences&, int groupIndex )

33.101.4.7 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.

33.101.4.8 ~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.

33.101.5 Member Function Documentation

33.101.5.1 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

| in  | key | name of entry to delete |

Generated by Doxygen
Returns

0 if deleting the entry failed

33.101.5.2 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

33.101.5.3 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 f there is an internal database error.

33.101.5.4 entries()

int Fl_Preferences::entries ( )

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

33.101.5.5 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 |
33.101.5.6 entry_exists()

```cpp
char Fl_Preferences::entry_exists ( const char * key )
```

Returns non-zero if an entry with this name exists.

**Parameters**

- **key**: name of entry that is searched for

**Returns**

0 if entry was not found

33.101.5.7 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)

33.101.5.8 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()

33.101.5.9 filename() [1/2]

```cpp
Fl_PREFERENCES::Root Fl_PREFERENCES::filename ( char * buffer, size_t buffer_size )
```

Generated by Doxygen
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.

33.101.5.10 filename() [2/2]

Fl_Preferences::Root Fl_Preferences::filename {
    char * buffer,
    size_t buffer_size,
    Root root,
    const char * vendor,
    const char * application } [static]

Determine the file name and path to preferences that would be openend with these parameters. Find the possible location of a preference file on disk without touching any of the pathname components. This can be used to check if a preference file already exists.

Parameters

<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 )

33.101.5.11 flush()

int Fl_Preferences::flush ( )

Writes preferences to disk if they were modified. This method can be used to verify that writing a preference file went well. Deleting the base preferences object will also write the contents of the database to disk.

Returns

-1 if anything went wrong, i.e. file could not be opened, permissions blocked writing, etc.
0 if the file was written to disk. This does not check if the disk ran out of space and the file is truncated.
1 if no data was written to the database and no write attempt to disk was made.
33.101.5.12 get() [1/9]

```cpp
char Fl_Preferences::get (  
    const char * key,  
    char *& text,  
    const char * defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was
used (0). `get()` allocates memory of sufficient size to hold the value. The buffer must be 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

33.101.5.13 get() [2/9]

```cpp
char Fl_Preferences::get (  
    const char * key,  
    char * text,  
    const char * defaultValue,  
    int maxSize )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was
used (0). 'maxSize' is the maximum length of text that will be read. The text buffer must allow for one additional byte
for a trailing zero.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>text</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
<tr>
<td>in</td>
<td>maxSize</td>
<td>maximum length of value plus one byte for a trailing zero</td>
</tr>
</tbody>
</table>

**Returns**

0 if the default value was used

33.101.5.14 get() [3/9]

```cpp
char Fl_Preferences::get (  
    const char * key,  
    double & value,  
    double defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was
used (0).
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

### Returns

0 if the default value was used

#### 33.101.5.15 get() [4/9]

```cpp
char Fl_Preferences::get (
    const char * key,
    Fl_String & value,
    const Fl_String & 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

#### 33.101.5.16 get() [5/9]

```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

33.101.5.17 get() [6/9]

```c
char Fl_Preferences::get (  
    const char * key,  
    int & value,  
    int defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>returned from preferences or default value if none was set</td>
</tr>
<tr>
<td>in</td>
<td>defaultValue</td>
<td>default value to be used if no preference was set</td>
</tr>
</tbody>
</table>

Returns

0 if the default value was used

33.101.5.18 get() [7/9]

```c
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

33.101.5.19 get() [8/9]

```c
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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
<td>name of entry</td>
</tr>
<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

33.101.5.20 get() [9/9]

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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
<td>name of entry</td>
</tr>
<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 )

33.101.5.21 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:
```cpp
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( path, FL_PATH_MAX );
```
creates the preferences database in the directory (User 'matt' on Linux):
/Users/matt/.fltk/matthiasm.com/test.prefs
..and returns the userdata path:
/Users/matt/.fltk/matthiasm.com/test/

**Parameters**

<table>
<thead>
<tr>
<th>out path</th>
<th>buffer for user data path</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pathlen</td>
<td>size of path buffer (should be at least FL_PATH_MAX )</td>
</tr>
</tbody>
</table>

**Returns**

1 if there is no filename (path will be unmodified)
1 if pathlen is 0 (path will be unmodified)
1 if a path was created successfully, path will contain the path name ending in a '
0 if path was not created for some reason; path will contain the pathname that could not be created

See also

`Fl_Preferences::Fl_Preferences(Root, const char*, const char*)`

### 33.101.5.22 group()

```cpp
const char * Fl_Preferences::group ( int num_group )
```

Returns the name of the Nth (`num_group`) group.
There is no guaranteed order of group names. The index must be within the range given by groups().

**Parameters**

| in num_group | number indexing the requested group |

**Returns**

'C' string pointer to the group name

### 33.101.5.23 group_exists()

```cpp
char Fl_Preferences::group_exists ( const char * key )
```

Returns non-zero if a group with this name exists.
Group names are relative to the `Fl_Preferences` node and can contain a path. "." describes the current node, ".//" describes the topmost node. By preceding a groupname with a ".//" its path becomes relative to the topmost node.

**Parameters**

| in key | name of group that is searched for |

Generated by Doxygen
Returns

0 if no group by that name was found

33.101.5.24  groups()

int Fl_Preferences::groups ( )
Returns the number of groups that are contained within a group.

Returns

0 for no groups at all

33.101.5.25  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!

33.101.5.26  set() [1/8]

char Fl_Preferences::set ( const char ∗ key,
                        const char ∗ text )
Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value
was actually stored in the 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

33.101.5.27  set() [2/8]

char Fl_Preferences::set ( const char ∗ entry,
                        const Fl_String & 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>entry</th>
<th>name of entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>set this entry to value (stops at the first nul character).</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

33.101.5.28 set() [3/8]

```cpp
class 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

33.101.5.29 set() [4/8]

```cpp
class 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

33.101.5.30 set() [5/8]

```cpp
class Fl_Preferences::set {
    const char * key,
    long 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
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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

33.101.5.31 set() [6/8]

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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
</tr>
</tbody>
</table>

Returns

0 if setting the value failed

33.101.5.32 set() [7/8]

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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>key</td>
</tr>
<tr>
<td>in</td>
<td>value</td>
</tr>
<tr>
<td>in</td>
<td>precision</td>
</tr>
</tbody>
</table>
Returns

0 if setting the value failed

33.101.5.33  set() [8/8]

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

33.101.5.34  size()

int Fl_Preferences::size (  
    const char * key )

Returns the size of the value part of an entry.

Parameters

| in | key | name of entry |

Returns

size of value

33.101.6  Member Data Documentation

33.101.6.1  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.

33.101.6.2  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.

33.101.6.3  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

33.102 Fl_Printer Class Reference

OS-independent print support.

#include <Fl_Printer.H>

Inheritance diagram for Fl_Printer:

```
Fl_Surface_Device
  Fl_Paged_Device
    Fl_Widget_Surface
      Fl_Printer
```

Public Member Functions

- int `begin_job` (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL) FL_OVERRIDE
  Begins a print job.
- int `begin_page` (void) FL_OVERRIDE
  Begins a new printed page.
- void `end_job` (void) FL_OVERRIDE
  To be called at the end of a print job.
- int `end_page` (void) FL_OVERRIDE
  To be called at the end of each page.
- Fl_Printer (void) FL_OVERRIDE
  The constructor.
- bool `is_current` () FL_OVERRIDE
  Is this surface the current drawing surface?
- void `margins` (int *left, int *top, int *right, int *bottom) FL_OVERRIDE
  Computes the dimensions of margins that lie between the printable page area and the full page.
- void `origin` (int *x, int *y) FL_OVERRIDE
  Computes the coordinates of the current origin of graphics functions.
- void `origin` (int x, int y) FL_OVERRIDE
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int `printable_rect` (int *w, int *h) FL_OVERRIDE
  Computes the width and height of the drawable area of the drawing surface.
- void `rotate` (float angle) FL_OVERRIDE
  Rotates the graphics operations relatively to paper.
- void `scale` (float scale_x, float scale_y=0.) FL_OVERRIDE
  Changes the scaling of page coordinates.
• void \texttt{set\_current} (void) \texttt{FL\_OVERRIDE}
  
  \textit{Make this surface the current drawing surface.}

• void \texttt{translate} (int \(x\), int \(y\)) \texttt{FL\_OVERRIDE}
  
  \textit{Translates the current graphics origin accounting for the current rotation.}

• void \texttt{untranslate} (void) \texttt{FL\_OVERRIDE}
  
  \textit{Undoes the effect of a previous translate() call.}

• \texttt{\~\textbackslash{\textit{F}l\_P}rinter} (void)
  
  The destructor.

\section*{Static Public Attributes}

These attributes are useful for the Linux/Unix platform only.

• static const char * \texttt{dialog\_title} = "Print"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_printer} = "Printer:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_range} = "Print Range"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_copies} = "Copies"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_all} = "All"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_pages} = "Pages"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_from} = "From:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_to} = "To:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_properties} = "Properties..."
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_copyNo} = "# Copies:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_print\_button} = "Print"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_cancel\_button} = "Cancel"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{dialog\_print\_to\_file} = "Print To File"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_title} = "Printer Properties"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_pagesize} = "Page Size:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_mode} = "Output Mode:"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_use} = "Use"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_save} = "Save"
  
  \textit{[this text may be customized at run-time]}

• static const char * \texttt{property\_cancel} = "Cancel"
  
  \textit{[this text may be customized at run-time]}

Generated by Doxygen
Additional Inherited Members

33.102.1 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 with begin_job(), begin_page(), printable_rect() and origin() calls and finish by end_page() and end_job() calls.

Example of use: print a widget centered in a page

```cpp
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>

int width, height;
Fl_Widget *widget = ... // a widget we want printed
Fl_Printer *printer = new Fl_Printer();
if (printer->begin_job(1) == 0) {
    printer->begin_page();
    printer->printable_rect(&width, &height);
    fl_color(FL_BLACK);
    fl_line_style(FL_SOLID, 2);
    fl_rect(0, 0, width, height);
    fl_font(FL_COURIER, 12);
    time_t now; time(&now); fl_draw(ctime(&now), 0, fl_height());
    printer->origin(width/2, height/2);
    printer->print_widget(widget, -widget->w()/2, -widget->h()/2);
    printer->end_page();
    printer->end_job();
}
delete printer;
```

Recommended method to refresh GUI while printing:

```cpp
printer->begin_job(0);
......
Fl_Surface_Device::push_current(Fl_Display_Device::display_device());
Fl::check(); // or any operation that draws to display
Fl_Surface_Device::pop_current();
......
printer->end_job();
```

Platform specifics

- X11 and Wayland platforms:
  - FLTK expresses all graphics data using (Level 2) PostScript and sends that to the selected printer. See class **Fl_PostScript_File_Device** for a description of how text and transparent images appear in print.
  - If the GTK library is available at run-time, class **Fl_Printer** runs GTK's printer dialog which allows to set printer, paper size and orientation.
  - Under the KDE desktop, **Fl_Printer** runs the `kdialog` command to create KDE-styled file dialogs if that command is available at run-time, unless FLTK was built with CMake and option `OPTION_USE_KDIALOG` turned off. In that case, **Fl_Printer** attempts to run the GTK dialog.
  - If the GTK library is not available, or if `Fl::option(Fl::OPTION_PRINTER_USES_GTK)` has been turned off, class **Fl_Printer** runs FLTK's print dialog.

- Unless it has been previously changed, the default paper size is A4. To change that, press the "Properties" button of the "Print" dialog window opened by an **Fl_Printer::begin_job()** call. This opens a "Printer Properties" window where it's possible to select the adequate paper size. Finally press the "Save" button therein to assign the chosen paper size to the chosen printer for this and all further print operations.

- Use the static public attributes of this class to set the print dialog to other languages than English.

  For example, the "Print" dialog item **Fl_Printer::dialog_printer** can be set to French with:

  ```cpp
  Fl_Printer::dialog_printer = "Imprimante:";
  ```

  before creation of the **Fl_Printer** object.

- Use **Fl_PostScript_File_Device::file_chooser_title** to customize the title of the file chooser dialog that opens when using the "Print To File" option of the print dialog.
• Windows platform: Transparent Fl_RGB_Image 's don't print with exact transparency on most printers (a workaround is to use print_window_part()). Fl_RGB_Image 's don't rotate() well.

• Mac OS X platform: all graphics requests print as on display and accept rotation and scaling.

### 33.102.2 Member Function Documentation

#### 33.102.2.1 begin_job()

```cpp
int Fl_Printer::begin_job (  
    int pagecount = 0,  
    int * frompage = NULL,  
    int * topage = NULL,  
    char ** perr_message = NULL  
) [virtual]
```

Begins a print job.

**Parameters**

<table>
<thead>
<tr>
<th>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[I]’d after use.</td>
</tr>
</tbody>
</table>

**Returns**

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented from Fl_Paged_Device.

#### 33.102.2.2 begin_page()

```cpp
int Fl_Printer::begin_page (  
    void  
) [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.

This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

**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.

#### 33.102.2.3 end_job()

```cpp
void Fl_Printer::end_job (  
    void  
) [virtual]
```
To be called at the end of a print job.
Reimplemented from Fl_Paged_Device.

### 33.102.2.4 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.

### 33.102.2.5 is_current()

```cpp
bool Fl_Printer::is_current () [virtual]
```

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

### 33.102.2.6 margins()

```cpp
void Fl_Printer::margins ( int *left,
                            int *top,
                            int *right,
                            int *bottom ) [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out left</td>
<td>If non-null, +left is set to the left margin size.</td>
</tr>
<tr>
<td>out top</td>
<td>If non-null, +top is set to the top margin size.</td>
</tr>
<tr>
<td>out right</td>
<td>If non-null, +right is set to the right margin size.</td>
</tr>
<tr>
<td>out bottom</td>
<td>If non-null, +bottom is set to the bottom margin size.</td>
</tr>
</tbody>
</table>

Reimplemented from Fl_Paged_Device.

### 33.102.2.7 origin() [1/2]

```cpp
void Fl_Printer::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.

### 33.102.2.8 origin() [2/2]

```cpp
void Fl_Printer::origin (
    int x,
    int y ) [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface. Arguments should be expressed relatively to the result of a previous `printable_rect()` call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive `origin()` calls don't combine their effects. Origin() calls are not affected by `rotate()` calls (for classes derived from Fl_Paged_Device).

Parameters

<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.

### 33.102.2.9 printable_rect()

```cpp
int Fl_Printer::printable_rect (  
    int * w,  
    int * h ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

### 33.102.2.10 rotate()

```cpp
void Fl_Printer::rotate (  
    float angle ) [virtual]
```

Rotates the graphics operations relatively to paper. The rotation is centered on the current graphics origin. Successive `rotate()` calls don't combine their effects.

Parameters

| angle |

Rotation angle in counter-clockwise degrees.

Reimplemented from Fl_Paged_Device.

### 33.102.2.11 scale()

```cpp
void Fl_Printer::scale (  
```

Reimplemented from Fl_Paged_Device.
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 \texttt{scale()} call, do a \texttt{printable_rect()} call to get the new dimensions of the printable page area. Successive \texttt{scale()} calls don't combine their effects.

Parameters

| \texttt{scale—} | Horizontal dimensions of plot are multiplied by this quantity. |
| \texttt{scale—} | Same as above, vertically. The value 0. is equivalent to setting \texttt{scale_y = scale_x}. Thus, \texttt{scale(factor)}; is equivalent to \texttt{scale(factor, factor)}; |

Reimplemented from \texttt{Fl_Paged_Device}.

33.102.2.12 \texttt{set_current()}

\texttt{void Fl_Printer::set_current (}
\texttt{ 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 \texttt{Fl_Surface_Device::push_current()} / \texttt{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 \texttt{Fl_Printer} or an \texttt{Fl_Image_Surface} object, calling \texttt{set_current()} on this object;
- draw to that surface;
- make the display current again with \texttt{Fl_Display_Device::display_device()->set_current();} . Don't do any other call to \texttt{set_current()} before this one.

Other scenarios of drawing surface changes should be performed via \texttt{Fl_Surface_Device::push_current()} / \texttt{Fl_Surface_Device::pop_current()}.  

Reimplemented from \texttt{Fl_Surface_Device}.

33.102.2.13 \texttt{translate()}

\texttt{void Fl_Printer::translate (}
\texttt{ int x,}
\texttt{ int y ) [virtual]}

Translates the current graphics origin accounting for the current rotation. Each \texttt{translate()} call must be matched by an \texttt{untranslate()} call. Successive \texttt{translate()} calls add up their effects.  

Reimplemented from \texttt{Fl_Widget_Surface}.

33.102.2.14 \texttt{untranslate()}

\texttt{void Fl_Printer::untranslate (}
\texttt{ void ) [virtual]}

Undoes the effect of a previous \texttt{translate()} call.  

Reimplemented from \texttt{Fl_Widget_Surface}.

The documentation for this class was generated from the following files:

- \texttt{Fl_Printer.H}
- \texttt{Fl_Printer.cxx}
## Fl_Progress Class Reference

Displays a progress bar for the user.

```cpp
#include <Fl_Progress.H>
```

Inheritance diagram for Fl_Progress:

![Inheritance Diagram](image)

### 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

- **void draw () FL_OVERRIDE**

  Draws the progress bar.

### Additional Inherited Members

#### 33.103.1 Detailed Description

Displays a progress bar for the user.

#### 33.103.2 Constructor & Destructor Documentation

##### 33.103.2.1 Fl_Progress()

```cpp
Fl_Progress::Fl_Progress (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char *l = 0 )
```

The constructor creates the progress bar using the position, size, and label.

You can set the background color with `color()` and the progress bar color with `selection_color()`, or you can set both colors together with `color(unsigned bg, unsigned sel)`. The default colors are `FL_BACKGROUND2_COLOR` and `FL_YELLOW`, resp.
33.103.3 Member Function Documentation

33.103.3.1 draw()  
void Fl_Progress::draw (  
  void ) [protected], [virtual]  
Draws the progress bar.  
Implements Fl_Widget.

33.103.3.2 maximum() [1/2]  
float Fl_Progress::maximum ( ) const [inline]  
Gets the maximum value in the progress widget.

33.103.3.3 maximum() [2/2]  
void Fl_Progress::maximum (  
  float v ) [inline]  
Sets the maximum value in the progress widget.

33.103.3.4 minimum() [1/2]  
float Fl_Progress::minimum ( ) const [inline]  
Gets the minimum value in the progress widget.

33.103.3.5 minimum() [2/2]  
void Fl_Progress::minimum (  
  float v ) [inline]  
Sets the minimum value in the progress widget.

33.103.3.6 value() [1/2]  
float Fl_Progress::value ( ) const [inline]  
Gets the current value in the progress widget.

33.103.3.7 value() [2/2]  
void Fl_Progress::value (  
  float v ) [inline]  
Sets the current value in the progress widget.

The documentation for this class was generated from the following files:

- FL_Progress.H
- FL_Progress.cxx
33.104 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Button</td>
</tr>
<tr>
<td>Fl_Radio_Button</td>
</tr>
</tbody>
</table>
```

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

33.104.1 Constructor & Destructor Documentation

33.104.1.1 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

33.105 Fl_Radio_Light_Button Class Reference

Inheritance diagram for Fl_Radio_Light_Button:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Button</td>
</tr>
<tr>
<td>Fl_Radio_Light_Button</td>
</tr>
</tbody>
</table>
```
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

### 33.106 Fl_Radio_Round_Button Class Reference

Inheritance diagram for Fl_Radio_Round_Button:

```
Fl_Widget
Fl_Button
Fl_Light_Button
Fl_Radio_Light_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

#### 33.106.1 Constructor & Destructor Documentation

```
33.106.1.1 Fl_Radio_Round_Button()

Fl_Radio_Round_Button::Fl_Radio_Round_Button (  
    int X,  
    int Y,  
    int W, 
```
int $H$,  
const char * $L = 0$)

Creates a new Fl_Radio_Button widget using the given position, size, and label string. The button type() is set to FL_RADIO_BUTTON.

Parameters

| in | $X,Y,W,H$ | position and size of the widget |
| in | $L$ | widget label, default is no label |

The documentation for this class was generated from the following files:

- Fl_Radio_Round_Button.H
- Fl_Round_Button.cxx

### 33.107 Fl_Rect Class Reference

Rectangle with standard FLTK coordinates ($X$, $Y$, $W$, $H$).

```cpp
#include <Fl_Rect.H>
```

#### Public Member Functions

- **int b () const**
  
  gets the bottom edge ($y + h$).

- **Fl_Rect ()**
  
  The default constructor creates an empty rectangle ($x = y = w = h = 0$).

- **Fl_Rect (const Fl_Widget &widget)**
  
  This constructor creates a rectangle based on a widget's position and size.

- **Fl_Rect (const Fl_Widget *const widget)**
  
  This constructor creates a rectangle based on a widget's position and size.

- **Fl_Rect (int W, int H)**
  
  This constructor creates a rectangle with $x = y = 0$ and the given width and height.

- **Fl_Rect (int X, int Y, int W, int H)**
  
  This constructor creates a rectangle with the given $x,y$ coordinates and the given width and height.

- **Fl_Rect (int X, int Y, int W, int H, Fl_Boxtype bt)**
  
  This constructor creates a rectangle with the given $x,y$ coordinates and the given width and height reduced by the box frame size.

- **int h () const**
  
  gets the height

- **void h (int H)**
  
  sets the height

- **void inset (Fl_Boxtype bt)**
  
  Move all edges in by the frame size of box type $bt$.

- **void inset (int d)**
  
  Move all edges in by $d$.

- **void inset (int left, int top, int right, int bottom)**
  
  Move all edges in by left, top, right, bottom.

- **int r () const**
  
  gets the right edge ($x + w$).

- **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)

### 33.107.1 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.

### 33.107.2 Constructor & Destructor Documentation

#### 33.107.2.1 Fl_Rect()

```cpp
Fl_Rect::Fl_Rect (  
    int X,  
    int Y,  
    int W,  
    int H,  
    Fl_Boxtype bt ) [inline]
```

This constructor creates a rectangle with the given x,y coordinates and the given width and height reduced by the box frame size. This is the same as using the constructor w/o `bt` and subsequently calling `inset(bt)`.

### 33.107.3 Member Function Documentation

#### 33.107.3.1 b()

```cpp
int Fl_Rect::b ( ) const [inline]
gets the bottom edge (y + h).  
```

Note

r() and b() are coordinates outside the area of the rectangle.

#### 33.107.3.2 inset() [1/3]

```cpp
void Fl_Rect::inset (  
    Fl_Boxtype bt ) [inline]
```

Move all edges in by the frame size of box type `bt`. Shrinks the rectangle at all sides by the frame width or height of the given box type `bt`. This method uses the frame sizes given by the box type `bt` using

- Fl::box_dx(bt)
- Fl::box_dy(bt)
- Fl::box_dw(bt)
- Fl::box_dh(bt)

If the rectangle is smaller than the frame sizes the result is undefined, i.e. an invalid or empty rectangle.
33.107.3.3 inset() [2/3]

void Fl_Rect::inset ( 
    int d ) [inline]

Move all edges in by d.
Shrinks the rectangle by d at all sides keeping the center of the rectangle at the same spot.
If d is negative, the rectangle is enlarged.
If d >= w() or h() the result is undefined, i.e. an invalid or empty rectangle.

33.107.3.4 inset() [3/3]

void Fl_Rect::inset ( 
    int left, 
    int top, 
    int right, 
    int bottom ) [inline]

Move all edges in by left, top, right, bottom.
Shrinks the rectangle on all sides keeping the center of the rectangle at the same spot.
If any value is negative, the rectangle is enlarged.
Values are not range checked; it is possible to create an invalid or empty rectangle.

33.107.3.5 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

33.108 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

33.108.1 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
33.109  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

33.109.1 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

33.110  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) FL_OVERRIDE
  Handles the specified event.

Additional Inherited Members

33.110.1 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.

33.110.2 Constructor & Destructor Documentation
33.110.2.1 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.

33.110.3 Member Function Documentation

33.110.3.1 handle()

int Fl_Repeat_Button::handle (
    int event ) [virtual]

Handles the specified event.
You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.

Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |

See also

Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- FL_Repeat_Button.H
- FL_Repeat_Button.cxx

33.111 Fl_Return_Button Class Reference

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.
#include <Fl_Return_Button.H>

Inheritance diagram for Fl_Return_Button:

```
Fl_Widget
   |
   Fl_Button
   |
Fl_Return_Button
```
Public Member Functions

- **Fl_Return_Button** (int X, int Y, int W, int H, const char *l=0)
  
  Creates a new Fl_Return_Button widget using the given position, size, and label string.

- **int handle (int)** FL_OVERRIDE
  
  Handles the specified event.

Protected Member Functions

- **void draw ()** FL_OVERRIDE
  
  Draws the widget.

Additional Inherited Members

33.111.1 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.

![Image of Fl_Return_Button]

Figure 33.32 Fl_Return_Button

33.111.2 Constructor & Destructor Documentation

33.111.2.1 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.

33.111.3 Member Function Documentation

33.111.3.1 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:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Button.
33.112.3.2 handle()

```cpp
int Fl_Return_Button::handle (  
    int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

| 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

### 33.112 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.

```cpp
#include <Fl_Image.H>
```

Inheritance diagram for Fl_RGB_Image:

```
Fl_RGB_Image  
  |   
  v   
Fl_Image  
  |   
Fl_RGB_Image  
  |   
Fl_BMP_Image  Fl_JPEG_Image  Fl_PNG_Image  Fl_PNM_Image  Fl_SVG_Image  
  |   
Fl_ICO_Image
```

**Public Member Functions**

- virtual Fl_SVG_Image * as_svg_image ()
  
  Returns whether an image is an Fl_SVG_Image or not.

- void color_average (Fl_Color c, float i) FL_OVERRIDE
  
  The color_average() method averages the colors in the image with the provided FLTK color value.

- Fl_Image * copy () const

- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  
  Creates a resized copy of the image.
• void desaturate () FL_OVERRIDE

   The desaturate() method converts an image to grayscale.

• void draw (int X, int Y)

• void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE

   Draws the image to the current drawing surface with a bounding box.

• Fl_RGB_Image (const Fl_Pixmap ∗pxm, Fl_Color bg=FL_GRAY)

   The constructor creates a new RGBA image from the specified Fl_Pixmap.

• Fl_RGB_Image (const uchar ∗bits, int bits_length, int W, int H, int D, int LD)

   The constructor creates a new image from the specified data.

• Fl_RGB_Image (const uchar ∗bits, int W, int H, int D=3, int LD=0)

   The constructor creates a new image from the specified data.

• void label (Fl_Menu_Item ∗m) FL_OVERRIDE

   This method is an obsolete way to set the image attribute of a menu item.

• void label (Fl_Widget ∗w) FL_OVERRIDE

   This method is an obsolete way to set the image attribute of a widget or menu item.

• virtual void normalize ()

   Makes sure the object is fully initialized.

• void uncache () FL_OVERRIDE

   If the image has been cached for display, delete the cache data.

• ~Fl_RGB_Image () FL_OVERRIDE

   The destructor frees all memory and server resources that are used by the image.

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.

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

33.112.1 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.

33.112.2 Constructor & Destructor Documentation
33.112.2.1  Fl_RGB_Image() [1/3]

The constructor creates a new image from the specified data.

\[ \text{The data array } \text{bits must contain sufficient data to provide } W \times H \times D \text{ image bytes and optional line padding, see LD.} \]

\[ \text{W and H are the width and height of the image in pixels, resp.} \]

\[ D \text{ is the image depth and can be:} \]

- \( D=1 \): each uchar in \text{bits[]} is a grayscale pixel value
- \( D=2 \): each uchar pair in \text{bits[]} is a grayscale + alpha pixel value
- \( D=3 \): each uchar triplet in \text{bits[]} is an R/G/B pixel value
- \( D=4 \): each uchar quad in \text{bits[]} is an R/G/B/A pixel value

\[ \text{LD specifies the line data size of the array, see Fl_Image::ld(int). If LD is zero, then } W \times D \text{ is assumed, otherwise LD must be greater than or equal to } W \times D \text{ to account for (unused) extra data per line (padding).} \]

The caller is responsible that the image data array \text{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 \text{bits}, set alloc_array to non-zero after construction.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bits</th>
<th>The image data array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W</td>
<td>The width of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>H</td>
<td>The height of the image in pixels.</td>
</tr>
<tr>
<td>in</td>
<td>D</td>
<td>The image depth, or 'number of channels' (default=3).</td>
</tr>
<tr>
<td>in</td>
<td>LD</td>
<td>Line data size (default=0).</td>
</tr>
</tbody>
</table>

See also

Fl_Image::data(), Fl_Image::w(), Fl_Image::h(), Fl_Image::d(), Fl_Image::ld(int)

33.112.2.2  Fl_RGB_Image() [2/3]

The constructor creates a new image from the specified data.

If the provided array is too small to contain all the image data, the constructor will not generate the image to avoid illegal memory read access and instead set \text{data} to NULL and \text{ld} to ERR_MEMORY_ACCESS.

Parameters

<table>
<thead>
<tr>
<th>bits</th>
<th>image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_length</td>
<td>length of the bits array in bytes</td>
</tr>
<tr>
<td>W</td>
<td>image width in pixels</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H$</td>
<td>image height in pixels</td>
</tr>
<tr>
<td>$D$</td>
<td>image depth in bytes, 1 for grayscale, 2 for grayscale with alpha, 3 for RGB, and 4 for RGB plus alpha</td>
</tr>
<tr>
<td>$LD$</td>
<td>line length in bytes, or 0 to use $W \times D$.</td>
</tr>
</tbody>
</table>

See also

```cpp
FL_RGB_Image(const uchar *bits, int $W$, int $H$, int $D$, int $LD$)
```

### 33.112.2.3 $\texttt{FL\_RGB\_Image()}$ [3/3]

```cpp
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 $\texttt{FL\_RGB\_Image::alloc\_array}$ to 1 so the data array is deleted when the image is destroyed.

### 33.112.3 Member Function Documentation

#### 33.112.3.1 as_svg_image()

```cpp
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.

#### 33.112.3.2 color_average()

```cpp
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.

#### 33.112.3.3 copy()

```cpp
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`, `Fl_Anim_GIF_Image` and `Fl_Shared_Image`. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an `Fl_GIF_Image` is an object of class `Fl_Pixmap`.

**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`.

### 33.112.3.4 desaturate()

```cpp
void Fl_RGB_Image::desaturate() override { }
```

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`.

### 33.112.3.5 draw()

```cpp
void Fl_RGB_Image::draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) override { }
```

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`. 

---

Generated by Doxygen
33.112.3.6 label() [1/2]

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`.

33.112.3.6 label() [2/2]

void Fl_RGB_Image::label (  
    Fl_Widget * widget ) [virtual]

This method is an obsolete way to set the image attribute of a widget or menu item.

**Deprecated** Please use `Fl_Widget::image()` or `Fl_Widget::deimage()` instead.

Reimplemented from `Fl_Image`.

33.112.3.8 max_size() [1/2]

static size_t Fl_RGB_Image::max_size ( ) [inline], [static]

Returns the maximum allowed image size in bytes when creating an `Fl_RGB_Image` object.

See also

```cpp
void Fl_RGB_Image::max_size(size_t)
```

33.112.3.9 max_size() [2/2]

static void Fl_RGB_Image::max_size (  
    size_t size ) [inline], [static]

Sets the maximum allowed image size in bytes when creating an `Fl_RGB_Image` object. The image size in bytes of an `Fl_RGB_Image` object is the value of the product \( w() \times h() \times d() \). 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.

33.112.3.10 normalize()

virtual void Fl_RGB_Image::normalize ( ) [inline], [virtual]

Makes sure the object is fully initialized.

In particular, makes sure member variable `array` is non-null.

Reimplemented in `Fl_SVG_Image`.

33.112.3.11 uncache()

void Fl_RGB_Image::uncache ( ) [virtual]

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented from `Fl_Image`.

33.112.4 Member Data Documentation
33.112.4.1 array

const uchar* Fl_RGB_Image::array
Points to the start of the object's data array.

See also

class Fl_SVG_Image which delays initialization of this member variable.

The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

33.113 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) FL_OVERRIDE
  Handles the specified event.

Protected Member Functions

- void draw () FL_OVERRIDE
  Draws the widget.

Additional Inherited Members

33.113.1 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.
33.113.2 Constructor & Destructor Documentation

33.113.2.1 Fl_Roller()

Fl_Roller::Fl_Roller (  
  int X, 
  int Y, 
  int W, 
  int H, 
  const char * L = 0 )

Creates a new Fl_Roller widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
Inherited destructor destroys the valuator.

33.113.3 Member Function Documentation

33.113.3.1 draw()

void Fl_Roller::draw ( ) [protected], [virtual]

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as
soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded
scrollbar, you can do it (because draw() is virtual) like this:
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();             // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

33.113.3.2 handle()

int Fl_Roller::handle (  
  int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:
33.114  Fl_Round_Button Class Reference

Buttons generate callbacks when they are clicked by the user.
#include <Fl_Round_Button.H>

Inheritance diagram for Fl_Round_Button:

```
Fl_Widget
  |
  v
Fl_Button
  |
  v
Fl_Light_Button
  |
  v
Fl_Round_Button
  |
  v
Fl_Radio_Round_Button
```

Public Member Functions

- **Fl_Round_Button (int x, int y, int w, int h, const char ∗l=0)**
  Creates a new Fl_Round_Button widget using the given position, size, and label string.

Additional Inherited Members

33.114.1  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 33.34 Fl_Round_Button
```

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.

33.114.2  Constructor & Destructor Documentation

33.114.2.1  Fl_Round_Button()

Fl_Round_Button::Fl_Round_Button (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char ∗L = 0 )

Creates a new Fl_Round_Button widget using the given position, size, and label string.
The **Fl_Round_Button** subclass displays the "ON" state by turning on a light, rather than drawing pushed in. The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark. The shape of the "light" is set with `down_box()` and its default value is FL_ROUND_DOWN_BOX. The color of the light when on is controlled with `selection_color()`, which defaults to FL_FOREGROUND_COLOR (usually black).

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
</tr>
</tbody>
</table>

The documentation for this class was generated from the following files:

- Fl_Round_Button.H
- Fl_Round_Button.cxx

### 33.115 Fl_Round_Clock Class Reference

A clock widget of type FL_ROUND_CLOCK.

```cpp
#include <Fl_Round_Clock.H>
```

Inheritance diagram for Fl_Round_Clock:

```
Fl_Widget
  `-- Fl_Clock
      `-- Fl_Clock_Output
          `-- 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

### 33.115.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK. Has no box.

### 33.115.2 Constructor & Destructor Documentation
33.115.2.1  Fl_Round_Clock()

Fl_Round_Clock::Fl_Round_Clock (
    int X,
    int Y,
    int W,
    int H,
    const char ∗ L = 0 )
Creates the clock widget, setting his type and box.
Create an Fl_Round_Clock widget using the given position, size, and label string.
The clock type is FL_ROUND_CLOCK and the boxtype is FL_NO_BOX.
This constructor is the same as Fl_Clock(FL_ROUND_CLOCK, X, Y, W, H, L).
See also
    Fl_Clock(uchar, int, int, int, int, const char ∗ )

Parameters

<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

33.116  Fl_Scheme Class Reference

Static Public Member Functions

• static int add_scheme_name (const char ∗name)  
  Add a scheme name to the list of known schemes.
• static const char ∗ ∗ names ()  
  Return a list of all known scheme names.
• static int num_schemes ()  
  Return the number of currently registered schemes.

33.116.1  Member Function Documentation

33.116.1.1  add_scheme_name()

int Fl_Scheme::add_scheme_name (const char ∗ name ) [static]
Add a scheme name to the list of known schemes.
This method is public in FLTK 1.4.0 because derived classes of Fl_Scheme are not yet implemented. Thus, users implementing their own schemes can use this method to add the scheme name to the list of known schemes which is for instance used in Fl_Scheme::names().

Note

Attention! In a future version, when subclasses of Fl_Scheme will be implemented, this method will either be replaced by another protected method or it will no longer do anything (kept only for ABI reasons).

The new scheme name must consist of valid ASCII characters as described below:

• lowercase letters 'a’ - 'z'
• numbers '0' - '9'
• any character in "$+_." (w/o the quotes).

The name must not be longer than 12 ASCII characters (bytes). The new scheme name is added to the end of the unordered list.

Note
  Call this method only once for each scheme name. If the returned value is $\leq 0$ you should check the scheme name.

The given scheme name is copied and may be freed directly after the call to `add_scheme_name()`.

Parameters

- **in name** New scheme name

Returns

The new number of schemes if the name was successfully added. This is the same as the index of the scheme + 1.

Return values

- **0** Scheme name already exists
- **-1** Invalid character(s) in name
- **-2** The name is too long

Since

1.4.0

### 33.116.1.2 names()

```cpp
const char ** Fl_Scheme::names ( ) [static]
```

Return a list of all known scheme names.

This list is only valid until a new scheme is added or one is removed. It is possible that scheme names are appended to the list during the runtime of the program but schemes can't be removed.

Getting the list of known schemes can be useful to populate a menu of scheme choices to let the user select a scheme. You should process the names immediately and you should never store a pointer to the list or any individual name for later reference because the location of the list can be changed (reallocated) when schemes are added.

The list of scheme names is nul-terminated.

Note

Currently (in FLTK 1.4.0) schemes can only be added to the list and not removed from the list. This may change in a later version.

Returns

List of currently known scheme names.

### 33.116.1.3 num_schemes()

```cpp
static int Fl_Scheme::num_schemes ( ) [inline], [static]
```

Return the number of currently registered schemes.
Fl_Scheme_Choice Class Reference

Returns

Number of registered schemes.

The documentation for this class was generated from the following files:

- Fl_Scheme.H
- Fl_Scheme.cxx

33.117 Fl_Scheme_Choice Class Reference

Inheritance diagram for Fl_Scheme_Choice:

```
Fl_Widget
  ↓
Fl_Menu_
  ↓
Fl_Choice
  ↓
Fl_Scheme_Choice
```

Public Member Functions

- **Fl_Scheme_Choice (int X, int Y, int W, int H, const char *L=0)**
  The constructor initializes the Fl_Scheme_Choice object with all known schemes.

- **int handle (int event)** FL_OVERRIDE
  Handle FLTK events.

- **virtual void init_value ()**
  Public method to initialize the value of the Fl_Scheme_Choice widget.

Static Protected Member Functions

- **static void scheme_cb_ (Fl_Widget *w, void *)**
  Internal Fl_Scheme_Choice callback function (protected).

Additional Inherited Members

33.117.1 Constructor & Destructor Documentation

33.117.1.1 Fl_Scheme_Choice()

Fl_Scheme_Choice::Fl_Scheme_Choice {
  int X,
  int Y,
  int W,
  int H,
  const char * L = 0
}

The constructor initializes the Fl_Scheme_Choice object with all known schemes.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X,Y</th>
<th>Widget coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>W,H</td>
<td>Widget size (width, height)</td>
</tr>
<tr>
<td>in</td>
<td>L</td>
<td>Widget label (default: NULL, no label)</td>
</tr>
</tbody>
</table>
33.117.2 Member Function Documentation

33.117.2.1 handle()

```cpp
int Fl_Scheme_Choice::handle (int event) [virtual]
```

Handle FLTK events.
This widget uses FL_SHOW and some other events to initialize its `value()` according to the current scheme.
All events are also handled by the base class `Fl_Choice`.

Parameters

- `event`

Returns

- 1 if the event was used, 0 otherwise

Reimplemented from `Fl_Choice`.

33.117.2.2 init_value()

```cpp
void Fl_Scheme_Choice::init_value () [virtual]
```

Public method to initialize the value of the `Fl_Scheme_Choice` widget.
Normally you don't need to call this unless you change the current scheme by calling `Fl::scheme(const char *)`.
The `Fl_Scheme_Choice` widget does this automatically when the widget is shown (when receiving the FL_SHOW event) which should always be after `Fl_Window::show(argc, argv)` which may set the current scheme by interpreting the commandline.

Since

1.4.0

33.117.2.3 scheme_cb_()

```cpp
void Fl_Scheme_Choice::scheme_cb_ (Fl_Widget * w, void * ) [static], [protected]
```

Internal `Fl_Scheme_Choice` callback function (protected).
You don't need to set a callback for this widget. The default callback changes the scheme (`Fl::scheme()`) and redraws all open windows.
You may override the callback if changing the scheme shall redraw other windows or don't redraw the window at all.

Parameters

- `w` The `Fl_Scheme_Choice` widget

The documentation for this class was generated from the following files:

- Fl_Scheme_Choice.H
- Fl_Scheme_Choice.cxx

33.118 Fl_Scroll Class Reference

This container widget lets you maneuver around a set of widgets much larger than your window.
#include <Fl_Scroll.H>

Inheritance diagram for Fl_Scroll:

```
Fl_Widget
  ↓
Fl_Group
  ↓
Fl_Scroll
```

## Classes
- struct Fl_Region_LRTB
  A local struct to manage a region defined by left/right/top/bottom.
- struct Fl_Region_XYWH
  A local struct to manage a region defined by xywh.
- struct Fl_Scrollbar_Data
  A local struct to manage a scrollbar's xywh region and tab values.
- struct ScrollInfo
  Structure to manage scrollbar and widget interior sizes.

## Public Types
- enum { HORIZONTAL = 1, VERTICAL = 2, BOTH = 3, ALWAYS_ON = 4, HORIZONTAL_ALWAYS = 5, VERTICAL_ALWAYS = 6, BOTH_ALWAYS = 7 }

## Public Member Functions
- void clear ()
  Clear all but the scrollbars...
- int delete_child (int n) FL_OVERRIDE
  Removes the widget at index from the group and deletes it.
- Fl_Scroll (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Scroll widget using the given position, size, and label string.
- int handle (int) FL_OVERRIDE
  Handles the specified event.
- void resize (int X, int Y, int W, int H) FL_OVERRIDE
  Resizes the Fl_Scroll widget and moves its children if necessary.
- void scroll_to (int, int)
  Moves the contents of the scroll group to a new position.
- int scrollbar_size () const
  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
- int xposition () const
  Gets the current horizontal scrolling position.
- int yposition () const
  Gets the current vertical scrolling position.
- virtual ~Fl_Scroll ()
  The destructor also deletes all the children.
Public 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 () FL_OVERRIDE`
  
  *Draws the widget.*

- void `fix_scrollbar_order ()`
  
  *Ensure the scrollbars are the last children.*

- int `on_insert (Fl_Widget *, int) FL_OVERRIDE`
  
  *Change insert position of a child before it is added.*

- int `on_move (int, int) FL_OVERRIDE`
  
  *Change new position of a child before it is moved.*

- void `recalc_scrollbars (ScrollInfo &si) const`
  
  *Calculate visibility/size/position of scrollbars, find children's bounding box.*

Additional Inherited Members

33.118.1 Detailed Description

This container widget lets you maneuver around a set of widgets much larger than your window. If the child widgets are larger than the size of this object then scrollbars will appear so that you can scroll over to them:

![](image)

*Figure 33.36 Fl_Scroll*

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:

```cpp
Fl_Scroll scroll(100, 100, 200, 200); // Fl_Scroll at (100, 100)
Fl_Box(100, 100, 1, 1); // Fl_Box in top left corner
Fl_Input(150, 120, 60, 30, "Input:"); // left most widget with label
// ... more widgets ...
scroll.end();
```

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.
33.118 Fl_Scroll Class Reference

- 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.

33.118.2 Constructor & Destructor Documentation

33.118.2.1 Fl_Scroll()

Fl_Scroll::Fl_Scroll (  
  int X,  
  int Y,  
  int W,  
  int H,  
  const char * L = 0 )

Creates a new Fl_Scroll widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Scroll and all of its children can be automatic (local) variables, but you must declare the Fl_Scroll first, so that it is destroyed last.

33.118.2.2 ~Fl_Scroll()

Fl_Scroll::~Fl_Scroll () [virtual]
The destructor also deletes all the children.

See also

Fl_Group::~Fl_Group()

33.118.3 Member Function Documentation

33.118.3.1 bbox()

void Fl_Scroll::bbox (  
  int & X,  
  int & Y,  
  int & W,  
  int & H ) const [protected]
Returns the bounding box for the interior of the scrolling area, inside the scrollbars. This method does not change the scrollbars or their visibility. First the scrollbar positions and visibility are calculated as they should be, according to the positions and sizes of the children. Then the bounding box is calculated. You may need to call `redraw()` to make sure the widget gets updated.

See also
recalc_scrollbars()

33.118.3.2 delete_child()

```cpp
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>1</td>
<td>index out of range</td>
</tr>
<tr>
<td>2</td>
<td>widget not allowed to be removed (see note)</td>
</tr>
</tbody>
</table>

See also

- `Fl_Group::delete_child(int index)`

Since

FLTK 1.4.0

Reimplemented from `Fl_Group`.

33.118.3.3 draw()

```cpp
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`.

33.118.3.4 fix_scrollbar_order()

```cpp
void Fl_Scroll::fix_scrollbar_order ( ) [protected]
```

Generated by Doxygen

---

806 Class Documentation
Ensure the scrollbars are the last children.
When Fl_Scroll is instantiated the first child of the Fl_Group is the vertical scrollbar scrollbar and the second child is the horizontal scrollbar hscrollbar.
These two widgets must always be the last two widgets and in this order to guarantee the correct drawing order and event delivery.
Since FLTK 1.4.0 the new method on_insert() modifies the insert position of other children if it would be after the scrollbars.

33.118.3.5 handle()

```cpp
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.

33.118.3.6 on_insert()

```cpp
int Fl_Scroll::on_insert (  
    Fl_Widget * candidate,  
    int index  ) [protected], [virtual]
```
Change insert position of a child before it is added.
Fix insert position if the new child is planned to be inserted after the scrollbars. We can assume that the scrollbars are always the last two children!
Fl_Group calls this when a new widget is added. We return the new index if the new widget would be added after the scrollbars.

Parameters

| in | candidate | the candidate will be added to the child array_ after this method returns. |
| in | index | add the child at this position in the array_ |

Returns

index to position the child as planned
a new index to force the child to a different position
-1 to keep the group from adding the candidate
33.118.3.7 on_move()

```c
int Fl_Scroll::on_move ( int old_index,
                         int new_index ) [protected], [virtual]
```

Change new position of a child before it is moved.
Fix new position if the new child is planned to be moved after the scrollbars. We can assume that the scrollbars are always the last two children!
*Fl_Group* calls this when a widget is moved within the list of children. We return a new index if the widget would be moved after the scrollbars.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>old_index</em></td>
<td>the current index of the child that will be moved</td>
</tr>
<tr>
<td><em>new_index</em></td>
<td>the new index of the child</td>
</tr>
</tbody>
</table>

**Returns**

new index, possibly corrected to avoid last two scrollbar entries

Reimplemented from *Fl_Group*.

33.118.3.8 recalc_scrollbars()

```c
void Fl_Scroll::recalc_scrollbars ( ScrollInfo & si ) const [protected]
```

Calculate visibility/size/position of scrollbars, find children's bounding box.
The *si* parameter will be filled with data from the calculations. Derived classes can make use of this call to figure out the scrolling area eg. during *resize()* handling.
This method does not change the scrollbars or their visibility. It calculates the scrollbar positions and visibility as they should be, according to the positions and sizes of the children.
You may need to call *redraw()* to make sure the widget gets updated.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>si</em></td>
<td>ScrollInfo structure, filled with data</td>
</tr>
</tbody>
</table>

See also

*bbox()*
int \( W \),
int \( H \)  [virtual]

Resizes the \texttt{Fl\_Scroll} widget and moves its children if necessary. The \texttt{Fl\_Scroll} widget first resizes itself, and then it moves all its children if (and only if) the \texttt{Fl\_Scroll} widget has been moved. The children are moved by the same amount as the \texttt{Fl\_Scroll} widget has been moved, hence all children keep their relative positions.

\textbf{Note}

\texttt{Fl\_Scroll::resize()} does not call \texttt{Fl\_Group::resize()}, and child widgets are not resized.

Since children of an \texttt{Fl\_Scroll} are not resized, the \texttt{resizable()} widget is ignored (if it is set). The scrollbars are moved to their proper positions, as given by \texttt{Fl\_Scroll::scrollbar.align()}, and switched on or off as necessary.

\textbf{Note}

Due to current (FLTK 1.3.x) implementation constraints some of this may effectively be postponed until the \texttt{Fl\_Scroll} is drawn the next time. This may change in a future release.

See also

\texttt{Fl\_Group::resizable()}

\texttt{Fl\_Widget::resize(int,int,int,int)}

Reimplemented from \texttt{Fl\_Group}.

\subsection{scroll_to()}

\begin{verbatim}
void Fl_Scroll::scroll_to(
    int \( X \),
    int \( Y \)
)
\end{verbatim}

Moves the contents of the scroll group to a new position. This is like moving the scrollbars of the \texttt{Fl\_Scroll} around. For instance:

\begin{verbatim}
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);
\end{verbatim}

will move the logical origin of the internal scroll area to (-50,-100) relative to the origin of the \texttt{Fl\_Scroll} (10,10), i.e. \texttt{Fl\_Box} b3 will be visible in the top left corner of the scroll area.

\subsection{scrollbar_size() [1/2]}

\begin{verbatim}
int Fl_Scroll::scrollbar_size() const [inline]
\end{verbatim}

Gets the current size of the scrollbars' troughs, in pixels. If this value is zero (default), this widget will use the \texttt{Fl::scrollbar_size()} value as the scrollbar's width.

\textbf{Returns}

Scrollbar size in pixels, or 0 if the global \texttt{Fl::scrollbar_size()} is being used.

See also

\texttt{Fl::scrollbar_size(int)}
### scrollbar_size()

```c
void Fl_Scroll::scrollbar_size ( int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels. Normally you should not need this method, and should use `Fl::scrollbar_size(int)` instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want. Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare. Setting `newSize` to the special value of 0 causes the widget to track the global `Fl::scrollbar_size()`, which is the default.

**Parameters**

| in  | `newSize` | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global `Fl::scrollbar_size()` |

See also

- `Fl::scrollbar_size()`

### xposition()

```c
int Fl_Scroll::xposition ( ) const [inline]
```

Gets the current horizontal scrolling position.

### yposition()

```c
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`

### Fl_Scrollbar Class Reference

The `Fl_Scrollbar` widget displays a slider with arrow buttons at the ends of the scrollbar.

```c
#include <Fl_Scrollbar.H>
```

Inheritance diagram for `Fl_Scrollbar`:

![Inheritance Diagram](generated/inheritance.png)
Public Member Functions

- **Fl_Scrollbar** (int X, int Y, int W, int H, const char *L=0)
  
  Creates a new Fl_Scrollbar widget with given position, size, and label.

- **int handle** (int) FL_OVERRIDE
  
  Handles the specified event.

- **int linesize** () 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 integer 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** () FL_OVERRIDE
  
  Draws the widget.

Additional Inherited Members

33.119.1 Detailed Description

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar. Clicking on the arrows move up/left and down/right by `linesize()`. Scrollbars also accept FL_SHORTCUT events: the arrows move by `linesize()`, and vertical scrollbars take Page Up/Down (they move by the page size minus `linesize()`) and Home/End (they jump to the top or bottom). Scrollbars have step(1) preset (they always return integers). If desired you can set the step() to non-integer values. You will then have to use casts to get at the floating-point versions of `value()` from Fl_Slider.

![Figure 33.37 Fl_Scrollbar](image)

33.119.2 Constructor & Destructor Documentation

33.119.2.1 Fl_Scrollbar()

Fl_Scrollbar::Fl_Scrollbar (  
  int X,  
  int Y,  
  int W,
int $H$,
const char * $L = 0$

Creates a new `Fl_Scrollbar` widget with given position, size, and label.
You need to do `type(FL_HORIZONTAL)` if you want a horizontal scrollbar.

### 33.119.3 Member Function Documentation

#### 33.119.3.1 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`.

#### 33.119.3.2 handle()

```cpp
int Fl_Scrollbar::handle ( int event ) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th>Event</th>
<th>the kind of event received</th>
</tr>
</thead>
</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`.

#### 33.119.3.3 linesize()

```cpp
void Fl_Scrollbar::linesize ( int i ) [inline]
```

This number controls how big the steps are that the arrow keys do.
In addition page up/down move by the size last sent to `value()` minus one `linesize()`. The default is 16.

#### 33.119.3.4 value() [1/3]

```cpp
int Fl_Scrollbar::value ( ) const [inline]
```

Generated by Doxygen
Gets the integer value (position) of the slider in the scrollbar. You can get the floating point value with Fl_Slider::value().

See also

 Fl_Scrollbar::value(int p)
 Fl_Scrollbar::value(int pos, int size, int first, int total)

33.119.3.5 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)

33.119.3.6 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>in</th>
<th>pos</th>
<th>position, first line displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>windowSize</td>
<td>number of lines displayed</td>
</tr>
<tr>
<td>in</td>
<td>first_line</td>
<td>number of first line</td>
</tr>
<tr>
<td>in</td>
<td>total_lines</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

You should call this every time your window changes size, your data changes size, or your scroll position changes (even if in response to a callback from this scrollbar). All necessary calls to redraw() are done. Calls Fl_Slider::scrollvalue(int pos, int size, int first, int total).

The documentation for this class was generated from the following files:

- Fl_Scroll.H
- Fl_Scrollbar.cxx

33.120 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

Generated by Doxygen
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

33.120.1 Detailed Description
A local struct to manage a scrollbar's xywh region and tab values.
The documentation for this struct was generated from the following file:
  • Fl_Scroll.H

33.121 Fl_Secret_Input Class Reference
The Fl_Secret_Input class is a subclass of Fl_Input that displays its input as a string of placeholders.
#include <Fl_Secret_Input.H>
Inheritance diagram for Fl_Secret_Input:

```
+----+     +----------+
|    |     |           |
|    | v    | Fl_Secret |
|    +-----+           |
|        |              |
|        +-----------+
|                    |
|                    |
|                    +-- Fl_Input
|                    |
|                    |
+--------------------+
```

Public Member Functions
  • Fl_Secret_Input (int X, int Y, int W, int H, const char *l=0)
    Creates a new Fl_Secret_Input widget using the given position, size, and label string.
  • int handle (int) FL_OVERRIDE
    Handles the specified event.

Additional Inherited Members
33.121.1 Detailed Description
The Fl_Secret_Input class is a subclass of Fl_Input that displays its input as a string of placeholders.
Depending on the platform this placeholder is either the asterisk ('*') or the Unicode bullet character (U+2022).
This subclass is usually used to receive passwords and other "secret" information.

33.121.2 Constructor & Destructor Documentation
33.121.2.1 Fl_Secret_Input()

Fl_Secret_Input::Fl_Secret_Input ( int X, int Y, int W, int H, const char ∗ l = 0 )

Creates a new Fl_Secret_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.

33.121.3 Member Function Documentation

33.121.3.1 handle()

int Fl_Secret_Input::handle ( int event ) [virtual]

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

- **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

33.122 Fl_Select_Browser Class Reference

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.
#include <Fl_Select_Browser.H>
Inheritance diagram for Fl_Select_Browser:
Public Member Functions

- Fl_Select_Browser (int X, int Y, int W, int H, const char *L=0)
  Creates a new Fl_Select_Browser widget using the given position, size, and label string.

Additional Inherited Members

33.122.1 Detailed Description

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.
As long as the mouse button is held down on an unselected item it is highlighted. Normally the callback is done when the user presses the mouse, but you can change this with when().
See Fl_Browser for methods to add and remove lines from the browser.

33.122.2 Constructor & Destructor Documentation

33.122.2.1 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

33.123 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:
Public Member Functions

- `Fl_Shared_Image * as_shared_image () FL_OVERRIDE`
   Returns whether an image is an `Fl_Shared_Image` or not.
- `void color_average (Fl_Color c, float i) FL_OVERRIDE`
   The `color_average()` method averages the colors in the image with the provided FLTK color value.
- `Fl_Image * copy () const`
- `Fl_Image * copy (int W, int H) const FL_OVERRIDE`
  Creates a resized copy of the image.
- `void desaturate () FL_OVERRIDE`
  The `desaturate()` method converts an image to grayscale.
- `void draw (int X, int Y)`
- `void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE`
  Draws the image to the current drawing surface with a bounding box.
- `const Fl_Image * image () const`
  Returns a pointer to the internal `Fl_Image` object.
- `const char * name ()`
  Returns the filename of the shared image.
- `int original ()`
  Returns whether this is an original image.
- `int refcount ()`
  Returns the number of references of this shared image.
- `void release () FL_OVERRIDE`
  Releases and possibly destroys (if refcount ≤ 0) a shared image.
- `virtual void reload ()`
  Reloads the shared image from disk.
- `void uncache () FL_OVERRIDE`
  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
- class FL_SVG_Image

Additional Inherited Members

33.123.1 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()
33.123.2 Constructor & Destructor Documentation

33.123.2.1 Fl_Shared_Image() [1/2]

Fl_Shared_Image::Fl_Shared_Image ( ) [protected]
Creates an empty shared image.
The constructors create a new shared image record in the image cache.
The constructors are protected and cannot be used directly from a program. Use the get() method instead.

33.123.2.2 Fl_Shared_Image() [2/2]

Fl_Shared_Image::Fl_Shared_Image ( const char ∗ n, Fl_Image ∗ img = 0 ) [protected]
Creates a shared image from its filename and its corresponding Fl_Image img.
The constructors create a new shared image record in the image cache.
The constructors are protected and cannot be used directly from a program. Use the get() method instead.

33.123.2.3 ~Fl_Shared_Image()

Fl_Shared_Image::~Fl_Shared_Image ( ) [protected], [virtual]
The destructor frees all memory and server resources that are used by the image.
The destructor is protected and cannot be used directly from a program. Use the Fl_Shared_Image::release() method instead.

33.123.3 Member Function Documentation

33.123.3.1 add()

void Fl_Shared_Image::add ( ) [protected]
Adds a shared image to the image cache.
This protected method adds an image to the cache, an ordered list of shared images. The cache is searched for a matching image whenever one is requested, for instance with Fl_Shared_Image::get() or Fl_Shared_Image::find().

33.123.3.2 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.

33.123.3.3 as_shared_image()

Fl_Shared_Image ∗ Fl_Shared_Image::as_shared_image ( ) [inline], [virtual]
Returns whether an image is an Fl_Shared_Image or not.
This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_Image.
Since 1.4.0

Reimplemented from Fl_Image.

### 33.123.3.4 color_average()

```cpp
void Fl_Shared_Image::color_average (
    Fl_Color c,
    float i ) [virtual]
```

The `color_average()` method averages the colors in the image with the 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.

### 33.123.3.5 compare()

```cpp
int Fl_Shared_Image::compare (
    Fl_Shared_Image **i0,
    Fl_Shared_Image **i1 ) [static], [protected]
```

Compares two shared images. The order of comparison is:

1. Image name, usually the filename used to load it
2. Image width
3. Image height

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).

<table>
<thead>
<tr>
<th>Return values</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>
33.123.3.6  copy()

```
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_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in W,H</td>
<td>Requested width and height of the new image</td>
</tr>
</tbody>
</table>

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.

33.123.3.7  desaturate()

```
void Fl_Shared_Image::desaturate ( ) [virtual]
```

The desaturate() method converts an image to grayscale.

If the image contains an alpha channel (depth = 4), the alpha channel is preserved.

An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.

Reimplemented from Fl_Image.

33.123.3.8  draw()

```
void Fl_Shared_Image::draw (  
   int X,  
   int Y,  
   int W,  
   int H,  
   int cx = 0,  
   int cy = 0 ) [virtual]
```

Draws the image to the current drawing surface with a bounding box.

Arguments X, Y, W, H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.

In other words: 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`.

### 33.123.3.9 `find()`

```cpp
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.

### 33.123.3.10 `get()` [1/2]

```cpp
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>W,H</th>
<th></th>
<th>name of the image</th>
<th>desired size</th>
</tr>
</thead>
</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)`

### 33.123.3.11 `get()` [2/2]

```cpp
Fl_Shared_Image * Fl_Shared_Image::get (  
    Fl_RGB_Image * rgb,  
    int own_it = 1 ) [static]
```

Builds a shared image from a pre-existing `Fl_RGB_Image`. 

---

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rgb</th>
<th>an Fl_RGB_Image used to build a new shared image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>own←_it</td>
<td>1 if the shared image should delete rgb when it is itself deleted, 0 otherwise</td>
</tr>
</tbody>
</table>

Version

1.3.4

33.123.3.12 image()

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

33.123.3.13 original()

int Fl_Shared_Image::original ( ) [inline]
Returns whether this is an original image.
Images loaded from a file or from memory are marked original as opposed to images created as a copy of another image with different size (width or height).

**Note**
This is useful for debugging (rarely used in user code).

Since

FLTK 1.4.0

33.123.3.14 refcount()

int Fl_Shared_Image::refcount ( ) [inline]
Returns the number of references of this shared image.
When reference is below 1, the image is deleted.
33.123.3.15 release()
void Fl_Shared_Image::release () [virtual]
Releases and possibly destroys (if refcount <= 0) a shared image.
In the latter case, it will reorganize the shared image array so that no hole will occur.
Reimplemented from Fl_Image.

33.123.3.16 uncache()
void Fl_Shared_Image::uncache () [virtual]
If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:

• Fl_Shared_IMAGE.H
• Fl_Shared_IMAGE.cxx

33.124 Fl_Shortcut_Button Class Reference
A button that allows the user to type a key combination to create shortcuts.
Inheritance diagram for Fl_Shortcut_Button:

Public Member Functions

• void default_clear ()
  No longer show the button to reverse to a default shortcut.
• Fl_Shortcut default_value ()
  Return the default shortcut.
• void default_value (Fl_Shortcut shortcut)
  Set the default shortcut.
• Fl_Shortcut Button (int X, int Y, int W, int H, const char *l=0)
  Construct a shortcut button.
• Fl_Shortcut value ()
  Return the user selected shortcut.
• void value (Fl_Shortcut shortcut)
  Set the displayed shortcut.

Protected Member Functions

• void do_end_hot_callback ()
  Call the callback if the user is interested.
• void draw () FL_OVERRIDE
  Draw the textual representation of the shortcut button.
• int handle (int) FL_OVERRIDE
  Handle keystrokes to catch the user’s shortcut.
Protected Attributes

- Fl_Shortcut shortcut_value

Additional Inherited Members

33.124.1 Detailed Description

A button that allows the user to type a key combination to create shortcuts. After clicked once, the button catches the following keyboard events and records the pressed keys and all modifiers. It draws a text representation of the shortcut. The backspace key deletes the current shortcut. A second click on the button or moving focus makes the last shortcut permanent. The Shortcut button calls the user callback after every change if FL_WHEN_CHANGED is set, and when the button is no longer recording shortcuts if FL_WHEN_RELEASE is set.

33.124.2 Constructor & Destructor Documentation

33.124.2.1 Fl_Shortcut_Button()

Fl_Shortcut_Button::Fl_Shortcut_Button ( int X, int Y, int W, int H, const char * l = 0 )

Construct a shortcut button.

Parameters

<table>
<thead>
<tr>
<th>X,Y,W,H</th>
<th>position and size of the button</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>label text when no shortcut is set</td>
</tr>
</tbody>
</table>

33.124.3 Member Function Documentation

33.124.3.1 default_value() [1/2]

Fl_Shortcut Fl_Shortcut_Button::default_value ( )

Return the default shortcut.

Returns

- shortcut encoded as key and modifier

33.124.3.2 default_value() [2/2]

void Fl_Shortcut_Button::default_value ( Fl_Shortcut shortcut )

Set the default shortcut.

If set, and additional 'reverse' button appears that the user can click to reset the shortcut to some default value (including 0).
**Parameters**

#### in shortcut encoded as key and modifier

33.124.3.3 draw()

```cpp
void Fl_Shortcut_Button::draw (void) [protected], [virtual]
```

Draw the textual representation of the shortcut button.
When the button can receive shortcut key events, it's "hot". A hot button is drawn in selection color. A cold button is drawn as a regular text box containing a human readable version of the shortcut key.
Reimplemented from Fl_Button.

33.124.3.4 handle()

```cpp
int Fl_Shortcut_Button::handle (int e) [protected], [virtual]
```

Handle keystrokes to catch the user's shortcut.
Reimplemented from Fl_Button.

33.124.3.5 value()[1/2]

```cpp
Fl_Shortcut Fl_Shortcut_Button::value ( )
```

Return the user selected shortcut.
Returns

- shortcut encoded as key and modifier

33.124.3.6 value()[2/2]

```cpp
void Fl_Shortcut_Button::value ( Fl_Shortcut shortcut )
```

Set the displayed shortcut.

**Parameters**

#### in shortcut encoded as key and modifier

The documentation for this class was generated from the following files:
- Fl_Shortcut_Button.H
- Fl_Shortcut_Button.cxx

### 33.125 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:
33.126 Fl_Simple_Terminal Class Reference

Public Member Functions

- Fl_Simple_Counter (int X, int Y, int W, int H, const char *L=0)

Additional Inherited Members

33.125.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

The documentation for this class was generated from the following files:

- Fl_Simple_Counter.H
- Fl_Counter.cxx

33.126 Fl_Simple_Terminal Class Reference

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:

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.

- `bool ansi_show_unknown () const`
  See if we should show unknown ANSI sequences with '¿' or not.

- `void ansi_show_unknown (bool val)`
  Enable showing unknown ESC sequences with the '¿' character.

- `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 () const`
  Get the current style char used for style buffer.

- `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`
  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_Terminal ()`
  Destructor for this widget; removes any internal allocations for the terminal, including text buffer, style buffer, etc.
Protected Member Functions

- void **append_ansi** (const char ∗s, int len)
  Handle appending string with ANSI escape sequences, and other 'special' character processing (such as backspaces).
- void **backspace_buffer** (unsigned int count)
  Destructive backspace from end of existing buffer() for specified count characters.
- void **draw ()** FL_OVERRIDE
  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 **handle_backspace ()**
  Handle a Unicode aware backspace.
- void **unknown_escape ()**
- 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

33.126.1 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[m"```
tty->printf("The time is now: \033[32m%s\033[0m", date_time_str);

Example application:
#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));
}

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

33.126.1 Style Tables For Color/Font/Fontsize Control

Internally this widget derives from Fl_Text_Display, and therefore inherits some of its idiosyncracies. In particular, when colors are used, the base class's concept of a 'style table' is used. The 'style table' is similar to a color mapped image; where each pixel is a single value that is an index into a table of colors to minimize per-pixel memory use.

The style table has a similar goal; since every character in the terminal can potentially be a different color, instead of managing several integer attribute values per-character, a single character for each character is used as an index into the style table, choosing one of the available color/font/weight/size values available. This saves on as much as 3 to 4 times the memory use, useful when there's a large amount of text.

When ansi() is set to 'true', ANSI sequences of the form "\033[#m" can be used to select different colors, font faces, font weights (bold,italic..), and font sizes, where '#' is the index number into the style table. Example:
"\033[0mThis text uses the 1st entry in the style table\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"etc..

There is a built-in style table that provides some commonly used ANSI colors for "\033[30m" through "\033[37m" (blk,red,grn,yel,blu,mag,cyn,wht), and a brighter version of those colors for "\033[40" through "\033[47m". See ansi(bool) for more info.

You can also supply a custom style table using style_table(Style_Table_Entry *,int,int), allowing you to define your own color/font/weight/size combinations. See that method's docs for more info.

All style index numbers are rounded to the size of the style table (via modulus) to protect the style array from overruns.

33.126.2 Member Function Documentation

33.126.2.1 ansi() [1/2]

bool Fl_Simple_Terminal::ansi ( ) const
Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.
When true, ANSI sequences in the text stream control color, font and font sizes of text (e.g. "\033[41mThis is Red\033[0m"). For more info, see ansi(bool).
33.126.2.2 **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 Face + Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>\033[0m</td>
<td>&quot;Normal&quot;</td>
<td>FL_COURIER, 14</td>
<td>Resets to default color/font/weight/size</td>
</tr>
<tr>
<td>\033[30m</td>
<td>Medium Black</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[31m</td>
<td>Medium Red</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[32m</td>
<td>Medium Green</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[33m</td>
<td>Medium Yellow</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[34m</td>
<td>Medium Blue</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[35m</td>
<td>Medium Magenta</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[36m</td>
<td>Medium Cyan</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[37m</td>
<td>Medium White</td>
<td>FL_COURIER, 14</td>
<td>The color when \033[0m reset is used</td>
</tr>
<tr>
<td>\033[40m</td>
<td>Bright Black</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[41m</td>
<td>Bright Red</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[42m</td>
<td>Bright Green</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[43m</td>
<td>Bright Yellow</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[44m</td>
<td>Bright Blue</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[45m</td>
<td>Bright Magenta</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[46m</td>
<td>Bright Cyan</td>
<td>FL_COURIER, 14</td>
<td></td>
</tr>
<tr>
<td>\033[47m</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 33.39 Fl_Simple_Terminal built-in ANSI sequences](image)

### 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()`
33.126.2.3  ansi_show_unknown() [1/2]

bool Fl_Simple_Terminal::ansi_show_unknown ( 
    void ) const

See if we should show unknown ANSI sequences with ‘¿’ or not.

See also

ansi_show_unknown(bool)

33.126.2.4  ansi_show_unknown() [2/2]

void Fl_Simple_Terminal::ansi_show_unknown ( 
    bool val )

Enable showing unknown ESC sequences with the ‘¿’ character.
By default this is off, and unknown escape sequences are silently ignored.

See also

ansi_show_unknown()

33.126.2.5  append()

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 if ansi(bool) is set to ‘true’, ANSI 'ESC' sequences (such as ESC[1m) and other control characters (such as backspace) are handled.

Parameters

<table>
<thead>
<tr>
<th>s</th>
<th>string to append.</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>optional length of string can be specified if known to save the internals from having to call strlen()</td>
</tr>
</tbody>
</table>

See also

printf(), vprintf(), text(), clear()

33.126.2.6  backspace_buffer()

void Fl_Simple_Terminal::backspace_buffer ( 
    unsigned int count ) [protected]

Destructive backspace from end of existing buffer() for specified count characters.
Takes into account multi-byte (Unicode) chars. So if count is 3, last 3 chars are deleted from end of buffer.

33.126.2.7  clear()

void Fl_Simple_Terminal::clear ( )

Clears the terminal's screen and history.
Cursor moves to top of window.

33.126.2.8  current_style()

int Fl_Simple_Terminal::current_style ( ) const
Get the current style char used for style buffer. This character appends in parallel with any text in the text buffer to specify the per-character styling. This is typically 'A' for the first entry, 'B' for the second entry, etc. This value is changed by `current_style_index(int)`.

See also

`current_style_index(int)`

### 33.126.2.9 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)`

### 33.126.2.10 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[0m"
tty->append("Back to normal text.\n");
```

This value can also be changed by an ANSI sequence like \`\033[#m\`, where # would be a new style index value. So if the application executes: `term->append("\033[4mTesting")`, then `current_style_index()` will be left set to 4.

The index number specified should be within the number of items in the current style table. Values larger than the table will be clamped to the size of the table with a modulus operation. Effective only when `ansi(bool)` is 'true'.

### 33.126.2.11 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_Group`.

### 33.126.2.12 enforce_history_lines()

```cpp
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.
33.126.2.13  

**enforce_stay_at_bottom()**

```cpp
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.

33.126.2.14  

**handle_backspace()**

```cpp
void Fl_Simple_Terminal::handle_backspace ( ) [protected]
```

Handle a Unicode aware backspace.
This flushes the string parsed so far to Fl_Text_Display, then lets Fl_Text_Display handle the unicode aware backspace.

33.126.2.15  

**history_lines()** [1/2]

```cpp
int Fl_Simple_Terminal::history_lines ( ) const
```

Get the maximum number of terminal history lines last set by history_lines(int).
-1 indicates an unlimited scroll history.

See also

- history_lines(int)

33.126.2.16  

**history_lines()** [2/2]

```cpp
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

<table>
<thead>
<tr>
<th>maxlines</th>
<th>Maximum number of lines kept on the terminal buffer history. Use -1 for an unlimited scroll history. A value of 0 is not recommended.</th>
</tr>
</thead>
</table>

33.126.2.17  

**normal_style_index()** [1/2]

```cpp
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)

33.126.2.18  

**normal_style_index()** [2/2]

```cpp
void Fl_Simple_Terminal::normal_style_index ( int val )
```

Sets the style table index used by the ANSI terminal reset sequence "\033[0m", which resets the current drawing color/font/weight/size to "normal".
Effective only when ansi(bool) is 'true'.

See also

ansi(bool), style_table(Style_Table_Entry*, int, int)

Note

Changing this value does not change the current drawing color. To change that, use current_style_index(int).

### 33.126.2.19 printf()

```cpp
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:

```cpp
#include <FL/Fl_Simple_Terminal.H>

int main(..) {
    Fl_Simple_Terminal *tty = new Fl_Simple_Terminal(..);
    tty->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
    tty->printf("The current PID is %ld.\n", (long)getpid());
    ...
}
```

Note

See Fl_Text_Buffer::vprintf() for limitations.

Parameters

- **fmt** is a printf format string for the message text.

### 33.126.2.20 remove_lines()

```cpp
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

### 33.126.2.21 stay_at_bottom()[1/2]

```cpp
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.
### 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.

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'.

### style_table()

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)

### style_table() [2/2]

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) = (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):

```cpp
int main() {
    // Our custom style table
    Fl_Text_Display::Style_Table_Entry mystyle[] = {
        { 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
    }
    // Font Color Font Face Font Size Index ANSI Sequence
    // ---------- ---------------- --------- ----- -------------
    // FL_WHITE FL_COURIER_BOLD 14  0  \033[0m  ("default")
    // FL_RED FL_COURIER_BOLD 14  1  \033[1m
    // FL_WHITE FL_HELVETICA 14  2  \033[2m
}
```
// Create terminal, enable ANSI and our style table
tty = new Fl_Simple_Terminal(..);
tty->ansi(true); // enable ANSI codes
tty->style_table(&mystyle[0], sizeof(mystyle), 0); // use our custom style table

// Now write to terminal, with ANSI that uses our style table
tty->printf("033[0mNormal Text
033[1mRed Courier Text
n");
tty->append("033[2mWhite Helvetica\033[0mBack to normal.\n");

Note

Changing the style table clears 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 'stable' 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>

33.126.2.25 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).

33.126.2.26 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.

33.126.2.27 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()

33.126.2.28 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(), 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

33.127 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
  |    
  v    
Fl_Group
  |    
  v    
Fl_Window
  |    
  v    
Fl_Single_Window
  |    
  v    
Fl_Menu_Window
```

Public Member Functions

• Fl_Single_Window (int W, int H, const char ∗l=0)
  Creates a new Fl_Single_Window widget using the given size, and label (title) string.
• Fl_Single_Window (int X, int Y, int W, int H, const char ∗l=0)
  Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.
• void show () FL_OVERRIDE
  Makes a widget visible.
• void show (int a, char ∗∗b)
  Same as Fl_Window::show(int a, char ∗∗b)
Additional Inherited Members

33.127.1 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.

33.127.2 Member Function Documentation

33.127.2.1 show()

```cpp
void Fl_Single_Window::show ( ) [virtual]
Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it and all of its parents.
Changing it will send FL_SHOW or FL_HIDE events to the widget. Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget. redraw() is called if necessary on this or the parent.
```

See also hide(), visible(), visible_r()

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Single_Window.H
- Fl_Single_Window.cxx

33.128 Fl_Slider Class Reference

The Fl_Slider widget contains a sliding knob inside a box.
#include <Fl_Slider.H>

Inheritance diagram for Fl_Slider:

```
Fl_Widget
   \-----\-----\-----\-----\-----\-----\-----\-----
       |                   |                   |
       v                   v                   v
Fl_Valuator
       |                   |                   |
       v                   v                   v
Fl_Slider
```

Public Member Functions

- void bounds (double a, double b)
  *Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char *L=0)
  *Creates a new Fl_Slider widget using the given position, size, and label string.
- Fl_Slider (uchar t, int X, int Y, int W, int H, const char *L)
  *Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) FL_OVERRIDE
  *Handles the specified event.
• int scrollvalue (int pos, int size, int first, int total)
  *Sets the size and position of the sliding knob in the box.*

• Fl_Boxtype slider () const
  *Gets the slider box type.*

• void slider (Fl_Boxtype c)
  *Sets the slider box type.*

• float slider_size () const
  *Get the dimensions of the moving piece of slider.*

• void slider_size (double v)
  *Set the dimensions of the moving piece of slider.*

**Protected Member Functions**

• void draw () FL_OVERRIDE
  *Draws the widget.*

• void draw (int, int, int, int)

• int handle (int, int, int, int)

**Additional Inherited Members**

### 33.128.1 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 33.40 Fl_Slider](image)

**33.128.2 Constructor & Destructor Documentation**
33.128.2.1 Fl_Slider()

Fl_Slider::Fl_Slider (  
    int X,  
    int Y,  
    int W,  
    int H,  
    const char ∗L = 0 )

Creates a new Fl_Slider widget using the given position, size, and label string.  
The default boxtype is FL_DOWN_BOX.

33.128.3 Member Function Documentation

33.128.3.1 bounds()

void Fl_Slider::bounds (  
    double a,  
    double b )

Sets the minimum (a) and maximum (b) values for the valuater widget.  
if at least one of the values is changed, a partial redraw is asked.

33.128.3.2 draw()

void Fl_Slider::draw ( ) [protected], [virtual]

Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed.  
If your widget must be redrawn as soon as possible, call redraw() instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

    Fl_Widget ∗s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
    s->draw(); // calls Fl_Scrollbar::draw()  

Implements Fl_Widget.  
Reimplemented in Fl_Value_Slider.

33.128.3.3 handle()

int Fl_Slider::handle (  
    int event ) [virtual]

Handles the specified event.  
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short- 
circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in  | event | the kind of event received |

Return values

| 0   | if the event was not used or understood |
| 1   | if the event was used and can be deleted |
See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Value_Slider.

### 33.128.3.4 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>pos</td>
<td>position of first line displayed</td>
</tr>
<tr>
<td>int</td>
<td>size</td>
<td>size of window in lines</td>
</tr>
<tr>
<td>int</td>
<td>first</td>
<td>number of first line</td>
</tr>
<tr>
<td>int</td>
<td>total</td>
<td>total number of lines</td>
</tr>
</tbody>
</table>

Returns Fl_Valuator::value(p)

### 33.128.3.5 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

### 33.129 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
```

**Classes**

- class Fl_Spinner_Input
Public Member Functions

- **Fl_Color color () const**

  Returns the background color of the spinner widget's input field.

- **void color (Fl_Color v)**

  Sets the background color of the spinner widget's input field.

- **Fl_Spinner (int X, int Y, int W, int H, const char ∗L=0)**

  Creates a new Fl_Spinner widget using the given position, size, and label string.

- **const char ∗ format () const**

  Returns the format string for the value.

- **void format (const char ∗f)**

  Sets the format string for the value.

- **int handle (int event) FL_OVERRIDE**

  Handles the specified event.

- **double maximum () const**

  Gets the maximum value of the widget.

- **void maximum (double m)**

  Sets the maximum value of the widget.

- **int maximum_size () const**

  Returns the maximum width of the input field.

- **void maximum_size (int m)**

  Sets the maximum width of the input field.

- **double minimum () const**

  Gets the minimum value of the widget.

- **void minimum (double m)**

  Sets the minimum value of the widget.

- **void range (double a, double b)**

  Sets the minimum and maximum values for the widget.

- **void resize (int X, int Y, int W, int H) FL_OVERRIDE**

  Resizes the Fl_Group widget and all of its children.

- **Fl_Color selection_color () const**

  Returns the selection color of the spinner widget's input field.

- **void selection_color (Fl_Color val)**

  Sets the selection color of the spinner widget's input field.

- **double step () const**

  Gets the amount to change the value when the user clicks a button.

- **void step (double s)**

  Sets or returns the amount to change the value when the user clicks a button.

- **Fl_Color textcolor () const**

  Gets the color of the text in the input field.

- **void textcolor (Fl_Color c)**

  Sets the color of the text in the input field.

- **Fl_Font textfont () const**

  Gets the font of the text in the input field.

- **void textfont (Fl_Font f)**

  Sets the font of the text in the input field.

- **Fl_Fontsize textsize () const**

  Gets the size of the text in the input field.

- **void textsize (Fl_Fontsize s)**

  Sets the size of the text in the input field.

- **uchar type () const**
Gets the numeric representation in the input field.

- **void** `type (uchar v)`
  
  Sets the numeric representation in the input field.

- **double** `value () const`
  
  Gets the current value of the widget.

- **void** `value (double v)`
  
  Sets the current value of the input widget.

- **int** `wrap () const`
  
  Gets the wrap mode of the Fl_Spinner widget.

- **void** `wrap (int set)`
  
  Sets whether the spinner wraps around at upper and lower bounds.

**Protected Member Functions**

- **void** `draw ()` **FL_OVERRIDE**
  
  Draws the widget.

**Protected Attributes**

- **Fl_Repeat_Button** `down_button_`
- **Fl_Spinner_Input** `input_`
- **Fl_Repeat_Button** `up_button_`

**Additional Inherited Members**

33.129.1 Detailed Description

This widget is a combination of a numerical input widget and repeat buttons. The user can either type into the input area or use the buttons to change the value.

![Fl_Spinner widget](image)

Figure 33.41 Fl_Spinner widget

33.129.2 Constructor & Destructor Documentation

33.129.2.1 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.
33.129.3  Member Function Documentation

33.129.3.1  draw()

```cpp
void Fl_Spinner::draw ( ) [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call `redraw()` instead.

Override this function to draw your own widgets.

If you ever need to call another widget’s draw method `from within your own draw()` method, e.g. for an embedded scrollbar, you can do it (because `draw()` is virtual) like this:

```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Reimplemented from `Fl_Group`.

33.129.3.2  handle()

```cpp
int Fl_Spinner::handle ( int event ) [virtual]
```

Handles the specified event.

You normally don’t call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited `handle()` method in your overridden method so that you don’t short-circuit events that you don’t handle. In this last case you should return the callee retval.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></td>
<td>the kind of event received</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

See also

- `Fl_Event`

Reimplemented from `Fl_Group`.

33.129.3.3  resize()

```cpp
void Fl_Spinner::resize ( int X, int Y, int W, int H ) [virtual]
```

Resizes the `Fl_Group` widget and all of its children.

The `Fl_Group` widget first resizes itself, and then it moves and resizes all its children according to the rules documented for `Fl_Group::resizable(Fl_Widget*)`

See also

- `Fl_Group::resizable(Fl_Widget*)`
- `Fl_Group::resizable()`
- `Fl_Widget::resize(int,int,int,int)`
Reimplemented from Fl_Group.

### 33.129.3.4 step() [1/2]

double Fl_Spinner::step() const [inline]

Get the amount to change the value when the user clicks a button.

See also

Fl_Spinner::step(double)

### 33.129.3.5 step() [2/2]

void Fl_Spinner::step (double s)

Sets or returns the amount to change the value when the user clicks a button. Before setting step to a non-integer value, the spinner type() should be changed to floating point.

See also

double Fl_Spinner::step() const

### 33.129.3.6 type() [1/2]

uchar Fl_Spinner::type() const [inline]

Get the numeric representation in the input field.

See also

Fl_Spinner::type(uchar)

### 33.129.3.7 type() [2/2]

void Fl_Spinner::type (uchar v)

Sets the numeric representation in the input field. Valid values are FL_INT_INPUT and FL_FLOAT_INPUT. Also changes the format() template. Setting a new spinner type via a superclass pointer will not work.

Note

type() is not a virtual function.

### 33.129.3.8 value()

void Fl_Spinner::value (double v) [inline]

Sets the current value of the input widget. Before setting value to a non-integer value, the spinner type() should be changed to floating point.

### 33.129.3.9 wrap() [1/2]

int Fl_Spinner::wrap () const [inline]

Gets the wrap mode of the Fl_Spinner widget.
void wrap(int)

Since
1.4.0

33.129.3.10 wrap() [2/2]

Sets whether the spinner wraps around at upper and lower bounds. If wrap mode is on the spinner value is set to the minimum() or maximum() if the value exceeds the upper or lower bounds, resp., if it was changed by one of the buttons or the FL_Up or FL_Down keys. The spinner stops at the upper and lower bounds if wrap mode is off.
The default wrap mode is on for backwards compatibility with FLTK 1.3.x and older versions.

Note

Wrap mode does not apply to the input field if the input value is edited directly as a number. The input value is always clipped to the allowed range as if wrap mode was off when the input field is left (i.e. loses focus).

See also
minimum(), maximum()

Parameters

| in set | non-zero sets wrap mode, zero resets wrap mode |

Since
1.4.0

The documentation for this class was generated from the following files:
• Fl_Spinner.H
• Fl_Spinner.cxx

33.130 Fl_Spinner::Fl_Spinner_Input Class Reference

Inheritance diagram for Fl_Spinner::Fl_Spinner_Input:

```
+----------------+                  +----------------+                  +----------------+
|                |                  |                |                  |                |
| Fl_Widget      |                  | Fl_Input       |                  | Fl_Spinner     |
|                |                  |                |                  |::Fl_Spinner_Input |
|                |                  |                |                  |                |
```

Public Member Functions

• Fl_Spinner_Input (int X, int Y, int W, int H)
• int handle (int event) FL_OVERRIDE
  Handles events of Fl_Spinner’s embedded input widget.

Additional Inherited Members

33.130.1 Member Function Documentation

33.130.1.1 handle()

```c
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

33.131 Fl_String Class Reference

Fl_String is the basic string class for FLTK.
#include <Fl_String.H>
Inherited by HV_Edit_Buffer.

Public Member Functions

- **Fl_String & append** (char c)
  Append a single byte.
- **Fl_String & append** (const char *src, int n_ins=npos)
  Append a C-style string or data.
- **Fl_String & append** (const Fl_String &src)
  Append another string.
- **Fl_String & assign** (const char *cstr)
  Assign a C-style string.
- **Fl_String & assign** (const char *str, int size)
  Assign a data block of size bytes.
- **Fl_String & assign** (const Fl_String &str)
  Copy another string.
- char at (int pos) const
  Returns the character at specified bounds checked location.
- const char * c_str () const
  Return a pointer to the NUL terminated string.
- int capacity () const
  Return the number of chars that are allocated for storage.
- void clear ()
  Set an empty string.
- char * data ()
  Return a pointer to the writable NUL terminated string.
- const char * data () const
  Return a pointer to the NUL terminated string.
void debug (const char *info=0) const
Write some details about the string to stdout.

bool empty () const
Checks if the string is empty.

Fl_String & erase (int at, int n_del)
Erase some bytes within a string.

Fl_String ()
Allocate an empty string.

Fl_String (const char *cstr)
Constructor from a C-style string.

Fl_String (const char *str, int size)
Constructor from data of size bytes.

Fl_String (const Fl_String &str)
Copy constructor.

void hexdump (const char *info=0) const
Write some details about the string to stdout, followed by a hex dump of the string.

Fl_String & insert (int at, const char *src, int n_ins=npos)
Insert a C-style string or data.

Fl_String & insert (int at, const Fl_String &src)
Insert another string.

Fl_String & operator+= (char c)
Append a single byte.

Fl_String & operator+= (const char *src)
Append a C-style string or data.

Fl_String & operator+= (const Fl_String &src)
Append another string.

Fl_String & operator= (const char *cstr)
Assign a C-style string.

Fl_String & operator= (const Fl_String &str)
Copy assignment operator.

char & operator[] (int n)
Returns a reference to the character at specified location.

char operator[] (int n) const
Returns the character at specified location.

void pop_back ()
Remove the last character.

void push_back (char c)
Append a single character.

Fl_String & replace (int at, int n_del, const char *src, int n_ins=npos)
Replace part of the string with a C-style string or data.

Fl_String & replace (int at, int n_del, const Fl_String &src)
Replace part of the string with another string.

void reserve (int n)
Reserve n bytes for storage.

void resize (int n)
Resizes the string to n characters.

void shrink_to_fit ()
Shrink the capacity to fit the current size.

int size () const
Returns the number of bytes in the string.

int strlen () const
Returns the number of bytes until the first NULL byte.

- **Fl::String substr** (int pos=0, int n=pos) const
  Return a substring from a string.

- **~Fl::String**
  Destructor.

**Static Public Attributes**

- static const int npos = INT_MAX
  Indicate a maximum value or error.

**Static Protected Attributes**

- static const char NUL = 0

### 33.131.1 Detailed Description

**Fl::String** is the basic string class for FLTK.

In this version **Fl::String** can be used to store strings, copy strings, move strings, and do basic string manipulation. **Fl::String** implements a subset of std::string with a couple of extensions. std::string should be a drop-in if we ever decide to allow templates and the std library.

**Fl::String** always maintains a trailing NULL byte, but can also contain NULL bytes inside the string 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. c_str() and data() can be an empty string "", but never be NULL.

The method size() returns the full string size, whether the string contains embedded NULL bytes or not. The special method **Fl::String::strlen()** returns the length of the string up to the first NULL.

All methods of **Fl::String** work on a byte level. They are not UTF-8 aware, but may hold and manipulate UTF-8 strings if done with care.

Since

1.4.0

### 33.131.2 Constructor & Destructor Documentation

#### 33.131.2.1 Fl::String()

Fl::String::Fl::String (const Fl::String & str)

Copy constructor.

**Parameters**

- **in str** copy from another Fl::String

#### 33.131.2.2 Fl::String()

Fl::String::Fl::String (const char * cstr)

Constructor from a C-style string.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>cstr</em></td>
<td>a NUL terminated C-style string</td>
</tr>
</tbody>
</table>

### 33.131.3.3 Fl_String() [3/3]

Fl_String::Fl_String (  
    const char * str,  
    int size )  

Constructor from data of *size* bytes.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>str</em></td>
<td>a block of data that may contain NUL characters</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>size</em></td>
<td>number of bytes to copy</td>
</tr>
</tbody>
</table>

### 33.131.3 Member Function Documentation

#### 33.131.3.1 append() [1/3]

Fl_String & Fl_String::append (  
    char c )  

Append a single byte.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>c</em></td>
<td>single byte character</td>
</tr>
</tbody>
</table>

Returns

    self

#### 33.131.3.2 append() [2/3]

Fl_String & Fl_String::append (  
    const char * src,  
    int n_ins = npos )  

Append a C-style string or data.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>src</em></td>
<td>copy bytes from here</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>n_ins</em></td>
<td>optional number of bytes to copy - if not set, copy C-style string</td>
</tr>
</tbody>
</table>
Returns

self

33.131.3.3 append() [3/3]

Fl_String & Fl_String::append ( const Fl_String & src )

Append another string.

Parameters

| in | src | copy string from here |

Returns

self

33.131.3.4 assign() [1/3]

Fl_String & Fl_String::assign ( const char * cstr )

Assign a C-style string.

Parameters

| in | cstr | a NUL terminated C-style string |

Returns

self

33.131.3.5 assign() [2/3]

Fl_String & Fl_String::assign ( const char * str, int size )

Assign a data block of size bytes.

Parameters

| in  | str | a block of data that may contain NUL characters |
| in  | size | number of bytes to copy |

Returns

self

33.131.3.6 assign() [3/3]

Fl_String & Fl_String::assign ( const Fl_String & str )
Copy another string.

Parameters

in  str  copy from another Fl_String

Returns

self

33.131.3.7  at()

char Fl_String::at (  
    int n ) const

Returns the character at specified bounds checked location.

Parameters

in  n  index of character

Returns

character at that index, or NUL if out of bounds

33.131.3.8  c_str()

const char * Fl_String::c_str ( ) const

Return a pointer to the NUL terminated string.

Returns

reference to non-mutable string

Note

same as Fl_String::data() const

33.131.3.9  capacity()

int Fl_String::capacity ( ) const

Return the number of chars that are allocated for storage.

Returns

string capacity, not counting trailing NUL

33.131.3.10  data() [1/2]

char * Fl_String::data ( )

Return a pointer to the writable NUL terminated string.

Returns

reference to mutable string
33.131.3.11  data() [2/2]

const char * Fl_String::data ( ) const

Returns

reference to non-mutable string

33.131.3.12  debug()

void Fl_String::debug (  
    const char * info = 0 ) const

Write some details about the string to stdout.
Nothing at all is written if info is NULL, otherwise the short info string and details are written to stdout. 
The info string should not be longer than 20 characters to align the debug output of several strings.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>info</th>
<th>short info string or NULL</th>
</tr>
</thead>
</table>

33.131.3.13  empty()

bool Fl_String::empty ( ) const

Checks if the string is empty.

Returns

true if string contains no data

33.131.3.14  erase()

Fl_String & Fl_String::erase (  
    int at,  
    int n_del )

Erase some bytes within a string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>at</th>
<th>erase at this index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n_del</td>
<td>number of bytes to erase</td>
</tr>
</tbody>
</table>

Returns

self

33.131.3.15  hexdump()

void Fl_String::hexdump (  
    const char * info = 0 ) const

Write some details about the string to stdout, followed by a hex dump of the string.
The first part is the same as written by Fl_String::debug(). The following part is a hexadecimal dump of all bytes of 
the string. Embedded null bytes are possible and will be dumped as well.
33.131.3.16 insert() [1/2]

```cpp
Fl_String & Fl_String::insert (  
    int at,  
    const char * src,  
    int n_ins = npos )
```

Insert a C-style string or data.

**Parameters**

- **at**: insert at this index
- **src**: copy bytes from here
- **n_ins**: optional number of bytes to copy - if not set, copy C-style string

**Returns**

self

---

33.131.3.17 insert() [2/2]

```cpp
Fl_String & Fl_String::insert (  
    int at,  
    const Fl_String & src )
```

Insert another string.

**Parameters**

- **at**: insert at this index
- **src**: copy string from here

**Returns**

self

---

33.131.3.18 operator+=() [1/3]

```cpp
Fl_String & Fl_String::operator+= (  
    char c )
```

Append a single byte.

**Parameters**

- **c**: single byte character
33.131.3.19  operator+=() [2/3]

Fl_String & Fl_String::operator+= (  
    const char * src )

Append a C-style string or data.

Parameters

| in  | src  | copy C-style string from here |

Returns

self

33.131.3.20  operator+=() [3/3]

Fl_String & Fl_String::operator+= (  
    const Fl_String & src )

Append another string.

Parameters

| in  | src  | copy string from here |

Returns

self

33.131.3.21  operator=() [1/2]

Fl_String & Fl_String::operator= (  
    const char * cstr )

Assign a C-style string.

Parameters

| in  | cstr  | a NUL terminated C-style string |

Returns

self

33.131.3.22  operator=() [2/2]

Fl_String & Fl_String::operator= (  
    const Fl_String & str )

Copy assignment operator.
Parameters

```
in | str  | copy from another Fl_String
```

Returns

```
self
```

### 33.131.3.23 operator[](1/2)

```
char & Fl_String::operator[](int n)
```

Returns a reference to the character at specified location.

Parameters

```
in | n   | index of character
```

Returns

```
reference to that character, so it can be used as lvalue
```

### 33.131.3.24 operator[](2/2)

```
char Fl_String::operator[](int n) const
```

Returns the character at specified location.

Parameters

```
in | n   | index of character
```

Returns

```
character at that index
```

### 33.131.3.25 push_back()

```
void Fl_String::push_back(char c)
```

Append a single character.

Parameters

```
in | c   | append this byte
```

### 33.131.3.26 replace()

```
Fl_String & Fl_String::replace(int at,
```

Generated by Doxygen
int n_del,
  const char * src,
  int n_ins = npos )

Replace part of the string with a C-style string or data.

Parameters

- **at** erase and insert at this index
- **n_del** number of bytes to erase
- **src** copy bytes from here
- **n_ins** optional number of bytes to copy - if not set, copy C-style string

Returns

- self

### 33.131.3.27 replace() [2/2]

```cpp
Fl_String & Fl_String::replace (  
  int at,
  int n_del,
  const Fl_String & src )
```

Replace part of the string with another string.

Parameters

- **at** erase and insert at this index
- **n_del** number of bytes to erase
- **src** copy string from here

Returns

- self

### 33.131.3.28 reserve()

```cpp
void Fl_String::reserve (  
  int n )
```

Reserve n bytes for storage.
If n is less or equal than size, the capacity is set to size.

Parameters

- **n** requested minimum size, not counting trailing NUL

### 33.131.3.29 resize()

```cpp
void Fl_String::resize (  
  int n )
```

Resizes the string to n characters.
If n is less than the current size, the string will be cropped. If n is more than the current size, the new space will be
filled with NUL characters.

Parameters

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>new size of string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>n</strong></td>
<td></td>
</tr>
</tbody>
</table>

### 33.131.3.30  size()

```cpp
int Fl_String::size ( ) const
```

Returns the number of bytes in the string.

**Returns**

number of bytes in string, not counting trailing NUL

### 33.131.3.31  strlen()

```cpp
int Fl_String::strlen ( ) const
```

Returns the number of bytes until the first NUL byte.

**Returns**

number of bytes in C-style string

### 33.131.3.32  substr()

```cpp
Fl_String Fl_String::substr ( int pos = 0, int n = npos ) const
```

Return a substring from a string.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>copy string from here - if omitted, copy from start</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>pos</strong></td>
<td></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>n</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

a new string

### 33.131.4  Member Data Documentation

#### 33.131.4.1  npos

```cpp
const int Fl_String::npos = INT_MAX [static]
```

Indicate a maximum value or error.

This value is generally used as end of string indicator or as the error indicator by the functions that return a string index.

The documentation for this class was generated from the following files:

- Fl_String.H
- Fl_String.cxx
33.132  Fl_Surface_Device Class Reference

A drawing surface that’s susceptible to receive graphical output.

```cpp
#include <Fl_Device.H>
```

Inheritance diagram for Fl_Surface_Device:

```
Fl_Surface_Device
Fl_Display_Device
Fl_Widget_Surface
Fl_Copy_Surface Fl_EPS_File_Surface Fl_Image_Surface Fl_Paged_Device Fl_SVG_File_Surface
Fl_PostScript_File_Device Fl_Printer
```

### Public Member Functions

- `Fl_Graphics_Driver * driver ()`
  
  Returns the graphics driver of this drawing surface.

- `virtual bool is_current ()`
  
  Is this surface the current drawing surface?

- `virtual void set_current (void)`
  
  Make this surface the current drawing surface.

- `virtual ~Fl_Surface_Device ()`
  
  The destructor.

### Static Public Member Functions

- `static Fl_Surface_Device * pop_current ()`
  
  Removes the top element from the current drawing surface stack, and makes the new top element current.

- `static void push_current (Fl_Surface_Device * new_current)`
  
  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.

- `static Fl_Surface_Device * surface ()`
  
  The current drawing surface.

### Protected Member Functions

- `void driver (Fl_Graphics_Driver * graphics_driver)`
  
  Sets the graphics driver of this drawing surface.

- `virtual void end_current ()`
  
  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.

### 33.132.2 Member Function Documentation

#### 33.132.2.1 end_current()

```cpp
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.

#### 33.132.2.2 is_current()

```cpp
bool Fl_Surface_Device::is_current ( ) [virtual]
```

Is this surface the current drawing surface?

Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, and Fl_Printer.

#### 33.132.2.3 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 *)

Version

1.4.0
33.132.2.4 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

33.132.2.5 set_current()

```cpp
void Fl_Surface_Device::set_current ( void ) [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests. Starting from FLTK 1.4.0, the preferred API to change the current drawing surface is `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

**Note**

It's recommended to use this function only as follows:

- The current drawing surface is the display;
- make current another surface, e.g., an `Fl_Printer` or an `Fl_Image_Surface` object, calling `set_current()` on this object;
- draw to that surface;
- make the display current again with `Fl_Display_Device::display_device()->set_current()`. Don't do any other call to `set_current()` before this one.

Other scenarios of drawing surface changes should be performed via `Fl_Surface_Device::push_current() / Fl_Surface_Device::pop_current()`.

Reimplemented in `Fl_Copy_Surface`, `Fl_Image_Surface`, `Fl_PostScript_File_Device`, and `Fl_Printer`.

33.132.2.6 surface()

```cpp
static Fl_Surface_Device * Fl_Surface_Device::surface ( ) [inline], [static]
```

The current drawing surface.
In other words, the `Fl_Surface_Device` object that currently receives all graphics requests.

**Note**

It's possible to transiently remove the GUI scaling factor in place in the current drawing surface with `fl_override_scale()`.

The documentation for this class was generated from the following files:

- `Fl_Device.H`
- `Fl_Device.cxx`

### 33.133 Fl_SVG_File_Surface Class Reference

A drawing surface producing a Scalable Vector Graphics (SVG) file.

```cpp
#include <Fl_SVG_File_Surface.H>
```

Inheritance diagram for `Fl_SVG_File_Surface`:
Public Member Functions

- **int close ()**
  Closes the FILE pointer where SVG data is output.
- **FILE * file ()**
  Returns the underlying FILE pointer.
- **Fl_SVG_File_Surface (int width, int height, FILE *svg, int(closef)(FILE *)=NULL)**
  Constructor of the SVG drawing surface.
- **void origin (int x, int y) FL_OVERRIDE**
  Computes the coordinates of the current origin of graphics functions.
- **void origin (int x, int y) FL_OVERRIDE**
  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- **int printable_rect (int w, int h) FL_OVERRIDE**
  Computes the width and height of the drawable area of the drawing surface.
- **void translate (int x, int y) FL_OVERRIDE**
  Translates the current graphics origin accounting for the current rotation.
- **void untranslate () FL_OVERRIDE**
  Undoes the effect of a previous translate() call.
- **~Fl_SVG_File_Surface ()**
  Destructor.

Additional Inherited Members

33.133.1 Detailed Description

A drawing surface producing a Scalable Vector Graphics (SVG) file.

This drawing surface allows to store any FLTK graphics in vectorial form in a "Scalable Vector Graphics" file.

Usage example:

```cpp
Fl_Window *win = ...// Window to draw to a .svg file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *svg = fl_fopen("/path/to/mywindow.svg", "w");
if (svg) {
    Fl_SVG_File_Surface *surface = new Fl_SVG_File_Surface(ww, wh, svg);
    Fl_Surface_Device::push_current(surface);
    fl_color(FL_WHITE);
    fl_rectf(0, 0, ww, wh);
    surface->draw_decorated_window(win);
    Fl_Surface_Device::pop_current();
    delete surface; // the .svg file is not complete until the destructor was run
    fclose(svg);
}
```

Note
FLTK uses the PNG and JPEG libraries to encode images to the SVG format. For this reason, class `Fl_SVG_File_Surface` is placed in the fltk_images library. If JPEG is not available at application build time, PNG is enough (but produces a quite larger output). If PNG isn't available either, images don't appear in the SVG output.

33.133.2 Constructor & Destructor Documentation
### 33.133.2.1 Fl_SVG_File_Surface()

```cpp
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**

<table>
<thead>
<tr>
<th>width, height</th>
<th>Width and height of the graphics area in FLTK drawing units</th>
</tr>
</thead>
<tbody>
<tr>
<td>svg</td>
<td>A writable FILE pointer where the SVG data are to be sent. The resulting SVG data are not complete until after destruction of the Fl_SVG_File_Surface object or after calling close().</td>
</tr>
<tr>
<td>closef</td>
<td>If not NULL, the destructor and close() will call closef(svg) after all SVG data has been sent. If NULL, fclose(svg) is called instead. This allows to close the FILE pointer by, e.g., pclose, or, using a function such as &quot;int keep_open(FILE*){return 0;}&quot;, to keep it open after completion of all output to svg. Function closef should return non zero to indicate an error.</td>
</tr>
</tbody>
</table>

### 33.133.2.2 ~Fl_SVG_File_Surface()

```cpp
Fl_SVG_File_Surface::~Fl_SVG_File_Surface ( )
```

Destructor.

The underlying FILE pointer is processed as by close().

### 33.133.3 Member Function Documentation

#### 33.133.3.1 close()

```cpp
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.

#### 33.133.3.2 origin()[1/2]

```cpp
void Fl_SVG_File_Surface::origin (  
    int ∗ x,  
    int ∗ y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

**Parameters**

| out x,y | If non-null, +x and +y are set to the horizontal and vertical coordinates of the graphics origin. |

Reimplemented from Fl_Widget_Surface.
33.133.3  origin()

```cpp
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**

- `x, y`: Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from `Fl_Widget_Surface`.

33.133.4  printable_rect()

```cpp
int Fl_SVG_File_Surface::printable_rect (  
    int * w,  
    int * h  ) [virtual]
```

Computes the width and height of the drawable area of the drawing surface. Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to `origin()`. If the object is derived from class `Fl_Paged_Device`, values account for the user-selected paper type and print orientation and are changed by `scale()` calls.

**Returns**

- 0 if OK, non-zero if any error

Reimplemented from `Fl_Widget_Surface`.

33.133.5  translate()

```cpp
void Fl_SVG_File_Surface::translate (  
    int x,  
    int y  ) [virtual]
```

Translates the current graphics origin accounting for the current rotation. Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.

Reimplemented from `Fl_Widget_Surface`.

33.133.6  untranslate()

```cpp
void Fl_SVG_File_Surface::untranslate (  
    void  ) [virtual]
```

Undoes the effect of a previous `translate()` call.

Reimplemented from `Fl_Widget_Surface`.

The documentation for this class was generated from the following file:

- `Fl_SVG_File_Surface.H`

33.134  Fl_SVG_Image Class Reference

The `Fl_SVG_Image` class supports loading, caching and drawing of scalable vector graphics (SVG) images.

```cpp
#include <Fl_SVG_Image.H>
```

Inheritance diagram for `Fl_SVG_Image`:
Public Member Functions

• `Fl_SVG_Image * as_svg_image () FL_OVERRIDE`
  Returns whether an image is an `Fl_SVG_Image` or not.

• `void color_average (Fl_Color c, float i) FL_OVERRIDE`
  The `color_average()` method averages the colors in the image with the provided FLTK color value.

• `Fl_Image * copy () const`

• `Fl_SVG_Image * copy (int W, int H) const FL_OVERRIDE`
  Creates a resized copy of the image.

• `void desaturate () FL_OVERRIDE`
  The `desaturate()` method converts an image to grayscale.

• `void draw (int X, int Y)`

• `void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE`
  Draws the image to the current drawing surface with a bounding box.

• `Fl_SVG_Image (const char *filename)`
  Load an SVG image from a file.

• `Fl_SVG_Image (const char *sharedname, const char *svg_data)`
  Load an SVG image from memory.

• `Fl_SVG_Image (const char *sharedname, const unsigned char *svg_data, size_t length)`
  Load an SVG image from memory.

• `void normalize () FL_OVERRIDE`
  Makes sure the object is fully initialized.

• `void resize (int width, int height)`
  Have the svg data (re-)rasterized using the given width and height values.

• `virtual ~Fl_SVG_Image ()`
  The destructor frees all memory and server resources that are used by the SVG image.

Public Attributes

• `bool proportional`
  Set this to `false` to allow image re-scaling that alters the image aspect ratio.

Additional Inherited Members

33.134.1 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_SVG_Image::fail()` to check if the `Fl_SVG_Image` failed to load. `fail()` returns ERR_FILE_ACCESS if the file could not be opened or read, and ERR_FORMAT if the SVG format could not be decoded. If the image has loaded correctly, `w()`, `h()`, and `d()` should return values greater than zero.

Rasterization is not done until the image is first drawn or `resize()` or `normalize()` is called. Therefore, `array` is NULL until then. The delayed rasterization ensures an `Fl_SVG_Image` is always rasterized to the exact screen resolution at which it is drawn.

The `Fl_SVG_Image` class draws images computed by nanosvg with the following known limitations.
• text between <text> and </text> marks,
• image elements, and
• <use> statements

are not rendered.

The FLTK library can optionally be built without SVG support; in that case, class Fl_SVG_Image is unavailable.

Example of displaying a hard-coded svg file:
```c++
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>

// A black rotated rectangle
const char *svg_data = "<svg viewBox="0 0 200 200" version="1.1">
  <rect x="25" y="50" width="150" height="100" fill="black">
    <transform>rotate(45 100 100)</transform>
  </rect>
</svg>";

int main(int argc, char **argv) {
  Fl_SVG_Image *svg = new Fl_SVG_Image(0, svg_data); // create SVG object
  Fl_Window *win = new Fl_Window(720, 486, "svg test");
  Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
  box->image(svg); // assign svg object to Fl_Box
  win->end();
  win->show(argc, argv);
  return(Fl::run());
}
```

Example of displaying an svg image from a file:
```c++
#include <errno.h>  // errno
#include <string.h> // strerror
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/fl_message.H>

int main(int argc, char **argv) {
  Fl_Window *win = new Fl_Window(720, 486, "svg test");
  Fl_Box *box = new Fl_Box(0, 0, win->w(), win->h());
  // Load svg image from disk, assign to a box
  const char *svgpath = "/var/tmp/simple.svg";
  Fl_SVG_Image *svg = new Fl_SVG_Image(svgpath); // load SVG object from disk
  switch (svg->fail()) {
    case Fl_Image::ERR_FILE_ACCESS:
      // File couldn't load? show path + os error to user
      fl_alert("%s: %s", svgpath, strerror(errno));
      return 1;
    case Fl_Image::ERR_FORMAT:
      // Parsing error
      fl_alert("%s: couldn't decode image", svgpath);
      return 1;
  }
  box->image(svg); // assign svg object to box
  win->end();
  win->show(argc, argv);
  return(Fl::run());
}
```

Example of fitting an svg image to a resizable Fl_Box:
```c++
#include <FL/Fl_SVG_Image.H>
#include <FL/Fl_Box.H>

class resizable_box : public Fl_Box {
public:
  resizable_box(int w, int h) : Fl_Box(0, 0, w, h, NULL) {}
  virtual void resize(int x, int y, int w, int h) {
    // p3 = proportional, p4 = can_expand
    fl_SVG_Image::scale(w, h, 1, 1);
  }
};

int main(int argc, char **argv) {
  Fl_SVG_Image *svg = new Fl_SVG_Image("/path/to/image.svg");
  resizable_box *box = new resizable_box(w, h);
  box->image(svg);
  box->scale(w, h);
  win->end();
  win->show(argc, argv);
  return(Fl::run());
}
```

33.134.2 Constructor & Destructor Documentation
33.134.2.1  Fl_SVG_Image() [1/3]

Fl_SVG_Image::Fl_SVG_Image (  
    const char * filename )

Load an SVG image from a file. This constructor loads the SVG image from a .svg or .svgz file. The reader recognizes if the data is compressed, and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

| filename | the filename for a .svg or .svgz file |

33.134.2.2  Fl_SVG_Image() [2/3]

Fl_SVG_Image::Fl_SVG_Image (  
    const char * sharedname,  
    const char * svg_data )

Load an SVG image from memory. This constructor loads the SVG image from a block of memory. This version is commonly used for uncompressed text data, but the reader recognizes if the data is compressed, and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

| sharedname | if not NULL, a shared image will be generated with this name |
| svg_data | a pointer to the memory location of the SVG image data |

Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

33.134.2.3  Fl_SVG_Image() [3/3]

Fl_SVG_Image::Fl_SVG_Image (  
    const char * name,  
    const unsigned char * svg_data,  
    size_t length )

Load an SVG image from memory. This constructor loads the SVG image from a block of memory. This version is commonly used for compressed binary data, but the reader recognizes if the data is uncompressed, and reads it as a text block.

Parameters

| name | if not NULL, a shared image will be generated with this name |
| svg_data | a pointer to the memory location of the SVG image data |
| length | of svg_data or 0 if the length is unknown. This will protect memory outside of the svg_data array from illegal read operations for compressed SVG data |

Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

33.134.3  Member Function Documentation

Generated by Doxygen
33.134.3.1 as_svg_image()

```cpp
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.

33.134.3.2 color_average()

```cpp
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.

33.134.3.3 copy()

```cpp
Fl_Image * Fl_SVG_Image::copy ( int W, int H ) const [virtual]
```

Creates a resized copy of the image.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image)
instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- \( w() = \text{data}_w() = W \)
- \( h() = \text{data}_h() = H \)

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Parameters

| in | W, H | Requested width and height of the new image |

Generated by Doxygen
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`.

### 33.134.3.4 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`.

### 33.134.3.5 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`.

### 33.134.3.6 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`.

### 33.134.3.7 resize()

```cpp
void Fl_SVG_Image::resize(int width, int height)
```

Have the svg data (re-)rasterized using the given `width` and `height` values.
By default, the resulting image `w()` and `h()` will be close to `width` and `height` while preserving the width/height
ratio of the SVG data. If `proportional` was set to `false`, the image is rasterized to the exact `width` and `height`
values. In both cases, `data_w()` and `data_h()` values are set to `w()` and `h()`, respectively.

### 33.134.4 Member Data Documentation
33.134.4.1 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

33.135 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)
  
  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 *, 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)

Generated by Doxygen
create a system menu bar using the given list of menu structs

- **int mode**(int i) const
  
  Gets the flags of item i.

- **void mode**(int i, int fl)
  
  Sets the flags of item i.

- **void remove**(int n)
  
  remove an item from the system menu bar

- **void replace**(int index, const char *name)
  
  rename an item from the system menu bar

- **void setonly**(Fl_Menu_Item *item)
  
  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

- **void shortcut**(int i, int s)
  
  Changes the shortcut of item i to n.

- **void update**() FL_OVERRIDE
  
  Updates the menu bar after any change to its items.

- **virtual ~Fl_Sys_Menu_Bar**()
  
  The destructor.

### 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

- **void draw**() FL_OVERRIDE
  
  Draws the widget.

### Additional Inherited Members

#### 33.135.1 Detailed Description

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen. On other than macOS platforms, Fl_Sys_Menu_Bar is a synonym of class Fl_Menu_Bar.

On the macOS platform, replace Fl_Menu_Bar with Fl_Sys_Menu_Bar, and a system menu at the top of the screen will be available. This menu will match an array of Fl_Menu_Item's exactly as in all other FLTK menus (except for the submenu with the application's own name and the 'Window' menu; see below). There is, though, an important difference between an Fl_Sys_Menu_Bar object under macOS and under other platforms: only a single object from this class can be created, because macOS uses a single system menu bar. Therefore, porting to macOS an app that creates, on other platforms, several Fl_Menu_Bar objects, one for each of several windows, is more complex than just replacing Fl_Menu_Bar by Fl_Sys_Menu_Bar.

On the macOS platform, the system menu bar of any FLTK app begins with the Application menu which the FLTK library automatically constructs. Functions Fl_Mac_App_Menu::custom_application_menu_items() and Fl_Sys_Menu_Bar::about() can be used to further customize the Application menu. The FLTK library also automatically constructs and handles a Window menu which can be further customized (or even removed) calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum style). Other member functions of this class allow the app to generate the rest of the system menu bar. It is recommended to localize the system menu bar using the standard Mac OS X localization procedure (see Internationalization).
Changes to the menu state are immediately visible in the menubar when they are made using member functions of the \texttt{Fl\_Sys\_Menu\_Bar} class. Other changes (e.g., by a call to \texttt{Fl\_Menu\_Item::set()}) should be followed by a call to \texttt{update()} to be visible in the menubar across all platforms. macOS global variable \texttt{fl\_sys\_menu\_bar} points to the unique, current system menubar.

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 \texttt{Fl\_Mac\_App\_Menu::custom\_application\_menu\_items()}. For example, to make your own “Appname – Preferences” dialog, you might use:

```cpp
#include <FL/platform.H> // for Fl\_Mac\_App\_Menu class
#include <FL/Fl\_Sys\_Menu\_Bar.H> // for Fl\_Menu\_Item

void prefs\_cb(Fl\_Widget *, void *data)
{
    // ..Open your preferences dialog here..
}

int main(..) {
    // Items to add to the application menu
    static Fl\_Menu\_Item appitems[] = {
        { "Preferences", 0, prefs\_cb, 0, 0 },
        { 0 }, { 0 }
    }
    Fl\_Mac\_App\_Menu::custom\_application\_menu\_items(appitems); // adds it
}
```

..the result being:

![Figure 33.42 Mac Application submenu](image)

### 33.135.2 Member Enumeration Documentation

#### 33.135.2.1 \texttt{window\_menu\_style\_enum}

enum \texttt{Fl\_Sys\_Menu\_Bar::window\_menu\_style\_enum}

Possible styles of the Window menu in the system menu bar.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_window_menu</td>
<td>No Window menu in the system menu bar.</td>
</tr>
<tr>
<td>tabbing_mode_none</td>
<td>No tabbed windows, but the system menu bar contains a Window menu.</td>
</tr>
<tr>
<td>tabbing_mode_automatic</td>
<td>Windows are created by themselves but can be tabbed later.</td>
</tr>
<tr>
<td>tabbing_mode_preferred</td>
<td>Windows are tabbed when created.</td>
</tr>
</tbody>
</table>

### 33.135.3 Constructor & Destructor Documentation
33.135.3.1  

**Fl_Sys_Menu_Bar()**

```c
Fl_Sys_Menu_Bar::Fl_Sys_Menu_Bar (  
    int x,  
    int y,  
    int w,  
    int h,  
    const char ∗ l = 0 
)
```

The constructor.

On Mac OS X, all arguments are unused. On other platforms they are used as by `Fl_Menu_Bar::Fl_Menu_Bar()`.

---

33.135.4  

**Member Function Documentation**

---

33.135.4.1  

**about()**

```c
void Fl_Sys_Menu_Bar::about (  
    Fl_Callback ∗ cb,  
    void ∗ data       ) [static]
```

Attaches a callback to the "About myprog" item of the system application menu.

This cross-platform function is effective only under the MacOS platform.

**Parameters**

| `cb` | a callback that will be called by "About myprog" menu item with NULL 1st argument. |
| `data` | a pointer transmitted as 2nd argument to the callback. |

---

33.135.4.2  

**add() [1/3]**

```c
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

```c
Fl_Menu_::add(const char ∗ label, int shortcut, Fl_Callback ∗, void ∗user_data=0, int flags=0)
```

---

33.135.4.3  

**add() [2/3]**

```c
int Fl_Sys_Menu_Bar::add (  
    const char ∗ label,  
    int shortcut,  
    Fl_Callback ∗ cb,  
    void ∗ user_data = 0,  
    int flags = 0       )
```

Add a new menu item to the system menu bar.

Add to the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.

**Parameters**

| `label` | - new menu item's label |

---
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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)

33.135.4.4 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)

33.135.4.5 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()

33.135.4.6 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)
33.135.4.7  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_Windo::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_Windo::show().
This function does nothing on non MacOS platforms.

Version
1.4

33.135.4.8  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.

33.135.4.9  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)

33.135.4.10  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)

33.135.4.11 menu()

void Fl_Sys_Menu_Bar::menu (const Fl_Menu_Item * m )
create a system menu bar using the given list of menu structs

Author
Matthias Melcher

Parameters

   m  Zero-ending list of Fl_Menu_Item's

33.135.4.12 mode()

void Fl_Sys_Menu_Bar::mode (int i, int fl )
Sets the flags of item i.

See also
Fl_Menu_::mode(int i, int fl)

33.135.4.13 remove()

void Fl_Sys_Menu_Bar::remove (int index )
remove an item from the system menu bar

Parameters

   index  the index of the item to remove

33.135.4.14 replace()

void Fl_Sys_Menu_Bar::replace (int index,
    const char * name )
rename an item from the system menu bar

Parameters

   index  the index of the item to rename
Parameters

| name | the new item name as a UTF8 string |

33.135.4.15  update()

```c
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.

33.135.4.16  window_menu_style()

```c
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

33.136  Fl_Table Class Reference

A table of widgets or other content.

```c
#include <Fl_Table.H>
```

Inheritance diagram for Fl_Table:
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) FL_OVERRIDE**
  Handle resize events if user resizes parent window.
- **int row_header ()**
  Returns if row headers are enabled or not.
- **void row_header (int flag)**
  Enables/disables showing the row headers.
- **Fl_Color row_header_color ()**
  Returns the current row header color.
- **void row_header_color (Fl_Color val)**
  Sets the row header color and causes the screen to redraw.
- **int row_header_width ()**
Returns the current row header width (in pixels).

- **void** row_header_width (int width)
  
  Sets the row header width to n and causes the screen to redraw.

- **int** row_height (int row)
  
  Returns the current height of the specified row as a value in pixels.

- **void** row_height (int row, int height)
  
  Sets the height of the specified row in pixels, and the table is redrawn.

- **void** row_height_all (int height)
  
  Convenience method to set the height of all rows to the same value, in pixels.

- **int** row_position (
  
  Returns the current row scroll position as a row number.

- **void** row_position (int row)
  
  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.

- **int** row_resize ()
  
  Returns if row resizing by the user is allowed.

- **void** row_resize (int flag)
  
  Allows/disallows row resizing by the user.

- **int** row_resize_min ()
  
  Returns the current row minimum resize value.

- **void** row_resize_min (int val)
  
  Sets the current row minimum resize value.

- **int** rows ()
  
  Returns the number of rows in the table.

- **virtual void** rows (int val)
  
  Sets the number of rows in the table, and the table is redrawn.

- **int** scrollbar_size () const
  
  Gets the current size of the scrollbars' troughs, in pixels.

- **void** scrollbar_size (int newSize)
  
  Sets the pixel size of the scrollbars' troughs to newSize, in pixels.

- **void** set_selection (int row_top, int col_left, int row_bot, int col_right)
  
  Sets the region of cells to be selected (highlighted).

- **int** tab_cell_nav () const
  
  Get state of table's 'Tab' key cell navigation flag.

- **void** tab_cell_nav (int val)
  
  Flag to control if Tab navigates table cells or not.

- **void** table_box (Fl_Boxtype val)
  
  Sets the kind of box drawn around the data table, the default being FL_NO_BOX.

- **Fl_Boxtype** table_box (void)
  
  Returns the current box type used for the data table.

- **int** top_row ()
  
  Returns the current top row shown in the table.

- **void** top_row (int row)
  
  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

- **void** visible_cells (int &r1, int &r2, int &c1, int &c2)
  
  Returns the range of row and column numbers for all visible and partially visible cells in the table.

- **void** when (Fl_When flags)
  
  The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

- ~Fl_Table ()
  
  The destructor for Fl_Table.
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 () FL_OVERRIDE
  Draws the entire Fl_Table.
- virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)
  Subclass should override this method to handle drawing the cells.
- int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
  Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.
- void get_bounds (TableContext context, int &X, int &Y, int &W, int &H)
  Returns the (X,Y,W,H) bounding region for the specified 'context'.
- int handle (int e) FL_OVERRIDE
  Handle FLTK events.
- int is_fltk_container ()
  Does the table contain any child fltk widgets?
- void recalc_dimensions ()
  Recalculate the dimensions of the table, and affect any children.
- void redraw_range (int topRow, int botRow, int leftCol, int rightCol)
  Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).
- int row_col_clamp (TableContext context, int &R, int &C)
  Return specified row/col values R and C to within the table’s current row/col limits.
- long row_scroll_position (int row)
  Returns the scroll position (in pixels) of the specified ‘row’.
- void table_resized ()
  Call this if table was resized, to recalculate internal data.
- void table_scrolled ()
  Recalculate internals after a scroll.

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 thi`
  
  Data table’s inner h dimension, inside bounding box. See Table Dimension Diagram.

- `int thw`
  
  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 tow`
  
  top row# of currently visible table on screen

- `int toprow_scrollop`
  
  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 w dimension, inside bounding box. See Table Dimension Diagram.

• int wix
  Table widget's inner x dimension, inside bounding box. See Table Dimension Diagram.

• int wiy
  Table widget's inner y dimension, inside bounding box. See Table Dimension Diagram.

**Additional Inherited Members**

### 33.136.1 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.

![](image1)

*Figure 33.43 table-simple example*

![](image2)

*Figure 33.44 table-as-container example*

When acting as part of a custom widget, events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the setting for `Fl_Table::when()`.

When acting as a container for FLTK widgets, the FLTK widgets maintain themselves. Although the `draw_cell()` method must be overridden, its contents can be very simple. See the `draw_cell()` code in examples/table-simple.cxx.
The following variables are available to classes deriving from Fl_Table:

<table>
<thead>
<tr>
<th>x()</th>
<th>y()</th>
<th>w()</th>
<th>h()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Table widget's outer dimension. The outer edge of the border of the Fl_Table. (Red in the diagram above)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wix</th>
<th>wiw</th>
<th>wih</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Table widget's inner dimension. The inner edge of the border of the Fl_Table. eg. if the Fl_Table's box() is FL_NO_BOX, these values are the same as x() y() w() h(). (Yellow in the diagram above)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tox</th>
<th>toy</th>
<th>tow</th>
<th>toh</th>
</tr>
</thead>
<tbody>
<tr>
<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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tix</th>
<th>tiw</th>
<th>tih</th>
</tr>
</thead>
<tbody>
<tr>
<td>The table's inner dimension. The inner edge of the border around the cells, but inside the row/col headings and scrollbars. AKA the table's clip region. eg. if the table_box() is FL_NO_BOX, these values are the same as tox toy tow toh. (Blue in the diagram above)</td>
<td></td>
<td></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.hxx; 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]

33.136.2 Member Enumeration Documentation
33.136.2.1 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>

33.136.3 Constructor & Destructor Documentation

33.136.3.1 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.

33.136.3.2 ~Fl_Table()

Fl_Table::~Fl_Table ( )
The destructor for Fl_Table. Destroys the table and its associated widgets.

33.136.4 Member Function Documentation

33.136.4.1 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.

33.136.4.2 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:
• **Fl_Table::callback_row()** returns current row
• **Fl_Table::callback_col()** returns current column
• **Fl_Table::callback_context()** returns current table context

**callback_row()** and **callback_col()** will be set to the row and column number the event occurred on. If someone clicked on a row header, `row` will be 0. If someone clicked on a column header, `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",
            (int)R, (int)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
}
};
```

### 33.136.4.3 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.

### 33.136.4.4 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.

### 33.136.4.5 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.
33.136.4.6 child()

Fl_Widget * Fl_Table::child (int n) const [inline]

Returns the child widget by an index.

When using the Fl_Table as a container for FLTK widgets, this method returns the widget pointer from the internal array of widgets in the container.

Typically used in loops, eg:

```cpp
for ( int i=0; i<children(); i++ ) {
    Fl_Widget *w = child(i);
    [..]
}
```

33.136.4.7 children()

int Fl_Table::children ( ) const [inline]

Returns the number of children in the table.

When using the Fl_Table as a container for FLTK widgets, this method returns how many child widgets the table has.

See also

child(int)

33.136.4.8 clear()

virtual void Fl_Table::clear ( ) [inline], [virtual]

Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.

See also

rows(int), cols(int)

Reimplemented in Fl_Table_Row.

33.136.4.9 col_header()

void Fl_Table::col_header (int flag) [inline]

Enable or disable column headers.

If changed, the table is redrawn.

33.136.4.10 col_resize()

void Fl_Table::col_resize (int flag) [inline]

Allows/disallows column resizing by the user.

1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the column headers, col_header() must also be enabled to allow resizing.

33.136.4.11 col_resize_min()

void Fl_Table::col_resize_min (int val) [inline]

Sets the current column minimum resize value.

This is used to prevent the user from interactively resizing any column to be smaller than 'pixels'. Must be a value >=1.
33.136.4.12 `col_width()`

```cpp
void Fl_Table::col_width (  
    int col,  
    int width )
```

Sets the width of the specified column in pixels, and the table is redrawn. `callback()` will be invoked with CONTEXT_RC_RESIZE if the column's width was actually changed, and `when()` is FL_WHEN_CHANGED.

33.136.4.13 `col_width_all()`

```cpp
void Fl_Table::col_width_all (  
    int width ) [inline]
```

Convenience method to set the width of all columns to the same value, in pixels. The screen is redrawn.

33.136.4.14 `cursor2rowcol()`

```cpp
Fl_Table::TableContext Fl_Table::cursor2rowcol (  
    int & R,  
    int & C,  
    ResizeFlag & resizeflag ) [protected]
```

Find row/col for the recent mouse event. Returns the context, and the row/column values in R/C. Also returns 'resizeflag' if mouse is hovered over a resize boundary.

33.136.4.15 `damage_zone()`

```cpp
void Fl_Table::damage_zone (  
    int r1,  
    int c1,  
    int r2,  
    int c2,  
    int r3 = 0,  
    int c3 = 0 ) [protected]
```

Sets the damage zone to the specified row/col values. Calls `redraw_range()`.

33.136.4.16 `do_callback()`

```cpp
void Fl_Table::do_callback (  
    TableContext  context,  
    int  row,  
    int  col ) [inline]
```

Calls the widget callback. Saves the specified 'context', 'row', and 'col' values, so that the user's callback can then access them with the member functions `callback_context()`, `callback_row()` and `callback_col()`.

33.136.4.17 `draw()`

```cpp
void Fl_Table::draw (  
    void ) [protected], [virtual]
```

Draws the entire `Fl_Table`. Lets fltk widgets draw themselves first, followed by the cells via calls to `draw_cell()`. Reimplemented from `Fl_Group`. 

---

Generated by Doxygen
### draw_cell()

```cpp
virtual void Fl_Table::draw_cell ( 
    TableContext context, 
    int R = 0, 
    int C = 0, 
    int X = 0, 
    int Y = 0, 
    int W = 0, 
    int H = 0 ) [inline], [protected], [virtual]
```

Subclass should override this method to handle drawing the cells.

This method will be called whenever the table is redrawn, once per cell. Only cells that are completely (or partially) visible will be told to draw.

`context` will be one of the following:

| Fl_Table::CONTEXT_STARTPAGE | When table, or parts of the table, are about to be redrawn. Use to initialize static data, such as font selections. R/C will be zero, X/Y/W/H will be the dimensions of the table's entire data area. (Useful for locking a database before accessing; see also visible_cells()) |
| Fl_Table::CONTEXT_ENDPAGE   | When table has completed being redrawn. R/C will be zero, X/Y/W/H dimensions of table's data area. (Useful for unlocking a database after accessing) |
| Fl_Table::CONTEXT_ROW_HEADER | Whenever a row header cell needs to be drawn. R will be the row number of the header being redrawn, C will be zero, X/Y/W/H will be the fltk drawing area of the row header in the window |
| Fl_Table::CONTEXT_COL_HEADER | Whenever a column header cell needs to be drawn. R will be zero, C will be the column number of the header being redrawn, X/Y/W/H will be the fltk drawing area of the column header in the window |
| Fl_Table::CONTEXT_CELL     | Whenever a data cell in the table needs to be drawn. R/C will be the row/column of the cell to be drawn, X/Y/W/H will be the fltk drawing area of the cell in the window |
| Fl_Table::CONTEXT_RC_RESIZE | Whenever table or row/column is resized or scrolled, either interactively or via col_width() or row_height(). R/C/X/Y/W/H will all be zero. Useful for fltk containers that need to resize or move the child fltk widgets. |

R and C will be set to the row and column number of the cell being drawn. In the case of row headers, C will be 0.

In the case of column headers, R will be 0.

X/Y/W/H will be the position and dimensions of where the cell should be drawn.

In the case of custom widgets, a minimal `draw_cell()` override might look like the following. With custom widgets it is up to the caller to handle drawing everything within the dimensions of the cell, including handling the selection color. Note all clipping must be handled as well; this allows drawing outside the dimensions of the cell if so desired for 'custom effects'.

```cpp
// This is called whenever Fl_Table wants you to draw a cell
void MyTable::draw_cell(TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0) { 
    static char s[40];
    sprintf(s, "%d/%d", R, C); // text for each cell
    switch ( context ) { 
    case CONTEXT_STARTPAGE:  // Fl_Table telling us it’s starting to draw page
        fl_font(Fl_HELVETICA, 16);
        return;
    case CONTEXT_ROW_HEADER:  // Fl_Table telling us to draw row/col headers
        fl_push_clip(X, Y, W, H);
        { 
            fl_draw_box(Fl_THIN_UP_BOX, X, Y, W, H, color());
        }
    // other cases... 
    }
```

Generated by Doxygen
33.136.4.19 find_cell()

```cpp
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.

33.136.4.20 get_selection()

```cpp
void Fl_Table::get_selection ( 
        int & row_top,
        int & col_left,
        int & row_bot,
        int & col_right )
```

Gets the region of cells selected (highlighted).

Parameters

- **row_top** Returns the top row of selection area
- **col_left** Returns the left column of selection area
- **row_bot** Returns the bottom row of selection area
- **col_right** Returns the right column of selection area

33.136.4.21 handle()

```cpp
int Fl_Table::handle ( 
```
int e) [protected], [virtual]
Handle FLTK events.
Reimplemented from Fl_Group.
Reimplemented in Fl_Table_Row.

33.136.4.22 init_sizes()

void Fl_Table::init_sizes() [inline]
Resets the internal array of widget sizes and positions.
See also
   Fl_Group::init_sizes()

33.136.4.23 insert()

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.

33.136.4.24 is_interactive_resize()

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().

33.136.4.25 is_selected()

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.

33.136.4.26 move_cursor()

int Fl_Table::move_cursor( int R,
   int C,
   int shiftselect )
Moves the selection cursor a relative number of rows/columns
specifed 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.
33.136.4.27 **recalc_dimensions()**

```cpp
void Fl_Table::recalc_dimensions ( ) [protected]
```
Recalculate the dimensions of the table, and affect any children. Internally, `Fl_Group::resize()` and `init_sizes()` are called.

33.136.4.28 **redraw_range()**

```cpp
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.

33.136.4.29 **resize()**

```cpp
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`.

33.136.4.30 **row_col_clamp()**

```cpp
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.

33.136.4.31 **row_header()**

```cpp
void Fl_Table::row_header ( 
    int flag ) [inline]
```
Enables/disables showing the row headers. 1=enabled, 0=disabled. If changed, the table is redrawn.

33.136.4.32 **row_height()**

```cpp
void Fl_Table::row_height ( 
    int row, 
    int height)
```
Sets the height of the specified row in pixels, and the table is redrawn. `callback()` will be invoked with CONTEXT_RC_RESIZE if the row's height was actually changed, and `when()` is FL_WHEN_CHANGED.
33.136.4.33  row_height_all()

void Fl_Table::row_height_all (  
    int height ) [inline]

Convenience method to set the height of all rows to the same value, in pixels.  
The screen is redrawn.

33.136.4.34  row_resize()

void Fl_Table::row_resize (  
    int flag ) [inline]

Allows/disallows row resizing by the user.  
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the row headers,  
row_header() must also be enabled to allow resizing.

33.136.4.35  row_resize_min()

void Fl_Table::row_resize_min (  
    int val ) [inline]

Sets the current row minimum resize value.  
This is used to prevent the user from interactively resizing any row to be smaller than 'pixels'. Must be a value >=1.

33.136.4.36  rows()

void Fl_Table::rows (  
    int val ) [virtual]

Sets the number of rows in the table, and the table is redrawn.  
Reimplemented in Fl_Table_Row.

33.136.4.37  scrollbar_size() [1/2]

int Fl_Table::scrollbar_size ( ) const [inline]

Gets the current size of the scrollbars' troughs, in pixels.  
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.  
Returns  

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also  

Fl::scrollbar_size(int)

33.136.4.38  scrollbar_size() [2/2]

void Fl_Table::scrollbar_size (  
    int newSize ) [inline]

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.  
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL  
your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally  
what you want.  
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.  
Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the  
default.
Parameters

| in  | newSize | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size() |

See also

Fl::scrollbar_size()

33.136.4.39 set_selection()

void Fl_Table::set_selection (  
  int row_top,  
  int col_left,  
  int row_bot,  
  int col_right  
)

Sets the region of cells to be selected (highlighted).
So for instance, set_selection(0,0,0,0) selects the top/left cell in the table. And set_selection(0,0,1,1) selects the four cells in rows 0 and 1, column 0 and 1.
To deselect all cells, use set_selection(-1,-1,-1,-1);

Parameters

| in | row_top | Top row of selection area |
| in | col_left | Left column of selection area |
| in | row_bot | Bottom row of selection area |
| in | col_right | Right column of selection area |

33.136.4.40 tab_cell_nav() [1/2]

int Fl_Table::tab_cell_nav ( ) const [inline]
Get state of table's 'Tab' key cell navigation flag.

Returns
1 if Tab configured to navigate cells in table
0 to navigate widget focus (default)

See also

tab_cell_nav(int)

33.136.4.41 tab_cell_nav() [2/2]

void Fl_Table::tab_cell_nav (  
  int val  ) [inline]
Flag to control if Tab navigates table cells or not.
If on, Tab key navigates table cells. If off, Tab key navigates fltk widget focus. (default)
As of fltk 1.3, the default behavior of the Tab key is to navigate focus off of the current widget, and on to the next one. But in some applications, it's useful for Tab to be used to navigate cells in the Fl_Table.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 33.136.42 table_box()

```cpp
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.

#### 33.136.43 table_resized()

```cpp
void Fl_Table::table_resized () [protected]
```

Call this if table was resized, to recalculate internal data. Calls recall_dimensions(), and recalculates scrollbar sizes.

#### 33.136.44 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[xywh] has already been recalculated.

#### 33.136.45 top_row() [1/2]

```cpp
int Fl_Table::top_row () [inline]
```

Returns the current top row shown in the table. This row may be partially obscured.

#### 33.136.46 top_row() [2/2]

```cpp
void Fl_Table::top_row (int row) [inline]
```

Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn. If the table cannot be scrolled that far, it is scrolled as far as possible.

#### 33.136.47 visible_cells()

```cpp
void Fl_Table::visible_cells (int & r1, int & r2, int & c1, int & c2) [inline]
```

Returns the range of row and column numbers for all visible and partially visible cells in the table. These values can be used e.g. by your `draw_cell()` routine during CONTEXT_STARTPAGE to figure out what cells are about to be redrawn for the purposes of locking the data from a database before it's drawn.

```
<p>| V I S I B L E |</p>
<table>
<thead>
<tr>
<th>T A B L E</th>
</tr>
</thead>
</table>
```

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`
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>

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

33.137 Fl_Table_Row Class Reference

A table with row selection capabilities.

#include <Fl_Table_Row.H>

Inheritance diagram for Fl_Table_Row:

```
Fl_Widget
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Group</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Table</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Fl_Table_Row</td>
</tr>
</tbody>
</table>
```

Public Types

- enum TableRowSelectMode { SELECT_NONE, SELECT_SINGLE, SELECT_MULTI }

Public Member Functions

- void clear () FL_OVERRIDE
  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.
- Fl_Table_Row (int X, int Y, int W, int H, const char *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) FL_OVERRIDE`
  
  Sets the number of rows in the table, and the table is redrawn.
- `void select_all_rows (int flag=1)`
  
  This convenience function changes the selection state for all rows based on 'flag'.
- `int select_row (int row, int flag=1)`
  
  Changes the selection state for 'row', depending on the value of 'flag'.
- `TableRowSelectMode type () const`
- `void type (TableRowSelectMode val)`
  
  Sets the table selection mode.
- `~Fl_Table_Row ()`
  
  The destructor for the Fl_Table_Row.

Protected Member Functions

- `int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)`
- `int handle (int event) FL_OVERRIDE`
  
  Handle FLTK events.

Additional Inherited Members

33.137.1 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().

33.137.2 Constructor & Destructor Documentation

33.137.2.1 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.

33.137.2.2 ~Fl_Table_Row()

Fl_Table_Row::~Fl_Table_Row () [inline]

The destructor for the Fl_Table_Row.

Destroys the table and its associated widgets.

33.137.3 Member Function Documentation
33.137.3.1 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.

33.137.3.2 handle()

int Fl_Table_Row::handle (  
    int e ) [protected], [virtual]
Handle FLTK events.
Reimplemented from Fl_Table.

33.137.3.3 row_selected()

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)

33.137.3.4 rows()

void Fl_Table_Row::rows (  
    int val ) [virtual]
Sets the number of rows in the table, and the table is redrawn.
Reimplemented from Fl_Table.

33.137.3.5 select_all_rows()

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.

33.137.3.6 select_row()

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.

33.137.3.7 type()

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

33.138 Fl_Tabs Class Reference

The Fl_Tabs widget is a container widget that displays a set of tabs, with each tab representing a different child widget.

```cpp
#include <Fl_Tabs.H>
```

Inheritance diagram for Fl_Tabs:

```
Fl_Widget
 ↓
Fl_Group
 ↓
Fl_Tabs
```

Public Types

• enum { OVERFLOW_COMPRESS = 0, OVERFLOW_CLIP, OVERFLOW_PULLDOWN, OVERFLOW_DRAG }

Public Member Functions

• void client_area (int &rx, int &ry, int &rw, int &rh, int tabh=0)
  Returns the position and size available to be used by its children.
• Fl_Tabs (int X, int Y, int W, int H, const char ∗L=0)
  Creates a new Fl_Tabs widget using the given position, size, and label string.
• int handle (int) FL_OVERRIDE
  Handles the specified event.
• void handle_overflow (int ov)
  Set a method to handle an overflowing tab bar.
• Fl_Widget ∗ push () const
  Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.
  • int push (Fl_Widget ∗)
    This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.
• Fl_Align tab_align () const
  Gets the tab label alignment.
• void tab_align (Fl_Align a)
  Sets the tab label alignment.
• Fl_Widget ∗ value ()
  Gets the currently visible widget/tab.
• int value (Fl_Widget ∗)
  Sets the widget to become the current visible widget/tab.
• virtual Fl_Widget ∗ which (int event_x, int event_y)
  Return a pointer to the child widget with a tab at the given coordinates.
**Protected Member Functions**

- void **check_overflow_menu** ()
- virtual void **clear_tab_positions** ()
- void **draw** () FL_OVERRIDE
  
  Draws the widget.
- void **draw_overflow_menu_button** ()
- virtual void **draw_tab** (int x1, int x2, int W, int H, Fl_Widget ∗o, int flags, int sel)
- void **handle_overflow_menu** ()
- virtual int **hit_close** (Fl_Widget ∗o, int event_x, int event_y)
  
  Check whether the coordinates fall within the "close" button area of the tab.
- int **on_insert** (Fl_Widget ∗, int) FL_OVERRIDE
  
  Make sure that we redraw all tabs when new children are added.
- int **on_move** (int, int) FL_OVERRIDE
  
  Make sure that we redraw all tabs when children are moved.
- void **on_remove** (int) FL_OVERRIDE
  
  Make sure that we redraw all tabs when new children are removed.
- virtual void **redraw_tabs** ()
- void **resize** (int, int, int, int) FL_OVERRIDE
  
  Make sure that we redraw all tabs when the widget size changes.
- virtual int **tab_height** ()
- virtual int **tab_positions** ()

**Protected Attributes**

- int **has_overflow_menu**
- Fl_Menu_Item ∗ **overflow_menu**
- int **overflow_type**
- Fl_Align **tab_align_**
- int **tab_count**
- int ∗ **tab_flags**
- int **tab_offset**
- int ∗ **tab_pos**
- int ∗ **tab_width**

**Additional Inherited Members**

### 33.138.1 Detailed Description

The Fl_Tabs widget is a container widget that displays a set of tabs, with each tab representing a different child widget.

The user can select a tab by clicking on it, and the corresponding child widget will be displayed. The Fl_Tabs widget is useful for organizing a large number of controls or other widgets into a compact space, allowing the user to switch between different sets of controls as needed.
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**
The selected "tab" can be highlighted further by setting the `selection_color()` of the Fl_Tab itself, e.g.

```c
..
tabs = new Fl_Tabs(..);
tabs->selection_color(FL_DARK3);
..
```

The result of the above looks like:

![Figure 33.48 Highlighting the selected tab](image)

**Uniform Tab and Panel Appearance**

In order to have uniform tab and panel appearance, not only must the `color()` and `selection_color()` for each child group be set, but also the `selection_color()` of the Fl_Tab itself any time a new "tab" is selected. This can be achieved within the Fl_Tab callback, e.g.

```c
void MyTabCallback(Fl_Widget *w, void*) {
    Fl_Tabs *tabs = (Fl_Tabs*)w;
    // When tab changed, make sure it has same color as its group
    tabs->selection_color( (tabs->value())->color() );
}
..
int main(..) {
    // Define tabs widget
    tabs = new Fl_Tabs(..);
    tabs->callback(MyTabCallback);
    // Create three tabs each colored differently
    grp1 = new Fl_Group(.. "One");
    grp1->color(9);
    grp1->selection_color(9);
    grp1->end();
    grp2 = new Fl_Group(.. "Two");
    grp2->color(10);
    grp2->selection_color(10);
    grp2->end();
    grp3 = new Fl_Group(.. "Three");
    grp3->color(14);
    grp3->selection_color(14);
    grp3->end();
    ..
    // Make sure default tab has same color as its group
    tabs->selection_color( (tab->value())->color() );
    ..
    return Fl::run();
}
```

The result of the above looks like:

![Figure 33.49 Fl_Tabs with uniform colors](image)

**Close Button on Tabs**

The Fl_Tabs widget allows you to specify that a child widget should display a close button in its tab. If the `FL_WHEN_CLOSED` flag is set for the child widget, an "X" symbol will be displayed to the left of the label text in the tab. When the close button is clicked, the child widget's callback function will be called with the `FL_REASON_CLOSED` argument. It is then the responsibility of the child widget to remove itself from the Fl_Tabs container.

Generated by Doxygen
Tabs that are in a compressed state will not display a close button until they are fully expanded.

**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.
7. `Fl::callback_reason()` returns `FL_REASON_SELECTED` or `FL_REASON_RESELECTED`

**33.138.2 Constructor & Destructor Documentation**

**33.138.2.1 Fl_Tabs()**

```cpp
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.

### 33.138.3 Member Function Documentation

#### 33.138.3.1 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><code>tabh</code></th>
<th>position and optional height of tabs (see above)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>rx,ry,rw,rh</code></td>
<td>(x,y,w,h) of client area for children</td>
</tr>
</tbody>
</table>

Since

FLTK 1.3.0

#### 33.138.3.2 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`.

Generated by Doxygen
33.138.3.3 handle()

```cpp
int Fl_Tabs::handle(int event) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-
circuit events that you don't handle. In this last case you should return the callee retval.

**Parameters**

<table>
<thead>
<tr>
<th>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_Group.

33.138.3.4 handle_overflow()

```cpp
void Fl_Tabs::handle_overflow(int ov)
```

Set a method to handle an overflowing tab bar.
The Fl_Tabs widget allows you to specify how to handle the situation where there are more tabs than can be
displayed at once. The available options are:

- OVERFLOW_COMPRESS: Tabs will be compressed and overlaid on top of each other.
- OVERFLOW_CLIP: Only the first tabs that fit will be displayed.
- OVERFLOW_PULLDOWN: Tabs that do not fit will be placed in a pull-down menu.
- OVERFLOW_DRAG: The tab bar can be dragged horizontally to reveal additional tabs.

You can set the desired behavior using the overflow() method.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ov</td>
<td>overflow type</td>
</tr>
</tbody>
</table>

33.138.3.5 hit_close()

```cpp
int Fl_Tabs::hit_close(Fl_Widget * o, int event_x, int event_y) [protected], [virtual]
```

Check whether the coordinates fall within the "close" button area of the tab.
The Fl_Tabs::hit_close() method checks whether the given event coordinates fall within the area of the "close"
button on the tab of the specified child widget. This method should be called after the Fl_Tabs::which() method,
which updates a lookup table used to determine the width of each tab.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>o</code></td>
<td>check the tab of this widget</td>
</tr>
<tr>
<td><code>event_x, event_y</code></td>
<td>event coordinates</td>
</tr>
</tbody>
</table>

### Returns

- 1 if we hit the close button, and 0 otherwise

#### 33.138.3.6 on_insert()

```cpp
int Fl_Tabs::on_insert (Fl_Widget * candidate, int index) [protected], [virtual]
```

Make sure that we redraw all tabs when new children are added.
Reimplemented from `Fl_Group`.

#### 33.138.3.7 on_move()

```cpp
int Fl_Tabs::on_move (int a, int b) [protected], [virtual]
```

Make sure that we redraw all tabs when children are moved.
Reimplemented from `Fl_Group`.

#### 33.138.3.8 on_remove()

```cpp
void Fl_Tabs::on_remove (int index) [protected], [virtual]
```

Make sure that we redraw all tabs when new children are removed.
Reimplemented from `Fl_Group`.

#### 33.138.3.9 push() [1/2]

```cpp
Fl_Widget * Fl_Tabs::push () const [inline]
```

Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.
Otherwise, returns NULL.
While the user is down-clicked on a tab, the return value is the tab group for that tab. But as soon as the user releases, or drags off the tab with the button still down, the return value will be NULL.
See also `push(Fl_Widget*)`.

#### 33.138.3.10 push() [2/2]

```cpp
int Fl_Tabs::push (Fl_Widget * o)
```

This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.
Set back to zero on FL_RELEASE.
As of this writing, the value is mainly used by draw_tab() to determine whether or not to draw a 'down' box for the tab when it's clicked, and to turn it off if the user drags off it.
See also

push().

33.138.3.11 resize()

```cpp
def resize (int X, int Y, int W, int H) [protected], [virtual]
```

Make sure that we redraw all tabs when the widget size changes.
Reimplemented from Fl_Group.

33.138.3.12 tab_align() [1/2]

```cpp
def tab_align () const [inline]
```

Gets the tab label alignment.
See also

```cpp
tab_align(Fl_Align)
```

33.138.3.13 tab_align() [2/2]

```cpp
def 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_LEFT
IMAGE_NEXT_TO_TEXT is the recommended alignment to show the icon left of the text.

33.138.3.14 value() [1/2]

```cpp
def value ()
```

Gets the currently visible widget/tab.
The Fl_Tabs::value() method returns a pointer to the currently visible child widget of the Fl_Tabs container. The
visible child is the first child that is currently being displayed, or the last child if none of the children are being
displayed.
If child widgets have been added, moved, or deleted, this method ensures that only one tab is visible at a time.

Returns

a pointer to the currently visible child

33.138.3.15 value() [2/2]

```cpp
def value (Fl_Widget * newvalue)
```

Sets the widget to become the current visible widget/tab.
The Fl_Tabs::value() method allows you to set a particular child widget of the Fl_Tabs container to be the currently
visible widget. If the specified widget is a child of the Fl_Tabs container, it will be made visible and all other children
will be hidden. The method returns 1 if the value was changed, and 0 if the specified value was already set.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newvalue</td>
<td>a pointer to a child widget</td>
</tr>
</tbody>
</table>

Returns

- 1 if a different tab was chosen
- 0 if there was no change (new value already set)

### 33.138.3.16 which()

```cpp
Fl_Widget * Fl_Tabs::which (int event_x, int event_y) [virtual]
```

Return a pointer to the child widget with a tab at the given coordinates.

The `Fl_Tabs::which()` method returns a pointer to the child widget of the `Fl_Tabs` container that corresponds to the tab at the given event coordinates. If the event coordinates are outside the area of the tabs or if the `Fl_Tabs` container has no children, the method returns NULL.

Parameters

- event_x, event_y - event coordinates

Returns

- pointer to the selected child widget, or NULL

The documentation for this class was generated from the following files:

- Fl_Tabs.H
- Fl_Tabs.cxx

### 33.139 Fl_Text_Buffer Class Reference

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

```cpp
#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, int addedLength=-1)
  
  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.

• bool **can_redo** () const
  
  Check if undo is enabled and if the last undo action can be redone.

• bool **can_undo** () const
  
  Check if undo is enabled and if the last action can be undone.

• void **canUndo** (char flag=1)
  
  Enable or disable undo actions for this text buffer.

• unsigned int **char_at** (int pos) const
  
  Returns the character at the specified position pos in the buffer.

• void **copy** (Fl_Text_Buffer *fromBuf, int fromStart, int fromEnd, int toPos)
  
  Copies text from another Fl_Text_Buffer to this one.

• int **count_displayed_characters** (int lineStartPos, int targetPos) const
  
  Count the number of displayed characters between buffer position lineStartPos and targetPos.

• int **count_lines** (int startPos, int endPos) const
  
  Counts the number of newlines between startPos and endPos in buffer.

• int **findchar_backward** (int startPos, unsigned int searchChar, int *foundPos) const
  
  Search backwards in buffer buf for character searchChar, starting with the character before startPos, returning the result in foundPos.

• int **findchar_forward** (int startPos, unsigned searchChar, int *foundPos) const
  
  Finds the next occurrence of the specified character.

• Fl_Text_Buffer (int requestedSize=0, int preferredGapSize=1024)
  
  Create an empty text buffer of a pre-determined size.

• int **highlight** ()
  
  Returns a non-zero value if text has been highlighted, 0 otherwise.

• void **highlight** (int start, int end)
  
  Highlights the specified text within the buffer.

• int **highlight_position** (int *start, int *end)
  
  Highlights the specified text between start and end within the buffer.

• const Fl_Text_Selection * **highlight_selection** () const
  
  Returns the current highlight selection.

• char * **highlight_text** ()
  
  Returns the highlighted text.

• void **insert** (int pos, const char * text, int insertedLength=-1)
  
  Inserts null-terminated string text at position pos.

• int **insertfile** (const char * file, int pos, int buflen=128 *1024)
  
  Inserts a file at the specified position.

• bool **is_word_separator** (int pos) const
  
  Returns whether character at position pos is a word separator.

• int **length** () const
  
  Returns the number of bytes in the buffer.

• int **line_end** (int pos) const
  
  Finds and returns the position of the end of the line containing position pos (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).

• int **line_start** (int pos) const
  
  Returns the position of the start of the line containing position pos.

• char * **line_text** (int pos) const
  
  Returns the text from the entire line containing the specified character position.
- **int loadfile** (const char *file, int buflen=128 *1024)
  Loads a text file into the buffer.

- **int next_char** (int ix) const
  Returns the index of the next character.

- **int next_char_clipped** (int ix) const

- **int outputfile** (const char *file, int start, int end, int buflen=128 *1024)
  Writes the specified portions of the text buffer to a file.

- **int prev_char** (int ix) const
  Returns the index of the previous character.

- **int prev_char_clipped** (int ix) const

- **const Fl_Text_Selection * primary_selection ()**
  Returns the primary selection.

- **const Fl_Text_Selection * primary_selection () const**
  Returns the primary selection.

- **void printf** (const char *fmt,...)
  Appends printf formatted messages to the end of the buffer.

- **int redo** (int *cp=0)
  Redo previous undo action.

- **void remove** (int start, int end)
  Deletes a range of characters in the buffer.

- **void remove_modify_callback** (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)
  Removes a modify callback.

- **void remove_predelete_callback** (Fl_Text_Predelete_Cb predelCB, void *cbArg)
  Removes a callback routine bufPreDeleteCB associated with argument cbArg to be called before text is deleted from the buffer.

- **void remove_secondary_selection ()**
  Removes the text from the buffer corresponding to the secondary text selection object.

- **void remove_selection ()**
  Removes the text in the primary selection.

- **void replace** (int start, int end, const char *text, int insertedLength=-1)
  Deletes the characters between start and end, and inserts the null-terminated string text in their place in the buffer.

- **void replace_secondary_selection** (const char *text)
  Replaces the text from the buffer corresponding to the secondary text selection object with the new string text.

- **void replace_selection** (const char *text)
  Replaces the text in the primary selection.

- **int rewind_lines** (int startPos, int nLines)
  Finds and returns the position of the first character of the line nLines backwards from startPos (not counting the character pointed to by startPos if that is a newline) in the buffer.

- **int savelfile** (const char *file, int buflen=128 *1024)
  Saves a text file from the current buffer.

- **int search_backward** (int startPos, const char *searchString, int *foundPos, int matchCase=0) const
  Search backwards in buffer for string searchString, starting with the character at startPos, returning the result in foundPos.

- **int search_forward** (int startPos, const char *searchString, int *foundPos, int matchCase=0) const
  Search forwards in buffer for string searchString, starting with the character startPos, and returning the result in foundPos.

- **void secondary_select** (int start, int end)
  Selects a range of characters in the secondary selection.

- **int secondary_selected ()**
  Returns a non-zero value if text has been selected in the secondary text selection, 0 otherwise.

- **const Fl_Text_Selection * secondary_selection () const**
Returns the secondary selection.

- int **secondary_selection_position**(int *start, int *end)
  Returns the current selection in the secondary text selection object.

- char **secondary_selection_text**()
  Returns the text in the secondary selection.

- void **secondary_unselect**()
  Clears any selection in the secondary text selection object.

- void **select**(int start, int end)
  Selects a range of characters in the buffer.

- int **selected**( ) const
  Returns a non-zero value if text has been selected, 0 otherwise.

- int **selection_position**(int *start, int *end)
  Gets the selection position.

- char **selection_text**()
  Returns the currently selected text.

- int **skip_displayed_characters**(int lineStartPos, int nChars)
  Count forward from buffer position startPos in displayed characters.

- int **skip_lines**(int startPos, int nLines)
  Finds the first character of the line nLines forward from startPos in the buffer and returns its position.

- int **tab_distance**( ) const
  Gets the tab width.

- void **tab_distance**(int tabDist)
  Set the hardware tab distance (width) used by all displays for this buffer, and used in computing offsets for rectangular selection operations.

- char **text**( ) const
  Get a copy of the entire contents of the text buffer.

- void **text**(const char *text)
  Replaces the entire contents of the text buffer.

- char **text_range**(int start, int end) const
  Get a copy of a part of the text buffer.

- int **undo**(int *cp=0)
  Undo text modification according to the undo variables or insert text from the undo buffer.

- void **unhighlight**()
  Unhighlights text in the buffer.

- void **unselect**()
  Cancels any previous selection on the primary text selection object.

- int **utf8_align**(int) const
  Align an index into the buffer to the current or previous UTF-8 boundary.

- void **vprintf**(const char *fmt, va_list ap)
  Can be used by subclasses that need their own printf() style functionality.

- int **word_end**(int pos) const
  Returns the position corresponding to the end of the word.

- int **word_start**(int pos) const
  Returns the position corresponding to the start of the word.

- ~**Fl_Text_Buffer**( )
  Frees a text buffer.

Public Attributes

- int **input_file_was_transcoded**
  true if the loaded file has been transcoded to UTF-8.

- void(* **transcoding_warning_action**(Fl_Text_Buffer *))
  Pointer to a function called after reading a non UTF-8 encoded file.
Static Public Attributes

- static const char * file_encoding_warning_message
  This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

Protected Member Functions

- int apply_undo (Fl_Text_Undo_Action *action, int *cursorPos)
  Apply the current undo/redo operation, called from undo() or redo().

- void call_modify_callbacks (int pos, int nDeleted, int nInserted, int nRestyled, const char *deletedText) const
  Calls the stored modify callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

- void call_predelete_callbacks (int pos, int nDeleted) const
  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.

- int insert_ (int pos, const char *text, int insertedLength=-1)
  Internal (non-redisplaying) version of insert().

- void move_gap (int pos)
  Move the gap to start at a new position.

- void reallocate_with_gap (int newGapStart, int newGapLen)
  Reallocates the text storage in the buffer to have a gap starting at newGapStart and a gap size of newGapLen, preserving the buffer's current contents.

- void redisplay_selection (Fl_Text_Selection *oldSelection, Fl_Text_Selection *newSelection) const
  Calls the stored redisplay procedure(s) for this buffer to update the screen for a change in a selection.

- void remove_ (int start, int end)
  Internal (non-redisplaying) version of remove().

- void remove_selection_ (Fl_Text_Selection *sel)
  Removes the text from the buffer corresponding to sel.

- void replace_selection_ (Fl_Text_Selection *sel, const char *text)
  Replaces the text in selection sel.

- char * selection_text_ (Fl_Text_Selection *sel) const

- void update_selections (int pos, int nDeleted, int nInserted)
  Updates all of the selections in the buffer for changes in the buffer's text.

Protected Attributes

- char * mBuf
  allocated memory where the text is stored

- char mCanUndo
  if this buffer is used for attributes, it must not do any undo calls

- void ** mCbArgs
  caller arguments for modifyProcs above

- int mCursorPosHint
  hint for reasonable cursor position after a buffer modification operation

- int mGapEnd
  points to the first character after the gap

- int mGapStart
  points to the first character of the gap

- Fl_Text_Selection mHighlight
  highlighted areas

- int mLength
  length of the text in the buffer (the length of the buffer itself must be calculated: gapEnd - gapStart + length)
33.139.1 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/.

33.139.2 Constructor & Destructor Documentation

33.139.2.1 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

| requestedSize | use this to avoid unnecessary re-allocation if you know exactly how much the buffer will need to hold |
| preferredGapSize | Initial size for the buffer gap (empty space in the buffer where text might be inserted if the user is typing sequential characters) |
33.139.3 Member Function Documentation

33.139.3.1 add_modify_callback()

```cpp
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:

```cpp
typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,  
  int nRestyled, const char* deletedText,  
  void* cbArg);
```

33.139.3.2 address() [1/2]

```cpp
char * Fl_Text_Buffer::address (  
  int pos) [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address

33.139.3.3 address() [2/2]

```cpp
const char * Fl_Text_Buffer::address (  
  int pos) const [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

| pos | byte offset into buffer |

Returns

byte offset converted to a memory address

33.139.3.4 append()

```cpp
void Fl_Text_Buffer::append (  
  const char * t,  
  int addedLength = -1 ) [inline]
```

Appends the text string to the end of the buffer.

Parameters

| t | UTF-8 encoded text |
| addedLength | number of bytes to append, or -1 to indicate t is null-terminated |
### 33.139.3.5 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()`.

### 33.139.3.6 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**

<table>
<thead>
<tr>
<th>pos</th>
<th>byte offset into buffer</th>
</tr>
</thead>
</table>

**Returns**

unencoded raw byte

### 33.139.3.7 can_redo()

```cpp
bool Fl_Text_Buffer::can_redo ( ) const
```

Check if undo is enabled and if the last undo action can be redone. See also `can_undo()`

### 33.139.3.8 can_undo()

```cpp
bool Fl_Text_Buffer::can_undo ( ) const
```

Check if undo is enabled and if the last action can be undone. See also `can_undo()`

### 33.139.3.9 canUndo()

```cpp
void Fl_Text_Buffer::canUndo ( char flag = 1 )
```

Enable or disable undo actions for this text buffer. Undo actions are enabled for text buffer by default. If used as a style buffer in `Fl_Text_Display`, undo actions are disabled as they are handled by the text buffer. See also `can_undo()`
### 33.139.3.10 `char_at()`

```cpp
def char_at(int pos) const:
    Returns the character at the specified position \texttt{pos} in the buffer.
    Positions start at 0.
```

**Parameters**

- \texttt{pos} \texttt{byte offset into buffer, pos must be at a UTF-8 character boundary}

**Returns**

Unicode UCS-4 encoded character

### 33.139.3.11 `copy()`

```cpp
def copy(Fl_Text_Buffer* fromBuf, int fromStart, int fromEnd, int toPos):
    Copies text from another Fl_Text_Buffer to this one.
```

**Parameters**

- \texttt{fromBuf} \texttt{source text buffer, may be the same as this}
- \texttt{fromStart} \texttt{byte offset into buffer}
- \texttt{fromEnd} \texttt{byte offset into buffer}
- \texttt{toPos} \texttt{destination byte offset into buffer}

### 33.139.3.12 `count_displayed_characters()`

```cpp
def count_displayed_characters(int lineStartPos, int targetPos) const:
    Count the number of displayed characters between buffer position \texttt{lineStartPos} and \texttt{targetPos}.
    Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.
```

### 33.139.3.13 `count_lines()`

```cpp
def count_lines(int startPos, int endPos) const:
    Counts the number of newlines between \texttt{startPos} and \texttt{endPos} in buffer.
    The character at position \texttt{endPos} is not counted.
```

### 33.139.3.14 `findchar_backwards()`

```cpp
def findchar_backwards(int startPos, unsigned int searchChar, int* foundPos) const:
```
Search backwards in buffer buf for character searchChar, starting with the character before startPos, returning the result in foundPos. Returns 1 if found, 0 if not. The difference between this and search_backward() is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPos</td>
<td>byte offset to start position</td>
</tr>
<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

33.139.3.15 findchar_forward()

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startPos</td>
<td>byte offset to start position</td>
</tr>
<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

33.139.3.16 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.

33.139.3.17 insert()

void Fl_Text_Buffer::insert (  
    int pos,  
    const char * text,  
    int insertedLength = -1 )

Inserts null-terminated string text at position pos.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>insertion position as byte offset (must be UTF-8 character aligned)</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>text</th>
<th>UTF-8 encoded text</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertedLength</td>
<td>number of bytes to insert, or -1 to indicate text is null-terminated</td>
</tr>
</tbody>
</table>

33.139.3.18 insert()

```cpp
int Fl_Text_Buffer::insert_ (  
    int pos,  
    const char ∗ text,  
    int insertedLength = -1 ) [protected]
```

Internal (non-redisplaying) version of insert().

Returns the length of text inserted (this is just strlen(text) if insertedLength == -1, however this calculation can be expensive and the length will be required by any caller who will continue on to call redisplay). pos must be contiguous with the existing text in the buffer (i.e. not past the end).

Returns

the number of bytes inserted

33.139.3.19 insertfile()

```cpp
int Fl_Text_Buffer::insertfile ( 
    const char ∗ file,  
    int pos,  
    int buflen = 128*1024 )
```

Inserts a file at the specified position.

Returns

- 0 on success  
- non-zero on error (strerror() contains reason)  
- 1 indicates open for read failed (no data loaded)  
- 2 indicates error occurred while reading data (data was partially loaded)

File can be UTF-8 or CP1252 encoded. If the input file is not UTF-8 encoded, the Fl_Text_Buffer widget will contain data transcoded to UTF-8. By default, the message Fl_Text_Buffer::file_encoding_warning_message will warn the user about this.

See also

input_file_was_transcoded and transcoding_warning_action.

33.139.3.20 is_word_separator()

```cpp
bool Fl_Text_Buffer::is_word_separator (  
    int pos ) const
```

Returns whether character at position pos is a word separator. Pos must be at a character boundary.

33.139.3.21 length()

```cpp
int Fl_Text_Buffer::length () const [inline]
```

Returns the number of bytes in the buffer.

Returns

size of text in bytes
### line_end()

```c
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

### line_start()

```c
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

### line_text()

```c
char * Fl_Text_Buffer::line_text (int pos) const
```

Returns the text from the entire line containing the specified character position. When you are done with the text, free it using the `free()` function.

**Parameters**

- `pos` byte index into buffer

**Returns**

copy of UTF-8 text, must be free'd

### 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()`.
33.139.3.26 next_char()

int Fl_Text_Buffer::next_char ( int ix ) const

Returns the index of the next character.

Parameters
ix index to the current character

33.139.3.27 outputfile()

int Fl_Text_Buffer::outputfile ( const char * file, int start, int end, int buflen = 128*1024 )

Writes the specified portions of the text buffer to a file.

Returns
- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

See also
savefile(const char +file, int buflen)

33.139.3.28 prev_char()

int Fl_Text_Buffer::prev_char ( int ix ) const

Returns the index of the previous character.

Parameters
ix index to the current character

33.139.3.29 printf()

void Fl_Text_Buffer::printf ( const char * fmt, ... )

Appends printf formatted messages to the end of the buffer.

Example:
#include <FL/Fl_Text_Display.H>
int main(...) { ... }
tbuf->printf("The current date is: %s\nThe time is: %s\n", date_str, time_str);

tbuf->printf("The current PID is \%ld\n", (long)getpid());

Note

The expanded string is currently limited to 1024 characters.

Parameters

| in   | fmt | a printf format string for the message text |

33.139.3.30 remove()

void Fl_Text_Buffer::remove (  
    int start,  
    int end )

Deletes a range of characters in the buffer.

Parameters

| start | byte offset to first character to be removed |
| end   | byte offset to character after last character to be removed |

33.139.3.31 remove_()

void Fl_Text_Buffer::remove_ (  
    int start,  
    int end ) [protected]

Internal (non-redisplaying) version of remove().
Removes the contents of the buffer between start and end (and moves the gap to the site of the delete).

33.139.3.32 replace()

void Fl_Text_Buffer::replace (  
    int start,  
    int end,  
    const char * text,  
    int insertedLength = -1 )

Deletes the characters between start and end, and inserts the null-terminated string text in their place in the buffer.

Parameters

| start | byte offset to first character to be removed and new insert position |
| end   | byte offset to character after last character to be removed |
| text  | UTF-8 encoded text |
| insertedLength | number of bytes to insert, or -1 to indicate text is null-terminated |

33.139.3.33 rewind_lines()

int Fl_Text_Buffer::rewind_lines (
int startPos,
int nLines )
Finds and returns the position of the first character of the line nLines backwards from startPos (not counting the character pointed to by startPos if that is a newline) in the buffer.
nLines == 0 means find the beginning of the line.

33.139.3.34 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 (strerror() contains reason)
• 1 indicates open for write failed (no data saved)
• 2 indicates error occurred while writing data (data was partially saved)

See also
 outputfile(const char ∗ file, int start, int end, int buflen)

33.139.3.35 search_backward()

int Fl_Text_Buffer::search_backward ( int startPos,
const char ∗ searchString,
int ∗ foundPos,
int matchCase = 0 ) const
Search backwards in buffer for string searchString, starting with the character at startPos, returning the result in foundPos.
Returns 1 if found, 0 if not.

Parameters

<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

33.139.3.36 search_forward()

int Fl_Text_Buffer::search_forward ( int startPos,
const char ∗ searchString,
int ∗ foundPos,
int matchCase = 0 ) const
Search forwards in buffer for string searchString, starting with the character startPos, and returning the result in foundPos.
Returns 1 if found, 0 if not.

Parameters

| startPos | byte offset to start position |
| searchString | UTF-8 string that we want to find |
| foundPos | byte offset where the string was found |
| matchCase | if set, match character case |

Returns

1 if found, 0 if not

33.139.3.37  secondary_selection_text()  

char * Fl_Text_Buffer::secondary_selection_text ( )  

Returns the text in the secondary selection.

When you are done with the text, free it using the free() function.

33.139.3.38  selection_text()  

char * Fl_Text_Buffer::selection_text ( )  

Returns the currently selected text.

When you are done with the text, free it using the free() function.

33.139.3.39  skip_displayed_characters()  

int Fl_Text_Buffer::skip_displayed_characters (  
    int lineStartPos,  
    int nChars )  

Count forward from buffer position startPos in displayed characters.

Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.

Parameters

| lineStartPos | byte offset into buffer |
| nChars | number of bytes that are sent to the display |

Returns

byte offset in input after all output bytes are sent

33.139.3.40  tab_distance()  

int Fl_Text_Buffer::tab_distance ( ) const [inline]  

Gets the tab width.

The tab width is measured in characters. The pixel position is calculated using an average character width.

33.139.3.41  text() [1/2]  

char * Fl_Text_Buffer::text ( ) const  

Get a copy of the entire contents of the text buffer.

Memory is allocated to contain the returned string, which the caller must free.
33.139 Fl_Text_Buffer Class Reference

Returns

newly allocated text buffer - must be free'd, text is UTF-8

33.139.3.42 text() [2/2]

```cpp
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.

33.139.3.43 text_range()

```cpp
char * Fl_Text_Buffer::text_range ( int start, int end ) const
```

Get a copy of a part of the text buffer. Return a copy of the text between start and end character positions from text buffer buf. Positions start at 0, and the range does not include the character pointed to by end. When you are done with the text, free it using the free() function.

Parameters

<table>
<thead>
<tr>
<th>start</th>
<th>byte offset to first character</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>byte offset after last character in range</td>
</tr>
</tbody>
</table>

Returns

newly allocated text buffer - must be free'd, text is UTF-8

33.139.3.44 undo()

```cpp
int Fl_Text_Buffer::undo ( int * cursorPos = 0 )
```

Undo text modification according to the undo variables or insert text from the undo buffer. Take the previous changes and undo them. Return the previous cursor position in cursorPos. Returns 1 if the undo was applied. CursorPos will be at a character boundary.

33.139.3.45 vprintf()

```cpp
void Fl_Text_Buffer::vprintf ( const char * fmt, va_list ap )
```

Can be used by subclasses that need their own printf() style functionality. E.g. Fl_Simple_Terminal::printf() would wrap around this method.

Note

The expanded string is currently limited to 1024 characters.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fmt</th>
<th>is a printf format string for the message text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ap</td>
<td>is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.</td>
</tr>
</tbody>
</table>

### 33.139.3.46 word_end()

```
int Fl_Text_Buffer::word_end (int pos) const
```

Returns the position corresponding to the end of the word.

**Parameters**

| pos | byte index into buffer |

**Returns**

byte offset to word end

### 33.139.3.47 word_start()

```
int Fl_Text_Buffer::word_start (int pos) const
```

Returns the position corresponding to the start of the word.

**Parameters**

| pos | byte index into buffer |

**Returns**

byte offset to word start

### 33.139.4 Member Data Documentation

#### 33.139.4.1 file_encoding_warning_message

```
const char * Fl_Text_Buffer::file_encoding_warning_message [static]
```

*Initial value:*

"Displayed text contains the UTF-8 transcoding
"of the input file which was not UTF-8 encoded.
"Some changes may have occurred."

This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

#### 33.139.4.2 mTabDist

```
int Fl_Text_Buffer::mTabDist [protected]
equiv.
number of characters in a tab
```
33.139.4.3 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 
\texttt{fl\_alert()} with the text of \texttt{file\_encoding\_warning\_message}. No warning message is displayed if this pointer is set to 
\texttt{NULL}. Use \texttt{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:

\begin{itemize}
\item Fl_Text_Buffer.H
\item Fl_Text_Buffer.cxx
\end{itemize}

33.140 Fl_Text_Display Class Reference

Rich text display widget.

\#include <Fl_Text_Display.H>

Inheritance diagram for Fl_Text_Display:

\begin{center}
\begin{tikzpicture}
\node[concept] (fl_widget) {Fl_Widget};
\node[concept, below of=fl_widget] (fl_group) {Fl_Group};
\node[concept, below of=fl_group] (fl_text_display) {Fl_Text_Display};
\node[concept, below of=fl_text_display] (fl_simple_terminal) {Fl_Simple_Terminal};
\node[concept, right of=fl_text_display] (fl_text_editor) {Fl_Text_Editor};
\end{tikzpicture}
\end{center}

Classes

\begin{itemize}
\item 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

\begin{itemize}
\item enum {

\begin{verbatim}
NORMAL_CURSOR , CARET_CURSOR , DIM_CURSOR , BLOCK_CURSOR ,
HEAVY_CURSOR , SIMPLE_CURSOR
\end{verbatim}

text display cursor shapes enumeration
\item enum {

\begin{verbatim}
CURSOR_POS , CHARACTER_POS
\end{verbatim}

the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two
consecutive characters.
\item enum {

\begin{verbatim}
DRAG_NONE = -2 , DRAG_START_DND = -1 , DRAG_CHAR = 0 , DRAG_WORD = 1 ,
DRAG_LINE = 2
\end{verbatim}

drag types - they match \texttt{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.
\item enum {

\begin{verbatim}
WRAP_NONE , WRAP_AT_COLUMN , WRAP_AT_PIXEL , WRAP_AT_BOUNDS
\end{verbatim}

wrap types - used in \texttt{wrap\_mode()}
\item enum {

\begin{verbatim}
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
\end{verbatim}

attribute flags in \texttt{Style Table Entry.attr}
\item typedef void(* Unfinished_Style_Cb) (int, void *)
\end{itemize}

Generated by Doxygen
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.

• int handle (int e) FL_OVERRIDE
  Event handling.

• void hide_cursor ()
  Hides the text cursor.

• void highlight_data (*Fl_Text_Buffer* styleBuffer, const *Style_Table_Entry* styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)
  Attach (or remove) highlight information in text display and redisplay.

• int in_selection (int x, int y) const
  Check if a pixel position is within the primary selection.

• void insert (const char *text)
  Inserts "text" at the current cursor location.

• int insert_position () const
  Gets the position of the text insertion cursor for text display.

• void insert_position (int newPos)
  Sets the position of the text insertion cursor for text display.

• int line_end (int startPos, bool startPosIsLineStart) const
  Returns the end of a line.

• int line_start (int pos) const
  Return the beginning of a line.

• *Fl_Align* linenumber_align () const
  Returns the alignment used for line numbers (if enabled).

• void linenumber_align (*Fl_Align* val)
  Set alignment for line numbers (if enabled).

• *Fl_Color* linenumberbgcolor () const
  Returns the background color used for line numbers (if enabled).

• void linenumberbgcolor (*Fl_Color* val)
  

Set the background color used for line numbers (if enabled).

- `FL_Color linenumberbgcolor() const`
  Return the foreground color used for line numbers (if enabled).

- `void linenumberbgcolor(FL_Color val)`
  Set the foreground color used for line numbers (if enabled).

- `FL_Font linenumberfont() const`
  Return the font used for line numbers (if enabled).

- `void linenumberfont(FL_Font val)`
  Set the font used for line numbers (if enabled).

- `const char* linenumberformat() const`
  Returns the line number printf() format string.

- `void linenumberformat(const char* val)`
  Sets the printf() style format string used for line numbers.

- `FL_Fontsize linenumbersize() const`
  Return the font size used for line numbers (if enabled).

- `void linenumbersize(FL_Fontsize val)`
  Set the font size used for line numbers (if enabled).

- `int linenumberwidth() const`
  Return the screen area width provided for line numbers.

- `void linenumberwidth(int width)`
  Set width of screen area for line numbers.

- `int move_down()`
  Moves the current insert position down one line.

- `int move_left()`
  Moves the current insert position left one character.

- `int move_right()`
  Moves the current insert position right one character.

- `int move_up()`
  Moves the current insert position up one line.

- `void next_word(void)`
  Moves the current insert position right one word.

- `void overstrike(const char* text)`
  Replaces text at the current insert position.

- `int position_style(int lineStartPos, int lineLen, int lineIndex) const`
  Find the correct style for a character.

- `int position_to_xy(int pos, int *x, int *y) const`
  Convert a character index into a pixel position.

- `void previous_word(void)`
  Moves the current insert position left one word.

- `virtual void recalc_display()`
  Recalculate the display’s visible lines and scrollbar sizes.

- `void redisplay_range(int start, int end)`
  Marks text from start to end as needing a redraw.

- `void resize(int X, int Y, int W, int H) FL_OVERRIDE`
  Change the size of the displayed text area.

- `int rewind_lines(int startPos, int nLines)`
  Skip a number of lines back.

- `void scroll(int topLineNum, int horizOffset)`
  Scrolls the current buffer to start at the specified line and column.

- `FL_Align scrollbar_align() const`
  Gets the scrollbar alignment type.
• void scrollbar_align (Fl_Align a)
  
  Sets the scrollbar alignment type.

• int scrollbar_size () const
  
  Gets the current size of the scrollbars’ troughs, in pixels.

• void scrollbar_size (int newSize)
  
  Sets the pixel size of the scrollbars’ troughs to newSize, in pixels.

• int scrollbar_width () const
  
  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

• void scrollbar_width (int width)
  
  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

• Fl_Color secondary_selection_color () const
  
  Gets the background color for the secondary selection block.

• void secondary_selection_color (Fl_Color color)
  
  Sets the background color for the secondary selection block.

• int shortcut () const

• void shortcut (int s)

• void show_cursor (int b=1)
  
  Shows the text cursor.

• void show_insert_position ()
  
  Scrolls the text buffer to show the current insert position.

• int skip_lines (int startPos, int nLines, bool startPosIsLineStart)
  
  Skip a number of lines forward.

• Fl_Color spelling_underline_color () const
  
  Gets the underline color for style attribute ATTR_SPELLING.

• void spelling_underline_color (Fl_Color color)
  
  Sets the underline color for style attribute ATTR_SPELLING.

• Fl_Text_Buffer ∗ style_buffer () const
  
  Gets the current style buffer associated with the text widget.

• Fl_Color textcolor () const

• void textcolor (Fl_Color n)
  
  Sets the default color of text in the widget.

• Fl_Font textfont () const

• void textfont (Fl_Font s)
  
  Sets the default font used when drawing text in the widget.

• Fl_Fontsize textsize () const

• 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.
- void draw () FL_OVERRIDE
  Draw the widget.
- void draw_cursor (int, int)
  Draw a cursor with top center at X, Y.
- void draw_line_numbers (bool clearAll)
  Refresh the line number area.
- void draw_range (int start, int end)
  Draw a range of text.
- void draw_string (int style, int x, int y, int toX, const char ∗string, int nChars) const
  Draw a text segment in a single style.
- void draw_text (int X, int Y , int W, int H)
  Refresh a rectangle of the text display.
- void draw_vline (int visLineNum, int leftClip, int rightClip, int leftCharIndex, int rightCharIndex)
  Draw a single line of text.
- int empty_vlines () const
  Return true if there are lines visible with no corresponding buffer text.
- void extend_range_for_styles (int ∗start, int ∗end)
  I don't know what this does!
- void find_line_end (int pos, bool start_pos_is_line_start, int ∗lineEnd, int ∗nextLineStart) const
  Finds both the end of the current line and the start of the next line.
- void find_wrap_range (const char ∗deletedText, int pos, int nInserted, int nDeleted, int ∗modRangeStart, int ∗modRangeEnd, int ∗linesInserted, int ∗linesDeleted)
  Wrapping calculations.
- int find_x (const char ∗s, int len, int style, int x) const
  Find the index of the character that lies at the given x position / closest cursor position.
- int get_absolute_top_line_number () const
  Returns the absolute (non-wrapped) line number of the first line displayed.
- int handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const
  Universal pixel machine.
- int longest_vline () const
  Find the longest line of all visible lines.
- void maintain_absolute_top_line_number (int state)
  Line numbering stuff, currently unused.
• int maintaining_absolute_top_line_number () const
  Returns true if a separate absolute top line number is being maintained.

• void measure_deleted_lines (int pos, int nDeleted)
  Wrapping calculations.

• double measure_proportional_character (const char *s, int colNum, int pos) const
  Wrapping calculations.

• int measure_vline (int visLineNum) const
  Returns the width in pixels of the displayed line pointed to by "visLineNum".

• void offset_line_starts (int newTopLineNum)
  Offset line start counters for a new vertical scroll position.

• int position_to_line (int pos, int *lineNum) const
  Convert a position index into a line number offset.

• int position_to_linecol (int pos, int *lineNum, int *column) const
  Find the line and column number of position pos.

• void reset_absolute_top_line_number ()
  Reestablish the absolute (non-wrapped) top line number.

• int scroll_ (int topLineNum, int horizOffset)
  Scrolls the current buffer to start at the specified line and column.

• double string_width (const char *string, int length, int style) const
  Find the width of a string in the font of a particular style.

• void update_h_scrollbar ()
  Update horizontal scrollbar.

• void update_line_starts (int pos, int charInserted, int charsDeleted, int linesInserted, int linesDeleted, int *scrolled)
  Update line start arrays and variables.

• void update_v_scrollbar ()
  Update vertical scrollbar.

• int vline_length (int visLineNum) const
  Count number of bytes in a visible line.

• int wrap_uses_character (int lineEndPos) const
  Check if the line break is caused by a newline or by line wrapping.

• void wrapped_line_counter (Fl_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIs←LineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLast←LineMissingNewLine=true) const
  Wrapping calculations.

• int xy_to_position (int x, int y, int PosType=CHARACTER_POS) const
  Translate a pixel position into a character index.

• void xy_to_rowcol (int x, int y, int *row, int *column, int PosType=CHARACTER_POS) const
  Translate pixel coordinates into row and column.

### Static Protected Member Functions

• static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText, void *cbArg)
  This is called whenever the buffer is modified.

• static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)
  This is called before any characters are deleted.

• static void h_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)
  Callback for drag or valueChanged on horizontal scrollbar.

• static void scroll_timer_cb (void *)
  Timer callback for scroll events.

• static void v_scrollbar_cb (Fl_Scrollbar *w, Fl_Text_Display *d)
  Callback for drag or valueChanged on vertical scrollbar.
Protected Attributes

- int damage_range1_end
- int damage_range1_start
- int damage_range2_end
- int damage_range2_start
- int display_insert_position_hint
- int dragging
- int dragPos
- int dragType
- Fl_Color grammar_underline_color_
- Fl_Align linenumber_align_
- Fl_Color linenumberbgcolor_
- Fl_Color linenumberfgcolor_
- Fl_Font linenumberfont_
- const char * linenumberformat_
- Fl_Fontsize linenumbersize_
- int mAbsTopLineNum
- Fl_Text_Buffer * mBuffer
- double mColumnScale
- int mContinuousWrap
- Fl_Color mCursor_color
- int mCursorOldY
- int mCursorOn
- int mCursorPos
- int mCursorPreferredXPos
- int mCursorStyle
- int mCursorToHint
- int mFirstChar
- void * mHighlightCBArg
- int mHorizOffset
- int mHorizOffsetHint
- Fl_Scrollbar * mHScrollBar
- int mLastChar
- int mLineNumLeft
- int mLineNumWidth
- int * mLineStarts
- int mMaxsize
- int mModifyingTabDistance
- int mNBufferLines
- int mNeedAbsTopLineNum
- int mNLinesDeleted
- int mNStyles
- int mNVisibleLines
- Fl_Text_Buffer * mStyleBuffer
- const Style_Table_Entry * mStyleTable
- int mSuppressResync
- int mTopLineNum
- int mTopLineNumHint
- Unfinished_Style_CB mUnfinishedHighlightCB
- char mUnfinishedStyle
- Fl_Scrollbar * mVScrollBar
- int mWrapMarginPix
- Fl_Align scrollbar_align_
- int scrollbar_width_
934 Class Documentation

\begin{itemize}
\item \texttt{Fl\_Color secondary\_selection\_color}\_
\item \texttt{int shortcut}\_
\item \texttt{Fl\_Color spelling\_underline\_color}\_
\end{itemize}

\begin{verbatim}
struct {
    int \texttt{h} \texttt{w} \texttt{x} \texttt{y}
} \texttt{text\_area}
\end{verbatim}

\begin{itemize}
\item \texttt{Fl\_Color textcolor}\_
\item \texttt{Fl\_Font textfont}\_
\item \texttt{Fl\_Fontsize textsize}\_
\end{itemize}

Friends

\begin{itemize}
\item \texttt{void fl\_text\_drag\_me (int pos, Fl\_Text\_Display \*d)}
\item \texttt{int fl\_text\_drag\_prepare (int pos, int key, Fl\_Text\_Display \*d)}
\end{itemize}

Additional Inherited Members

33.140.1 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 \texttt{Fl\_Text\_Buffer} class. A single Text Buffer can be displayed by multiple Text Displays.
Example Use

```c
#include <FL/Fl_Text_Display.H>
...
int main() {
    Fl_Text_Buffer *buff = new Fl_Text_Buffer();
    Fl_Text_Display *disp = new Fl_Text_Display(10, 10, 640, 480);
    disp->buffer(buff); // attach text buffer to display widget
    buff->text("line one\nline two"); // add some text to buffer
    ...
}
```

Features

- Word wrap: `wrap_mode()`, `wrapped_column()`, `wrapped_row()`
- Font control: `textfont()`, `textsize()`, `textcolor()`
- Font styling: `highlight_data()`
- Cursor: `cursor_style()`, `show_cursor()`, `hide_cursor()`, `cursor_color()`
- Line numbers: `linenumber_width()`, `linenumber_font()`, `linenumber_size()`, `linenumber_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)`

33.140.2 Member Enumeration Documentation

33.140.2.1 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>

#### 33.140.2.2 anonymous enum

anonymous enum
wrap types - used in wrap_mode()

<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>

#### 33.140.2.3 anonymous enum

anonymous enum
attribute flags in Style_Table_Entry.attr

<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>
<tr>
<td>ATTR_BGCOLOR_EXT</td>
<td>(internal use)</td>
</tr>
<tr>
<td>ATTR_BGCOLOR_EXT</td>
<td>extend background color to the end of the line</td>
</tr>
<tr>
<td>ATTR_UNDERLINE</td>
<td>a single underline, underline types are mutually exclusive</td>
</tr>
<tr>
<td>ATTR_GRAMMAR</td>
<td>grammar suggestion (blue dotted underline)</td>
</tr>
<tr>
<td>ATTR_SPELLING</td>
<td>spelling suggestion (red dotted underline)</td>
</tr>
<tr>
<td>ATTR_STRIKE_THROUGH</td>
<td>line through the middle of the text</td>
</tr>
<tr>
<td>ATTR_LINES_MASK</td>
<td>the mask for all underline and strike through types</td>
</tr>
</tbody>
</table>

### 33.140.3 Constructor & Destructor Documentation

#### 33.140.3.1 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

<table>
<thead>
<tr>
<th>X, Y, W, H</th>
<th>position and size of widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>label text, defaults to none</td>
</tr>
</tbody>
</table>

### 33.140.3.2 ~Fl_Text_Display()

```cpp
Fl_Text_Display::~Fl_Text_Display ( )
```

Free a text display and release its associated memory.

**Note**

The text buffer that the text display displays is a separate entity and is not freed, nor are the style buffer or style table.

**See also**

- `Fl_Text_Display::buffer(Fl_Text_Buffer* buf)`

### 33.140.4 Member Function Documentation

#### 33.140.4.1 absolute_top_line_number()

```cpp
void Fl_Text_Display::absolute_top_line_number ( int oldFirstChar ) [protected]
```

Re-calculate absolute top line number for a change in scroll position. Does nothing if the absolute top line number is not being maintained.

#### 33.140.4.2 buffer() [1/3]

```cpp
Fl_Text_Buffer * Fl_Text_Display::buffer ( ) const [inline]
```

Gets the current text buffer associated with the text widget. Multiple text widgets can be associated with the same text buffer.

**Returns**

- current text buffer

**See also**

- `Fl_Text_Display::buffer(Fl_Text_Buffer* buf)`
- `Fl_Text_Display::buffer(Fl_Text_Buffer& buf)`

#### 33.140.4.3 buffer() [2/3]

```cpp
void Fl_Text_Display::buffer ( Fl_Text_Buffer & buf ) [inline]
```

Sets the current text buffer associated with the text widget. Multiple text widgets can be associated with the same text buffer.

**Parameters**

- `buf` new text buffer
See also

\texttt{Fl_Text_Display::buffer(Fl_Text_Buffer \ast \text{buf})}

### 33.140.4.4 buffer()

```cpp
void Fl_Text_Display::buffer (Fl_Text_Buffer \ast \text{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

### 33.140.4.5 buffer_modified_cb()

```cpp
void Fl_Text_Display::buffer_modified_cb (int \text{pos},
int \text{nInserted},
int \text{nDeleted},
int \text{nRestyled},
const char \ast \text{deletedText},
void \ast \text{cbArg}) [\text{static}], [\text{protected}]
```

This is called whenever the buffer is modified. Callback attached to the text buffer to receive modification information. This callback can be used to adjust the display or update other setting. It is not advisable to change any buffers or text in this callback, or line counting may get out of sync.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pos</code></td>
<td>starting index of modification</td>
</tr>
<tr>
<td><code>nInserted</code></td>
<td>number of bytes inserted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td><code>nDeleted</code></td>
<td>number of bytes deleted (must be UTF-8 aligned!)</td>
</tr>
<tr>
<td><code>nRestyled</code></td>
<td>??</td>
</tr>
<tr>
<td><code>deletedText</code></td>
<td>this is what was removed, must not be NULL if nDeleted is set</td>
</tr>
<tr>
<td><code>cbArg</code></td>
<td>&quot;this&quot; pointer for static callback function</td>
</tr>
</tbody>
</table>

### 33.140.4.6 buffer_predelete_cb()

```cpp
void Fl_Text_Display::buffer_predelete_cb (int \text{pos},
int \text{nDeleted},
void \ast \text{cbArg}) [\text{static}], [\text{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.
33.140 Fl_Text_Display Class Reference

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>starting index of deletion</td>
</tr>
<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>

### 33.140.4.7 calc_last_char()

```cpp
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.

### 33.140.4.8 calc_line_starts()

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startLine,endLine</td>
<td>range of lines to scan as line numbers</td>
</tr>
</tbody>
</table>

### 33.140.4.9 clear_rect()

```cpp
void Fl_Text_Display::clear_rect ( int style, int X, int Y, int width, int height ) const [protected]
```

Clear a rectangle with the appropriate background color for `style`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>index into style table</td>
</tr>
<tr>
<td>X,Y,width,height</td>
<td>size and position of background area</td>
</tr>
</tbody>
</table>

### 33.140.4.10 col_to_x()

```cpp
double Fl_Text_Display::col_to_x ( double col ) const
```

Convert a column number into an x pixel position.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>an approximate column number based on the main font</td>
</tr>
</tbody>
</table>
Returns

number of pixels from the left margin to the left of an average sized character

See also

x_to_col()

33.140.4.11 count_lines()

int Fl_Text_Display::count_lines (int startPos, int endPos, bool startPosIsLineStart) const

Count the number of lines between two positions.
Same as Fl_Text_Buffer::count_lines(), but takes into account wrapping if wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to first character</th>
</tr>
</thead>
<tbody>
<tr>
<td>endPos</td>
<td>index after last character</td>
</tr>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

Returns

number of lines

33.140.4.12 cursor_color() [1/2]

Fl_Color Fl_Text_Display::cursor_color () const [inline]

Gets the text cursor color.

Returns

cursor color

33.140.4.13 cursor_color() [2/2]

void Fl_Text_Display::cursor_color (Fl_Color n) [inline]

Sets the text cursor color.

Parameters

| n | new cursor color |

33.140.4.14 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

<table>
<thead>
<tr>
<th>style</th>
<th>new cursor style</th>
</tr>
</thead>
</table>

### 33.140.4.15 display_insert()

```cpp
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.

### 33.140.4.16 draw()

```cpp
void Fl_Text_Display::draw ( ) [protected], [virtual]
```

Draw the widget.

This function tries to limit drawing to smaller areas if possible.

Reimplemented from `Fl_Group`.

### 33.140.4.17 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>

### 33.140.4.18 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>
### draw_range()

```cpp
def Fl_Text_Display::draw_range (int startpos, int endpos) [protected]
```

Draw a range of text. Refresh all of the text between buffer positions `startpos` and `endpos` not including the character at the position `endpos`. If `endpos` points beyond the end of the buffer, refresh the whole display after `startpos`, including blank lines which are not technically part of any range of characters.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startpos</code></td>
<td>index of first character to draw</td>
</tr>
<tr>
<td><code>endpos</code></td>
<td>index after last character to draw</td>
</tr>
</tbody>
</table>

### draw_string()

```cpp
def Fl_Text_Display::draw_string (int style, int X, int Y, int toX, const char * string, int nChars) const [protected]
```

Draw a text segment in a single style. Draw a string or blank area according to parameter `style`, using the appropriate colors and drawing method for that style, with top left corner at `X`, `Y`. If `style` says to draw text, use `string` as source of characters, and draw `nChars`. If `style` is FILL, erase rectangle where text would have drawn from `X` to `toX` and from `Y` to the maximum y extent of the current font(s).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>style</code></td>
<td>index into style lookup table</td>
</tr>
<tr>
<td><code>X,Y</code></td>
<td>drawing origin</td>
</tr>
<tr>
<td><code>toX</code></td>
<td>rightmost position if this is a fill operation</td>
</tr>
<tr>
<td><code>string</code></td>
<td>text if this is a drawing operation</td>
</tr>
<tr>
<td><code>nChars</code></td>
<td>number of characters to draw</td>
</tr>
</tbody>
</table>

### draw_text()

```cpp
def Fl_Text_Display::draw_text (int left, int top, int width, int height) [protected]
```

Refresh a rectangle of the text display.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>left,top</code></td>
<td>are in coordinates of the text drawing window.</td>
</tr>
<tr>
<td><code>width,height</code></td>
<td>size in pixels</td>
</tr>
</tbody>
</table>
### 33.140.4.22 draw_vline()

```cpp
void Fl_Text_Display::draw_vline (  
  int visLineNum,  
  int leftClip,  
  int rightClip,  
  int leftCharIndex,  
  int rightCharIndex ) [protected]
```

Draw a single line of text.

Draw the text on a single line represented by `visLineNum` (the number of lines down from the top of the display), limited by `leftClip` and `rightClip` window coordinates and `leftCharIndex` and `rightCharIndex` character positions (not including the character at position `rightCharIndex`).

**Parameters**

<table>
<thead>
<tr>
<th>visLineNum</th>
<th>index of line in the visible line number lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>leftClip, rightClip</td>
<td>pixel position of clipped area</td>
</tr>
<tr>
<td>leftCharIndex, rightCharIndex</td>
<td>index into line of segment that we want to draw</td>
</tr>
</tbody>
</table>

### 33.140.4.23 empty_vlines()

```cpp
int Fl_Text_Display::empty_vlines ( ) const [protected]
```

Return true if there are lines visible with no corresponding buffer text.

**Returns**

- 1 if there are empty lines

### 33.140.4.24 extend_range_for_styles()

```cpp
void Fl_Text_Display::extend_range_for_styles (  
  int * startPos,  
  int * endpos ) [protected]
```

I don't know what this does!

Extend the range of a redraw request (from `start` to `end`) with additional redraw requests resulting from changes to the attached style buffer (which contains auxiliary information for coloring or styling text).

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>??</th>
</tr>
</thead>
<tbody>
<tr>
<td>endpos</td>
<td>??</td>
</tr>
</tbody>
</table>

**Todo** Unicode?

### 33.140.4.25 find_line_end()

```cpp
void Fl_Text_Display::find_line_end (  
  int startPos,  
  bool startPosIsLineStart,  
  int * lineEnd,  
  int * nextLineStart ) const [protected]
```

Generated by Doxygen
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>

33.140.4.26 find_wrap_range()

```cpp
void Fl_Text_Display::find_wrap_range (const char * deletedText, int pos, int nInserted, int nDeleted, int * modRangeStart, int * modRangeEnd, int * linesInserted, int * linesDeleted ) [protected]
```

Wrapping calculations.
When continuous wrap is on, and the user inserts or deletes characters, wrapping can happen before and beyond the changed position. This routine finds the extent of the changes, and counts the deleted and inserted lines over that range. It also attempts to minimize the size of the range to what has to be counted and re-displayed, so the results can be useful both for delimiting where the line starts need to be recalculated, and for deciding what part of the text to redisplay.

Parameters

<table>
<thead>
<tr>
<th>deletedText</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
</tr>
<tr>
<td>nInserted</td>
</tr>
<tr>
<td>nDeleted</td>
</tr>
<tr>
<td>modRangeStart</td>
</tr>
<tr>
<td>modRangeEnd</td>
</tr>
<tr>
<td>linesInserted</td>
</tr>
<tr>
<td>linesDeleted</td>
</tr>
</tbody>
</table>

33.140.4.27 find_x()

```cpp
int Fl_Text_Display::find_x (const char * s, int len, int style, int x ) const [protected]
```

Find the index of the character that lies at the given x position / closest cursor position.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>UTF-8 text string</td>
</tr>
<tr>
<td>len</td>
<td>length of string</td>
</tr>
<tr>
<td>style</td>
<td>index into style lookup table</td>
</tr>
<tr>
<td>x</td>
<td>position in pixels - negative returns closest cursor position</td>
</tr>
</tbody>
</table>

Returns

index into buffer

### 33.140.4.28 get_absolute_top_line_number()

```cpp
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.

### 33.140.4.29 grammar_underline_color() [1/2]

```cpp
Fl_Color Fl_Text_Display::grammar_underline_color ( ) const [inline]
```

Gets the underline color for style attribute ATTR_GRAMMAR.

Returns

underline color

### 33.140.4.30 grammar_underline_color() [2/2]

```cpp
void Fl_Text_Display::grammar_underline_color ( Fl_Color color ) [inline]
```

Sets the underline color for style attribute ATTR_GRAMMAR.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>underline color</td>
</tr>
</tbody>
</table>

### 33.140.4.31 handle()

```cpp
int Fl_Text_Display::handle ( int e ) [virtual]
```

Event handling.
Reimplemented from Fl_Group.
Reimplemented in Fl_Text_Editor.

### 33.140.4.32 handle_vline()

```cpp
int Fl_Text_Display::handle_vline ( int mode,
int lineStartPos,
int lineLen,
int leftChar,
int rightChar,
```
int Y, 
int bottomClip, 
int leftClip, 
int rightClip ) const [protected]

Universal pixel machine.
We use a single function that handles all line layout, measuring, and drawing

• draw a text range
• return the width of a text range in pixels
• return the index of a character that is at a pixel position

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX</td>
<td></td>
</tr>
<tr>
<td>lineStartPos</td>
<td>index of first character</td>
<td></td>
</tr>
<tr>
<td>lineLen</td>
<td>size of string in bytes</td>
<td></td>
</tr>
<tr>
<td>leftChar, rightChar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>drawing position</td>
<td></td>
</tr>
<tr>
<td>bottomClip, leftClip, rightClip</td>
<td>stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX.</td>
<td></td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAW_LINE</td>
<td>index of last drawn character</td>
</tr>
<tr>
<td>GET_WIDTH</td>
<td>width in pixels of text segment if we would draw it</td>
</tr>
<tr>
<td>FIND_INDEX</td>
<td>index of character at given x position in window coordinates</td>
</tr>
<tr>
<td>FIND_INDEX_FROM_ZERO</td>
<td>index of character at given x position without scrolling and widget offsets</td>
</tr>
</tbody>
</table>

Todo we need to handle hidden hyphens and tabs here!

we handle all styles and selections
we must provide code to get pixel positions of the middle of a character as well

33.140.4.33 highlight_data()

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 extend_range_for_styles() for more information on this protocol).
Style buffers, tables and their associated memory are managed by the caller.
Styles are ranged from 65 ('A') to 126.
Note

Style information in the style buffer must have the same byte offset as the corresponding character in the text buffer. UTF-8 characters can have a maximum length of four bytes. Style information must take this into account and fill the unused bytes with 0. See \texttt{fl_utf8len()}.  

Text: "\*g\*r ü +n\*", where normal style is 'A', and bold is 'B'
Text Buffer\{hex\}: \texttt{67 72 c3 bc 6e} : gr..n
Style Buffer\{hex\}: \texttt{42 41 41 00 42} : BAA.B

Parameters

- \texttt{styleBuffer}\hspace{1em}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.
- \texttt{styleTable}\hspace{1em}a list of styles indexed by the style buffer
- \texttt{nStyles}\hspace{1em}number of styles in the style table
- \texttt{unfinishedStyle}\hspace{1em}if this style is found, the callback below is called
- \texttt{unfinishedHighlightCB}\hspace{1em}if a character with an unfinished style is found, this callback will be called
- \texttt{cbArg}\hspace{1em}an optional argument for the callback above, usually a pointer to the Text Display.

See also

- \texttt{Fl_Text_Display::style_buffer()}

33.140.4.34 in_selection()

\texttt{int Fl_Text_Display::in_selection (}
\hspace{1em}\texttt{int X,}
\hspace{1em}\texttt{int Y ) const}

Check if a pixel position is within the primary selection.

Parameters

\texttt{X,Y}\hspace{1em}pixel position to test

Returns

1 if position (X, Y) is inside of the primary Fl_Text_Selection

33.140.4.35 insert()

\texttt{void Fl_Text_Display::insert (}
\hspace{1em}\texttt{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

\texttt{text}\hspace{1em}new text in UTF-8 encoding.
33.140.4.36  insert_position() [1/2]

int Fl_Text_Display::insert_position ( ) const [inline]
Gets the position of the text insertion cursor for text display.
The insert position is the byte count (offset) from the beginning of the text buffer (starting with 0). Returns 0 (zero) if no buffer is associated to the text display. Returns buffer()->length() if the insert position is at the end of the buffer.

Returns

insert position index into text buffer

See also

insert_position(int)

33.140.4.37  insert_position() [2/2]

void Fl_Text_Display::insert_position ( int newPos )
Sets the position of the text insertion cursor for text display.
Moves the insertion cursor in front of the character at newPos. This function may trigger a redraw.

Parameters

| newPos | new caret position |

33.140.4.38  line_end()

int Fl_Text_Display::line_end ( int startPos, bool startPosIsLineStart ) const
Returns the end of a line.
Same as buffer()->line_end(startPos), but takes into account line breaks when wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.
Note that the definition of the end of a line is less clear when continuous wrap is on. With continuous wrap off, it's just a pointer to the newline that ends the line. When it's on, it's the character beyond the last displayable character on the line, where a whitespace character which has been "converted" to a newline for wrapping is not considered displayable. Also note that a line can be wrapped at a non-whitespace character if the line had no whitespace. In this case, this routine returns a pointer to the start of the next line. This is also consistent with the model used by visLineLength.

Parameters

| startPos | index to starting character |
| startPosIsLineStart | avoid scanning back to the line start |

Returns

new position as index

33.140.4.39  line_start()

int Fl_Text_Display::line_start ( int pos ) const
Return the beginning of a line.
Same as buffer()->line_start(pos), but returns the character after last wrap point rather than the last newline.

Parameters

| pos | index to starting character |

Returns

new position as index

33.140.4.40 linenumber_align()

void Fl_Text_Display::linenumber_align (Fl_Align val)

Set alignment for line numbers (if enabled).
Valid values are FL_ALIGN_LEFT, FL_ALIGN_CENTER or FL_ALIGN_RIGHT.

Version

1.3.3

33.140.4.41 linenumber_bgcolor()

void Fl_Text_Display::linenumber_bgcolor (Fl_Color val)

Set the background color used for line numbers (if enabled).

Version

1.3.3

33.140.4.42 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

33.140.4.43 linenumber_font()

void Fl_Text_Display::linenumber_font (Fl_Font val)

Set the font used for line numbers (if enabled).

Version

1.3.3
### linenumber_format()

```cpp
void Fl_Text_Display::linenumber_format ( const char * val )
```

Sets the printf() style format string used for line numbers. Default is "%d" for normal unpadded decimal integers. An internal copy of `val` is allocated and managed; it is automatically freed whenever a new value is assigned, or when the widget is destroyed. The value of `val` must not be NULL.

Example values:
- "%d" -- For normal line numbers without padding (Default)
- "%03d" -- For 000 padding
- "%x" -- For hexadecimal line numbers
- "%o" -- For octal line numbers

**Version**

1.3.3

### linenumber_size()

```cpp
void Fl_Text_Display::linenumber_size ( Fl_Fontsize val )
```

Set the font size used for line numbers (if enabled).

**Version**

1.3.3

### linenumber_width()

```cpp
void Fl_Text_Display::linenumber_width ( int width )
```

Set width of screen area for line numbers. Use to also enable/disable line numbers. A value of 0 disables line numbering, values >0 enable the line number display.

**Parameters**

| width | The new width of the area for line numbers to appear, in pixels. 0 disables line numbers (default) |

### longest_vline()

```cpp
int Fl_Text_Display::longest_vline ( ) const [protected]
```

Find the longest line of all visible lines.

**Returns**

the width of the longest visible line in pixels

### maintain_absolute_top_line_number()

```cpp
void Fl_Text_Display::maintain_absolute_top_line_number ( int state ) [protected]
```

Generated by Doxygen
Line numbering stuff, currently unused.
In continuous wrap mode, internal line numbers are calculated after wrapping. A separate non-wrapped line count is maintained when line numbering is turned on. There is some performance cost to maintaining this line count, so normally absolute line numbers are not tracked if line numbering is off. This routine allows callers to specify that they still want this line count maintained (for use via Fl_Text_Display::position_to_linecol()). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

33.140.4.49 maintaining_absolute_top_line_number()

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).

33.140.4.50 measure_deleted_lines()

void Fl_Text_Display::measure_deleted_lines ( int pos, int nDeleted ) [protected]
Wrapping calculations.
This is a stripped-down version of the findWrapRange() function above, intended to be used to calculate the number of "deleted" lines during a buffer modification. It is called before the modification takes place.
This function should only be called in continuous wrap mode with a non-fixed font width. In that case, it is impossible to calculate the number of deleted lines, because the necessary style information is no longer available after the modification. In other cases, we can still perform the calculation afterwards (possibly even more efficiently).

Parameters

<table>
<thead>
<tr>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDeleted</td>
</tr>
</tbody>
</table>

33.140.4.51 measure_proportional_character()

double Fl_Text_Display::measure_proportional_character ( const char * s, int xPix, int pos ) const [protected]
Wrapping calculations.
Measure the width in pixels of the first character of string "s" at a particular column "colNum" and buffer position "pos". This is for measuring characters in proportional or mixed-width highlighting fonts.
A note about proportional and mixed-width fonts: the mixed width and proportional font code in nedit does not get much use in general editing, because nedit doesn't allow per-language-mode fonts, and editing programs in a proportional font is usually a bad idea, so very few users would choose a proportional font as a default. There are still probably mixed- width syntax highlighting cases where things don't redraw properly for insertion/deletion, though static display and wrapping and resizing should now be solid because they are now used for online help display.

Parameters

<table>
<thead>
<tr>
<th>s</th>
<th>text string</th>
</tr>
</thead>
<tbody>
<tr>
<td>xPix</td>
<td>x pixel position needed for calculating tab widths</td>
</tr>
<tr>
<td>pos</td>
<td>offset within string</td>
</tr>
</tbody>
</table>
Returns:
width of character in pixels

33.140.4.52 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

33.140.4.53 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

33.140.4.54 move_left()

```cpp
int Fl_Text_Display::move_left ( )
```

Moves the current insert position left one character.

Returns:
1 if the cursor moved, 0 if the beginning of the text was reached

33.140.4.55 move_right()

```cpp
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

33.140.4.56 move_up()

```cpp
int Fl_Text_Display::move_up ( )
```

Moves the current insert position up one line.

Returns:
1 if the cursor moved, 0 if the beginning of the text was reached
Offset line 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

---

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.

---

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
33.140.4.60  **position_to_line()**

```cpp
int Fl_Text_Display::position_to_line ( int pos, int * lineNum ) const [protected]
```

Convert a position index into a line number offset. Find the line number of position `pos` relative to the first line of displayed text, counting from 0 to `visible lines - 1`. The line number is returned in `lineNum`. Returns 0 if the line is not displayed. In this case `lineNum` is 0 as well. Returns 1 if the line is displayed. In this case `lineNum` is the relative line number.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pos</th>
<th>byte position in buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>lineNum</td>
<td>relative line number of byte <code>pos</code> in buffer</td>
</tr>
</tbody>
</table>

### Returns

whether the character at byte position `pos` is currently displayed

### Return values

<table>
<thead>
<tr>
<th>0</th>
<th>pos is not displayed; <code>lineNum</code> is invalid (zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pos is displayed; <code>lineNum</code> is valid</td>
</tr>
</tbody>
</table>

33.140.4.61  **position_to_linecol()**

```cpp
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 lineNum</td>
<td>absolute (unwrapped) line number</td>
</tr>
<tr>
<td>out column</td>
<td>character offset to the beginning of the line</td>
</tr>
</tbody>
</table>

### Returns

0 if `pos` is off screen, line number otherwise

**Todo** a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

33.140.4.62  **position_to_xy()**

```cpp
int Fl_Text_Display::position_to_xy ( int pos,
```
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 pixel position of character on screen</td>
</tr>
</tbody>
</table>

Returns

0 if character vertically out of view, X & Y positions otherwise

### 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

| startpos | index of first character needing redraw |
| endpos   | index after last character needing redraw |

### 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.

### 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.
### 33.140.4.66 rewind_lines()

```c
int Fl_Text_Display::rewind_lines (  
    int startPos,
    int nLines )
```

Skip a number of lines back. Same as `buffer() > rewind_lines(startPos, nLines)`, but takes into account line breaks when wrapping is turned on.

**Parameters**

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLines</td>
<td>number of lines to skip back</td>
</tr>
</tbody>
</table>

**Returns**

new position as index

### 33.140.4.67 scroll()

```c
void Fl_Text_Display::scroll (  
    int topLineNum,  
    int horizOffset )
```

Scrolls the current buffer to start at the specified line and column.

**Parameters**

<table>
<thead>
<tr>
<th>topLineNum</th>
<th>top line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizOffset</td>
<td>column number</td>
</tr>
</tbody>
</table>

**Todo** Column numbers make little sense here.

### 33.140.4.68 scroll(

```c
int Fl_Text_Display::scroll (  
    int topLineNum,  
    int horizOffset ) [protected]
```

Scrolls the current buffer to start at the specified line and column.

**Parameters**

<table>
<thead>
<tr>
<th>topLineNum</th>
<th>top line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizOffset</td>
<td>in pixels</td>
</tr>
</tbody>
</table>

**Returns**

0 if nothing changed, 1 if we scrolled

### 33.140.4.69 scroll_timer_cb()

```c
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.

33.140.4.70 scrollbar_align() [1/2]

Fl_Align Fl_Text_Display::scrollbar_align () const [inline]

Gets the scrollbar alignment type.

Returns

scrollbar alignment

33.140.4.71 scrollbar_align() [2/2]

void Fl_Text_Display::scrollbar_align ( Fl_Align a ) [inline]

Sets the scrollbar alignment type.

Parameters

a | new scrollbar alignment

33.140.4.72 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)

33.140.4.73 scrollbar_size() [2/2]

void Fl_Text_Display::scrollbar_size ( int newSize ) [inline]

Sets the pixel size of the scrollbars' troughs to newSize, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting newSize to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

<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()

### 33.140.4.74 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.

### 33.140.4.75 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.

### 33.140.4.76 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

### 33.140.4.77 secondary_selection_color() [2/2]

```cpp
void Fl_Text_Display::secondary_selection_color ( Fl_Color color ) [inline]
```

Sets the background color for the secondary selection block.

**Parameters**

| color  | background color |

### 33.140.4.78 shortcut() [1/2]

```cpp
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
33.140.4.79 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

33.140.4.80 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).

33.140.4.81 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()

33.140.4.82 skip_lines()

int Fl_Text_Display::skip_lines ( int startPos, int nLines, bool startPosIsLineStart )

Skip a number of lines forward.
Same as Fl_Text_Buffer::skip_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on. If the caller knows that startPos is at a line start, it can pass startPosIsLineStart as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

<table>
<thead>
<tr>
<th>startPos</th>
<th>index to starting character</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLines</td>
<td>number of lines to skip ahead</td>
</tr>
<tr>
<td>startPosIsLineStart</td>
<td>avoid scanning back to the line start</td>
</tr>
</tbody>
</table>

Returns

new position as index

33.140.4.83 spelling_underline_color() [1/2]

Fl_Color Fl_Text_Display::spelling_underline_color ( ) const [inline]
Gets the underline color for style attribute ATTR_SPELLING.

Returns
underline color

### 33.140.4.84 spelling_underline_color() [2/2]

```cpp
void Fl_Text_Display::spelling_underline_color (
    Fl_Color color) [inline]
```
Sets the underline color for style attribute ATTR_SPELLING.

**Parameters**
- `color` underline color

### 33.140.4.85 string_width()

```cpp
double Fl_Text_Display::string_width (const char *string, int length, int style) const [protected]
```
Find the width of a string in the font of a particular style.

**Parameters**
- `string` the text
- `length` number of bytes in string
- `style` index into style table

Returns
width of text segment in pixels

### 33.140.4.86 style_buffer()

```cpp
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()`

### 33.140.4.87 textcolor() [1/2]

```cpp
Fl_Color Fl_Text_Display::textcolor() const [inline]
```
Gets the default color of text in the widget.
33.140.4.88  **textcolor()[2/2]**

```cpp
void Fl_Text_Display::textcolor (Fl_Color n) [inline]
```

Sets the default color of text in the widget.

**Parameters**

- `n` | new text color

33.140.4.89  **textfont()[1/2]**

```cpp
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

33.140.4.90  **textfont()[2/2]**

```cpp
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

33.140.4.91  **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

33.140.4.92  **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

---

Generated by Doxygen
33.140.4.93  update_h_scrollbar()

void Fl_Text_Display::update_h_scrollbar ( ) [protected]
Update horizontal scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the horizontal scrollbar.

33.140.4.94  update_line_starts()

void Fl_Text_Display::update_line_starts ( int pos, int charsInserted, int charsDeleted, int linesInserted, int linesDeleted, int * scrolled ) [protected]
Update line start arrays and variables.
Update the line starts array, mTopLineNum, mFirstChar and lastChar for this text display after a modification to the text buffer, given by the position pos where the change began, and the numbers of characters and lines inserted and deleted.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>index into buffer of recent changes</td>
</tr>
<tr>
<td>charsInserted</td>
<td>number of bytes(!) inserted</td>
</tr>
<tr>
<td>charsDeleted</td>
<td>number of bytes(!) deleted</td>
</tr>
<tr>
<td>linesInserted</td>
<td>number of lines</td>
</tr>
<tr>
<td>linesDeleted</td>
<td>number of lines</td>
</tr>
<tr>
<td>scrolled</td>
<td>set to 1 if the text display needs to be scrolled</td>
</tr>
</tbody>
</table>

33.140.4.95  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.

33.140.4.96  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

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visLineNum</td>
<td>index of line in visible line array</td>
</tr>
</tbody>
</table>
number of bytes in this line

33.140.4.97 word_end()

int Fl_Text_Display::word_end (  
  int pos ) const  [inline]
Moves the insert position to the end of the current word.

Parameters

\begin{itemize}
  \item \texttt{pos} \begin{tabular}{l}
    start calculation at this index
  \end{tabular}
\end{itemize}

Returns

index of first character after the end of the word

33.140.4.98 word_start()

int Fl_Text_Display::word_start (  
  int pos ) const  [inline]
Moves the insert position to the beginning of the current word.

Parameters

\begin{itemize}
  \item \texttt{pos} \begin{tabular}{l}
    start calculation at this index
  \end{tabular}
\end{itemize}

Returns

beginning of the words

33.140.4.99 wrap_mode()

void Fl_Text_Display::wrap_mode (  
  int wrap,  
  int wrapMargin )
Set the new text wrap mode.
If \texttt{wrap} mode is not zero, this call enables automatic word wrapping at column \texttt{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:

\begin{itemize}
  \item WRAP_NONE : don't wrap text at all
  \item WRAP_AT_COLUMN : wrap text at the given text column
  \item WRAP_AT_PIXEL : wrap text at a pixel position
  \item WRAP_AT_BOUNDS : wrap text so that it fits into the widget width
\end{itemize}

Parameters

\begin{itemize}
  \item \texttt{wrap} \begin{tabular}{l}
    new\ wrap\ mode\ (\ see\ above)
  \end{tabular}
\end{itemize}
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.

Parameters

<table>
<thead>
<tr>
<th>wrapMargin</th>
</tr>
</thead>
</table>
| 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. |

33.140.4.100   wrap_uses_character()

int Fl_Text_Display::wrap_uses_character (int lineEndPos) const [protected]

Check if the line break is caused by a newline or by line wrapping.
Line breaks in continuous wrap mode usually happen at newlines (\n) or whitespace. This line-terminating character
is not included in line width measurements and has a special status as a non-visible character. However, lines with
no whitespace are wrapped without the benefit of a line terminating character, and this distinction causes endless
trouble with all of the text display code which was originally written without continuous wrap mode and always
expects to wrap at a newline character.
Given the position of the end of the line, as returned by Fl_Text_Display::line_end() or Fl_Text_Buffer::line_end(),
this returns true if there is a line terminating character, and false if there's not. On the last character in the buffer,
this function can't tell for certain whether a trailing space was used as a wrap point, and just guesses that it wasn't.
So if an exact accounting is necessary, don't use this function.

Parameters

<table>
<thead>
<tr>
<th>lineEndPos</th>
</tr>
</thead>
<tbody>
<tr>
<td>index of character where the line wraps</td>
</tr>
</tbody>
</table>

Returns

1 if a \n character causes the line wrap

33.140.4.101   wrapped_column()

int Fl_Text_Display::wrapped_column (int row, int column) const

Nobody knows what this function does.
Correct a column number based on an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to
be relative to the last actual newline in the buffer before the row and column position given, rather than the last line
start created by line wrapping. This is an adapter for rectangular selections and code written before continuous wrap
mode, which thinks that the unconstrained column is the number of characters from the last newline. Obviously this
is time consuming, because it involves character re-counting.

Parameters

| row |
| column |

Returns

something unknown

Todo What does this do and how is it useful? Column numbers mean little in this context. Which functions depend
on this one? Function TextDXYToUnconstrainedPosition does not exist (nedt port?)

Todo Unicode?
33.140.4.102 \texttt{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 \texttt{startPos} to either \texttt{maxPos} or \texttt{maxLines} (whichever is reached first), and return all relevant positions and line count. The provided \texttt{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 \texttt{styleBufOffset} argument must indicate the starting position of the copy, to take into account the correct style information.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>buf</th>
<th>The text buffer to operate on</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>startPos</td>
<td>Starting index position into the buffer</td>
</tr>
<tr>
<td>in</td>
<td>maxPos</td>
<td>Maximum index position into the buffer we'll reach</td>
</tr>
<tr>
<td>in</td>
<td>maxLines</td>
<td>Maximum number of lines we'll reach</td>
</tr>
<tr>
<td>in</td>
<td>startPosIsLineStart</td>
<td>Flag indicating if \texttt{startPos} is start of line. (If set, prevents our having to find the line start)</td>
</tr>
<tr>
<td>in</td>
<td>styleBufOffset</td>
<td>Offset index position into style buffer.</td>
</tr>
<tr>
<td>out</td>
<td>retPos</td>
<td>Position where counting ended. When counting lines, the position returned is the start of the line &quot;\texttt{maxLines}&quot; lines beyond &quot;\texttt{startPos}&quot;.</td>
</tr>
<tr>
<td>out</td>
<td>retLines</td>
<td>Number of line breaks counted</td>
</tr>
<tr>
<td>out</td>
<td>retLineStart</td>
<td>Start of the line where counting ended</td>
</tr>
<tr>
<td>out</td>
<td>retLineEnd</td>
<td>End position of the last line traversed</td>
</tr>
<tr>
<td>out</td>
<td>countLastLineMissingNewLine</td>
<td></td>
</tr>
</tbody>
</table>

33.140.4.103 \texttt{wrapped\_row()}

```cpp
int Fl_Text_Display::wrapped_row (  
    int row ) const
```

Nobody knows what this function does.
Correct a row number from an unconstrained position (as returned by \texttt{TextDXYToUnconstrainedPosition}) to a straight number of newlines from the top line of the display. Because rectangular selections are based on newlines, rather than display wrapping, and anywhere a rectangular selection needs a row, it needs it in terms of un-wrapped lines.

**Parameters**

| row | |
|-----|
Returns

something unknown

Todo  What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

33.140.4.104  x_to_col()

double Fl_Text_Display::x_to_col (  
double x ) const
Convert an x pixel position into a column number.

Parameters

| x  | number of pixels from the left margin |

Returns

an approximate column number based on the main font

33.140.4.105  xy_to_position()

int Fl_Text_Display::xy_to_position (  
    int X,  
    int Y,  
    int posType = CHARACTER_POS ) const [protected]
Translate a pixel position into a character index.
Translate window coordinates to the nearest (insert cursor or character cell) text position. The parameter posType specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest cursor position, and CHARACTER_POS means return the position of the character closest to (X, Y).

Parameters

| X, Y | pixel position |
| posType | CURSOR_POS or CHARACTER_POS |

Returns

index into text buffer

33.140.4.106  xy_to_rowcol()

void Fl_Text_Display::xy_to_rowcol (  
    int X,  
    int Y,  
    int * row,  
    int * column,  
    int posType = CHARACTER_POS ) const [protected]
Translate pixel coordinates into row and column.
Translate window coordinates to the nearest row and column number for positioning the cursor. This, of course, makes no sense when the font is proportional, since there are no absolute columns. The parameter posType specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest position between characters, and CHARACTER_POS means translate the position to the nearest character cell.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X, Y</td>
<td>pixel coordinates</td>
</tr>
<tr>
<td>out</td>
<td>row, column</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

### 33.141 Fl_Text_Editor Class Reference

This is the FLTK text editor widget.

```c
#include <Fl_Text_Editor.H>
```

Inheritance diagram for Fl_Text_Editor:

```
Fl_Widget
   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.
• int handle (int e) FL_OVERRIDE
  
  Event handling.
• int insert_mode ()
  
  Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.
• void insert_mode (int b)
  
  Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.
• void remove_all_key_bindings ()
  
  Removes all of the key bindings associated with the text editor or list.
• void remove_all_key_bindings (Key_Binding **list)
  
  Removes all of the key bindings associated with the text editor or list.
• void remove_key_binding (int key, int state)
  
  Removes the key binding associated with the key "key" of state "state".
• void remove_key_binding (int key, int state, Key_Binding **list)
  
  Removes the key binding associated with the key key of state state from the Key_Binding list list.
• int tab_nav () const
  
  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'.
• 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_home (int c, Fl_Text_Editor *e)
  
  Moves the text cursor to the beginning of the current line in editor 'e'.
• static int kf_ignore (int c, Fl_Text_Editor *e)
  
  Ignores the key 'c' in editor 'e'.
• static int kf_insert (int c, Fl_Text_Editor *e)
  
  Toggles the insert mode for editor 'e'.
• static int kf_left (int c, Fl_Text_Editor *e)
  
  Moves the text cursor one character to the left in editor 'e'.
• static int kf_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_redo (int c, Fl_Text_Editor *e)
  Redo last undo action.
• static int kf_right (int c, Fl_Text_Editor *e)
  Moves the text cursor one character to the right for editor 'e'.
• static int kf_select_all (int c, Fl_Text_Editor *e)
  Selects all text in the current buffer in editor 'e'.
• static int kf_shift_move (int c, Fl_Text_Editor *e)
  Extends the current selection in the direction of key 'c' in editor 'e'.
• static int kf_undo (int c, Fl_Text_Editor *e)
  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 (Fl_Callback_Reason reason=FL_REASON_CHANGED)
  does or does not a callback according to changed() and when() settings

Static Protected Attributes

• static Key_Binding * global_keyBindings
  Global key binding list.

Additional Inherited Members

33.141.1 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.

33.141.2 Member Function Documentation

33.141.2.1 add_key_binding()

void Fl_Text_Editor::add_key_binding (  
  int key,  
  int state,  
  Key_Func function,  
  Key_Binding ** list )

Adds a key of state state with the function function to an arbitrary key binding list list.
This can be used in derived classes to add global key bindings by using the global (static) Key_Binding list
Fl_Text_Editor::global_keyBindings.
33.141.2.2 handle()

int Fl_Text_Editor::handle (  
    int e  ) [virtual]

Event handling.
Reimplemented from Fl_Text_Display.

33.141.2.3 insert_mode() [1/2]

int Fl_Text_Editor::insert_mode ( ) [inline]

Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.  
Otherwise, new text replaces text at the current cursor position.

33.141.2.4 insert_mode() [2/2]

void Fl_Text_Editor::insert_mode (  
    int b  ) [inline]

Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.  
Otherwise, new text replaces text at the current cursor position.

33.141.2.5 kf_backspace()

int Fl_Text_Editor::kf_backspace (  
    int c,  
    Fl_Text_Editor * e  ) [static]

Does a backspace for key 'c' in the current buffer of editor 'e'.  
Any current selection is deleted. Otherwise, the character left is deleted and the cursor moved. The key value 'c' is currently unused.

33.141.2.6 kf_c_s_move()

int Fl_Text_Editor::kf_c_s_move (  
    int c,  
    Fl_Text_Editor * e  ) [static]

Extends the current selection in the direction indicated by control key 'c' in editor 'e'.  
See also  
    kf_ctrl_move().

33.141.2.7 kf_copy()

int Fl_Text_Editor::kf_copy (  
    int c,  
    Fl_Text_Editor * e  ) [static]

Does a copy of selected text or the current character in the current buffer of editor 'e'.  
The key value 'c' is currently unused.

33.141.2.8 kf_ctrl_move()

int Fl_Text_Editor::kf_ctrl_move (  
    int c,  
    Fl_Text_Editor * e  ) [static]

Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.  
Supported values for 'c' are currently:  
    FL_Home -- moves the cursor to the beginning of the document  
    FL_End   -- moves the cursor to the end of the document  
    FL_Left  -- moves the cursor left one word  
    FL_Right -- moves the cursor right one word
33.141.2.9 \texttt{kf\_cut()}

```cpp
int Fl_Text_Editor::kf_cut (  
    int c,
    Fl_Text_Editor * e ) [static]
```

Does a cut of selected text in the current buffer of editor 'e'.
The key value 'c' is currently unused.

33.141.2.10 \texttt{kf\_default()}

```cpp
int Fl_Text_Editor::kf_default (  
    int c,
    Fl_Text_Editor * e ) [static]
```

Inserts the text associated with key 'c' in editor 'e'.
Honors the current selection and insert/overstrike mode.

33.141.2.11 \texttt{kf\_delete()}

```cpp
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.

33.141.2.12 \texttt{kf\_down()}

```cpp
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 \texttt{kf\_move(FL\_Down, e)}. The key value 'c' is currently unused.

33.141.2.13 \texttt{kf\_end()}

```cpp
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 \texttt{kf\_move(FL\_End, e)}. The key value 'c' is currently unused.

33.141.2.14 \texttt{kf\_enter()}

```cpp
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.

33.141.2.15 \texttt{kf\_home()}

```cpp
int Fl_Text_Editor::kf_home (  
    int c,
    Fl_Text_Editor * e ) [static]
```

Moves the text cursor to the beginning of the current line in editor 'e'.
Same as \texttt{kf\_move(FL\_Home, e)}. The key value 'c' is currently unused.
33.141.2.16  kf_ignore()

int Fl_Text_Editor::kf_ignore (  
   int c,  
   Fl_Text_Editor * e ) [static]

Ignores the key 'c' in editor 'e'.  
This method can be used as a keyboard binding to disable a key that might otherwise be handled or entered as text.  
An example would be disabling FL_Escape, so that it isn't added to the buffer when invoked by the user.

33.141.2.17  kf_insert()

int Fl_Text_Editor::kf_insert (  
   int c,  
   Fl_Text_Editor * e ) [static]

Toggles the insert mode for editor 'e'.  
The key value 'c' is currently unused.

33.141.2.18  kf_left()

int Fl_Text_Editor::kf_left (  
   int c,  
   Fl_Text_Editor * e ) [static]

Moves the text cursor one character to the left in editor 'e'.  
Same as kf_move(FL_Left, e). The key value 'c' is currently unused.

33.141.2.19  kf_m_s_move()

int Fl_Text_Editor::kf_m_s_move (  
   int c,  
   Fl_Text_Editor * e ) [static]

Extends the current selection in the direction indicated by meta key 'c' in editor 'e'.  
See also  
   kf_meta_move().

33.141.2.20  kf_meta_move()

int Fl_Text_Editor::kf_meta_move (  
   int c,  
   Fl_Text_Editor * e ) [static]

Moves the current text cursor in the direction indicated by meta key 'c' in editor 'e'.  
Supported values for 'c' are currently:  
   FL_Up -- moves cursor to the beginning of the current document  
   FL_Down -- moves cursor to the end of the current document  
   FL_Left -- moves the cursor to the beginning of the current line  
   FL_Right -- moves the cursor to the end of the current line

33.141.2.21  kf_move()

int Fl_Text_Editor::kf_move (  
   int c,  
   Fl_Text_Editor * e ) [static]

Moves the text cursor in the direction indicated by key 'c' in editor 'e'.  
Supported values for 'c' are currently:  
   FL_Home -- moves the cursor to the beginning of the current line  
   FL_End -- moves the cursor to the end of the current line  
   FL_Left -- moves the cursor left one character  
   FL_Right -- moves the cursor right one character  
   FL_Up -- moves the cursor up one line  
   FL_Down -- moves the cursor down one line
33.141.2.22  kf_page_down()

int Fl_Text_Editor::kf_page_down (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor down one page for editor 'e'.  
Same as kf_move(FL_Page_Down, e). The key value 'c' is currently unused.

33.141.2.23  kf_page_up()

int Fl_Text_Editor::kf_page_up (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor up one page for editor 'e'.  
Same as kf_move(FL_Page_Up, e). The key value 'c' is currently unused.

33.141.2.24  kf_paste()

int Fl_Text_Editor::kf_paste (  
    int c,  
    Fl_Text_Editor * e ) [static]

Does a paste of selected text in the current buffer of editor 'e'.  
Any current selection is replaced with the pasted content. The key value 'c' is currently unused.

33.141.2.25  kf_redo()

int Fl_Text_Editor::kf_redo (  
    int c,  
    Fl_Text_Editor * e ) [static]

Redo last undo action.  
Also deselects previous selection. The key value 'c' is currently unused.

33.141.2.26  kf_right()

int Fl_Text_Editor::kf_right (  
    int c,  
    Fl_Text_Editor * e ) [static]

Moves the text cursor one character to the right for editor 'e'.  
Same as kf_move(FL_Right, e). The key value 'c' is currently unused.

33.141.2.27  kf_select_all()

int Fl_Text_Editor::kf_select_all (  
    int c,  
    Fl_Text_Editor * e ) [static]

Selects all text in the current buffer in editor 'e'.  
The key value 'c' is currently unused.

33.141.2.28  kf_shift_move()

int Fl_Text_Editor::kf_shift_move (  
    int c,  
    Fl_Text_Editor * e ) [static]

Extends the current selection in the direction of key 'c' in editor 'e'.

---

Generated by Doxygen
33.141.2.29  kf_undo()

```
int Fl_Text_Editor::kf_undo (  
   int c,  
   Fl_Text_Editor ∗ e ) [static]
```

Undo last edit in the current buffer of editor 'e'.
Also deselects previous selection. The key value 'c' is currently unused.

33.141.2.30  kf_up()

```
int Fl_Text_Editor::kf_up (  
   int c,  
   Fl_Text_Editor ∗ e ) [static]
```

Moves the text cursor one line up for editor 'e'.
Same as kf_move(FL_Up, e). The key value 'c' is currently unused.

33.141.2.31  remove_key_binding()

```
void Fl_Text_Editor::remove_key_binding (  
   int key,  
   int state,  
   Key_Binding ∗ ∗ list )
```

Removes the key binding associated with the key key of state state from the Key_Binding list list.
This can be used in derived classes to remove global key bindings by using the global (static) Key_Binding list Fl_Text_Editor::global_key_bindings.

33.141.2.32  tab_nav() [1/2]

```
int Fl_Text_Editor::tab_nav ( ) const
```

Check if Tab focus navigation is enabled.
If disabled (default), hitting Tab inserts a tab character into the editor buffer.
If enabled, hitting Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget.

Returns

- if Tab inserts tab characters or moves the focus

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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

33.141.2.33  tab_nav() [2/2]

```
void Fl_Text_Editor::tab_nav (  
   int val )
```

Generated by Doxygen
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>val</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tab inserts a tab character (default).</td>
</tr>
<tr>
<td>1</td>
<td>Tab navigates widget focus.</td>
</tr>
</tbody>
</table>

See also

tab_nav(), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

33.141.3 Member Data Documentation

33.141.3.1 global_key_bindings

Key_Binding* Fl_Text_Editor::global_key_bindings [static], [protected]

Global key binding list. Derived classes can add key bindings for all Fl_Text_Editor widgets by adding a Key_Binding to this list.

See also

add_key_binding(int key, int state, Key_Func f, Key_Binding** list);

The documentation for this class was generated from the following files:

- Fl_Text_Editor.H
- Fl_Text_Editor.cxx

33.142 Fl_Text_Selection Class Reference

This is an internal class for Fl_Text_Buffer to manage text selections.

#include <Fl_Text_Buffer.H>

Public Member Functions

- int end () const
  Returns the byte offset to the character after the last selected character.
- int includes (int pos) const
  Returns true if position pos is in the Fl_Text_Selection.
- int length () const
  Returns the size in bytes of the selection.
- int position (int *startpos, int *endpos) const
- bool selected () const
  Returns true if any text is selected.
- void selected (bool b)
Modifies the 'selected' flag.

- **int selected** (int *startpos, int *endpos) const
  Returns the status and the positions of this selection.

- **void set** (int startpos, int endpos)
  Sets the selection range.

- **int start ()** const
  Returns the byte offset to the first selected character.

- **void update** (int pos, int nDeleted, int nInserted)
  Updates a selection after text was modified.

Protected Attributes

- **int mEnd**
  byte offset to the character after the last selected character

- **bool mSelected**
  this flag is set if any text is selected

- **int mStart**
  byte offset to the first selected character

Friends

- **class Fl_Text_Buffer**

### 33.142.1 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):

```c
if (!selected()) : (start() == 0) && (end() == 0) && (end() == start())
if ( selected() ) : start() < end()
always : 0 <= start() <= end()
always : length() == end() - start()
```

The selection size in bytes can always (unconditionally) be computed by

```c
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.

### 33.142.2 Member Function Documentation

#### 33.142.2.1 end()

```c
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.

33.142.2.2 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.

33.142.2.3 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

```cpp
end() - start()
```

and it returns 0 if selected() == false.

Returns
size in bytes or 0 if not selected.

Since
FLTK 1.4.0

33.142.2.4 position()

```cpp
int Fl_Text_Selection::position ( int * startpos, int * endpos ) const [inline]
```

Deprecated "in 1.4.0 - use selected(startpos, endpos) instead"

33.142.2.5 selected()[1/3]

```cpp
bool Fl_Text_Selection::selected ( ) const [inline]
```

Returns true if any text is selected.

Returns
true if any text has been selected, or false if no text is selected.

33.142.2.6 selected()[2/3]

```cpp
void Fl_Text_Selection::selected ( bool b ) [inline]
```

Modifies the 'selected' flag.
33.142.2.7  selected() [3/3]

```cpp
int Fl_Text_Selection::selected (  
    int * startpos,  
    int * endpos ) const
```

Returns the status and the positions of this selection.

This method returns the same as `selected()` as an `int` (0 or 1) in its return value and the offsets to the start of the selection in `startpos` and to the byte after the last selected character in `endpos`, if `selected()` is true. If `selected()` is false, both offsets are set to 0.

**Note**

In FLTK 1.3.x `startpos` and `endpos` were **not modified** if `selected()` was false.

**Parameters**

- `startpos` : return byte offset to first selected character
- `endpos` : return byte offset pointing after last selected character

**Returns**

whether the selection is active (`selected()`) or not

**Return values**

- 0 if not selected
- 1 if selected

See also

- `selected()`, `start()`, `end()`

33.142.2.8  set()

```cpp
void Fl_Text_Selection::set (  
    int startpos,  
    int endpos )
```

Sets the selection range.

`startpos` and `endpos` must be at a character boundary.

If `startpos != endpos` `selected()` is set to true, else to false.

If `startpos` is greater than `endpos` they are swapped so that `startpos <= endpos`.

**Parameters**

- `in startpos` : byte offset to first selected character
- `in endpos` : byte offset pointing after last selected character
33.142.2.9 start()

```cpp
int Fl_Text_Selection::start () const [inline]
```

Returns the byte offset to the first selected character. The returned offset is only valid if `selected()` returns true. If the selection is not valid the returned offset is 0 since FLTK 1.4.0.

Notes

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.

33.142.2.10 update()

```cpp
void Fl_Text_Selection::update (int pos, int nDeleted, int nInserted )
```

Updates a selection after text was modified. Updates an individual selection for changes in the corresponding text.

Parameters

- `pos` byte offset into the text buffer at which the change occurred
- `nDeleted` number of bytes deleted from the buffer
- `nInserted` number of bytes inserted into the buffer

The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

33.143 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_Tile
    Fl_Group
       Fl_Widget

```

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) FL_OVERRIDE`  
  Handles the specified event.
- `void move_intersection (int oldx, int oldy, int newx, int newy)`
Drags the intersection at \((oldx, oldy)\) to \((newx, newy)\).

- void \texttt{position} (int oldx, int oldy, int newx, int newy)
- void \texttt{position} (int x, int y)
- void \texttt{resize} (int X, int Y, int W, int H) \texttt{FL\_OVERRIDE}

Resizes the \texttt{Fl\_Tile} widget and its children.

### Additional Inherited Members

#### 33.143.1 Detailed Description

The \texttt{Fl\_Tile} class lets you resize its children by dragging the border between them.

![Figure 33.52 Fl\_Tile](image)

For the tiling to work correctly, the children of an \texttt{Fl\_Tile} must cover the entire area of the widget, but not overlap. This means that all children must touch each other at their edges, and no gaps can be left inside the \texttt{Fl\_Tile}.

\texttt{Fl\_Tile} does not normally draw any graphics of its own. The "borders" which can be seen in the snapshot above are actually part of the children. Their boxtypes have been set to \texttt{FL\_DOWN\_BOX} creating the impression of "ridges" where the boxes touch. What you see are actually two adjacent \texttt{FL\_DOWN\_BOX}'s drawn next to each other. All neighboring widgets share the same edge - the widget's thick borders make it appear as though the widgets aren't actually touching, but they are. If the edges of adjacent widgets do not touch, then it will be impossible to drag the corresponding edges.

\texttt{Fl\_Tile} allows objects to be resized to zero dimensions. To prevent this you can use the \texttt{resizable()} to limit where corners can be dragged to. For more information see note below. Even though objects can be resized to zero sizes, they must initially have non-zero sizes so the \texttt{Fl\_Tile} can figure out their layout. If desired, call \texttt{position()} after creating the children but before displaying the window to set the borders where you want.

**Note on \texttt{resizable(Fl\_Widget \&w)}**

The "resizable" child widget (which should be invisible) limits where the borders can be dragged to. All dragging will be limited inside the resizable widget's borders. If you don't set it, it will be possible to drag the borders right to the edges of the \texttt{Fl\_Tile} widget, and thus resize objects on the edges to zero width or height. When the entire \texttt{Fl\_Tile} widget is resized, the \texttt{resizable()} widget will keep its border distance to all borders the same (this is normal resize behavior), so that you can effectively set a border width that will never change. To ensure correct event delivery to all child widgets the \texttt{resizable()} widget must be the first child of the \texttt{Fl\_Tile} widget group. Otherwise some events (e.g. \texttt{FL\_MOVE} and \texttt{FL\_ENTER}) might be consumed by the \texttt{resizable()} widget so that they are lost for widgets covered (overlapped) by the \texttt{resizable()} widget.

Note

You can still resize widgets \texttt{inside} the \texttt{resizable()} to zero width and/or height, i.e. box \texttt{2b} above to zero width and box \texttt{3a} to zero height.
See also

```cpp
void Fl_Group::resizable(Fl_Widget &w)
```

Example for resizable with 20 pixel border distance:

```cpp
int dx = 20, dy = dx;
Fl_Tile tile(50,50,300,300);
// create resizable() box first
Fl_Box r(tile.x()+dx,tile.y()+dy,tile.w()-2*dx,tile.h()-2*dy);
tile.resizable(r);
// ... create widgets inside tile (see test/tile.cxx) ...
tile.end();
```

See also the complete example program in test/tile.cxx.

### 33.143.2 Constructor & Destructor Documentation

#### 33.143.2.1 Fl_Tile()

```cpp
Fl_Tile::Fl_Tile ( int X, int Y, int W, int H, const char ∗L = 0 )
```

Creates a new `Fl_Tile` widget using the given position, size, and label string. The default boxtype is `FL_NO_BOX`. The destructor also deletes all the children. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the `Fl_Tile` and all of its children can be automatic (local) variables, but you must declare the `Fl_Tile` first, so that it is destroyed last.

See also

```cpp
class Fl_Group
```

### 33.143.3 Member Function Documentation

#### 33.143.3.1 handle()

```cpp
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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>event</td>
<td>the kind of event received</td>
</tr>
</tbody>
</table>

**Return values**

- 0 if the event was not used or understood
- 1 if the event was used and can be deleted
See also

Fl_Event

Reimplemented from Fl_Group.

### 33.143.3.2 move_intersection()

```c
void Fl_Tile::move_intersection (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.

### 33.143.3.3 position()

```c
void Fl_Tile::position (int oldx, int oldy, int newx, int newy) [inline]
```

Deprecated "in 1.4.0 - use move_intersection(p) instead"

### 33.143.3.4 resize()

```c
void Fl_Tile::resize (int X, int Y, int W, int H) [virtual]
```

Resizes the Fl_Tile widget and its children. Fl_Tile implements its own resize() method. It does not use Fl_Group::resize() to resize itself and its children. 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

### 33.144 Fl_Tiled_Image Class Reference

This class supports tiling of images over a specified area.

```
#include <Fl_Tiled_Image.H>
```

Inheritance diagram for Fl_Tiled_Image:
Public Member Functions

- void color_average (FL_Color c, float i) FL_OVERRIDE
  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy () const
- Fl_Image * copy (int W, int H) const FL_OVERRIDE
  Creates a resized copy of the image.
- void desaturate () FL_OVERRIDE
  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE
  Draws a tiled image.
- Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)
  The constructors create a new tiled image containing the specified image.
- Fl_Image * image ()
  Gets The image that is tiled.
- virtual ~Fl_Tiled_Image ()
  The destructor frees all memory and server resources that are used by the tiled image.

Protected Attributes

- int alloc_image_
- Fl_Image * image_

Additional Inherited Members

33.144.1 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.

33.144.2 Constructor & Destructor Documentation

33.144.2.1 Fl_Tiled_Image()

Fl_Tiled_Image::Fl_Tiled_Image ( Fl_Image * i,
 int W = 0,
 int H = 0 )

The constructors create a new tiled image containing the specified image. Use a width and height of 0 to tile the whole window/widget.
Note

Due to implementation constraints in FLTK 1.3.3 and later width and height of 0 may not work as expected when used as background image in widgets other than windows. You may need to center and clip the image (label) and set the label type to FL_NORMAL_LABEL. Doing so will let the tiled image fill the whole widget as its background image. Other combinations of label flags may or may not work.

```cpp
#include "bg.xpm"
FL_Pixmap *bg_xpm = new Fl_Pixmap(bg_xpm);
Fl_Tiled_Image *bg_tiled = new Fl_Tiled_Image(bg_xpm, 0, 0);
Fl_Box *box = new Fl_Box(40, 40, 300, 100, "");
box->box(FL_UP_BOX);
box->labeltype(FL_NORMAL_LABEL);
box->align(FL_ALIGN_INSIDE | FL_ALIGN_CENTER | FL_ALIGN_CLIP);
box->image(bg_tiled);
```

Note

Setting an image (label) for a window may not work as expected due to implementation constraints in FLTK 1.3.x and maybe later. The reason is the way Fl::scheme() initializes the window's label type and image. A possible workaround is to use another Fl_Group as the only child widget and to set the background image for this group as described above.

Todo

Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char *).

33.144.3 Member Function Documentation

33.144.3.1 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.

33.144.3.2 copy()

```cpp
Fl_Image * Fl_Tiled_Image::copy (int W, int H) const [virtual]
```

Creates a resized copy of the image. The new image should be released when you are done with it.

Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator delete for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's. The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling(). For the new image the following equations are true:

- \( w() = data_w() = W \)
- \( h() = data_h() = H \)

Note: the returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>W,H</td>
</tr>
</tbody>
</table>

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.

33.144.3.3 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.

33.144.3.4 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 H 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

33.145 FL_Timeout Class Reference

The internal class FL_Timeout handles all timeout related functions.
#include <FL_Timeout.h>

Generated by Doxygen
Static Public Member Functions

- static void `add_timeout` (double time, Fl_Timeout_Handler cb, void *data)
  Adds a one-shot timeout callback.

- static void `do_timeouts` ()
  Elapse timers and call their callbacks if any timers are expired.

- static void `elapse_timeouts` ()
  Elapse all timers w/o calling their callbacks.

- static int `has_timeout` (Fl_Timeout_Handler cb, void *data)
  Returns true if the timeout exists and has not been called yet.

- static void `remove_timeout` (Fl_Timeout_Handler cb, void *data)
  Remove a timeout callback.

- static void `repeat_timeout` (double time, Fl_Timeout_Handler cb, void *data)
  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

- static double `time_to_wait` (double 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.
33.145 Fl_Timeout Class Reference

33.145.1 Detailed Description

The internal class Fl_Timeout handles all timeout related functions. All code is platform independent except retrieving a timestamp which requires calling a system driver function and potentially results in different timer resolutions (from milliseconds to microseconds).

Related user documentation:

- Fl_Timeout_Handler
- Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)
- Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
- Fl::has_timeout(Fl_Timeout_Handler cb, void *data)
- Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)

33.145.2 Member Function Documentation

33.145.2.1 add_timeout()

```c
void Fl_Timeout::add_timeout (  
    double time,  
    Fl_Timeout_Handler cb,  
    void * data ) [static]
```

Adds a one-shot timeout callback. The callback function cb will be called by Fl::wait() at time seconds after this function is called.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>delta time in seconds until the timer expires</td>
</tr>
<tr>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

Implements Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)

See also

Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)

33.145.2.2 current()

```c
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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if no callback is currently running.</td>
</tr>
</tbody>
</table>
33.145.2.3  elapse_timeouts()

void Fl_Timeout::elapse_timeouts ( ) [static]
Elapse all timers w/o calling their callbacks.
All timer values are adjusted by the delta time since the last call. This method does NOT call timer callbacks if timers are expired.
This must be called before new timers are added to the timer queue to make sure that the next timer decrement does not count down too much time.
See also
    Fl_Timeout::do_timeouts()

33.145.2.4  get()

Fl_Timeout * Fl_Timeout::get (  
    double time,
    Fl_Timeout_Handler cb,
    void * data ) [static], [protected]
Get an Fl_Timeout instance for further handling.
The timer object will be initialized with the input parameters as given by Fl::add_timeout() or Fl::repeat_timeout().
Fl_Timeout objects are maintained in three queues:

- active timer queue
- list (stack, i.e. LIFO) of currently executing timer callbacks
- free timer entries.

When the FLTK program is launched all queues are empty. Whenever a new timer object is required the get() method is called and a timer object is either found in the queue of free timer entries or a new timer object is created (operator new).
Active timer entries are inserted into the "active timer queue" until they expire and their callback is called.
Before the callback is called the timer entry is inserted into the list of current timers, i.e. it becomes the 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()

33.145.2.5  has_timeout()

int Fl_Timeout::has_timeout (   

Generated by Doxygen
Fl_Timeout_Handler cb, 
void * data) [static]

Returns true if the timeout exists and has not been called yet.

Parameters

<table>
<thead>
<tr>
<th>in cb</th>
<th>Timer callback (must match)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in data</td>
<td>Callback user data (must match)</td>
</tr>
</tbody>
</table>

Returns

whether the timer was found in the queue

Return values

| 0 | not found |
| 1 | found |

Implements Fl::has_timeout(Fl_Timeout_Handler cb, void *data)

See also

Fl::has_timeout(Fl_Timeout_Handler cb, void *data)

33.145.2.6 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.

33.145.2.7 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()

33.145.2.8 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 \p t has expired, run its callback 
(t->callback)(t->data); 
t->release();

33.145.2.9 remove_timeout()

void Fl_Timeout::remove_timeout ( 
    Fl_Timeout_Handler cb, 
    void * data) [static]
Remove a timeout callback.
This method removes all matching timeouts, not just the first one. This may change in the future.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cb</th>
<th>Timer callback to be removed (must match)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>Wildcard if NULL, must match otherwise</td>
</tr>
</tbody>
</table>

Implements `Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)`

See also

`Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)`

### 33.145.2.10 repeat_timeout()

```cpp
void Fl_Timeout::repeat_timeout (  
    double time,  
    Fl_Timeout_Handler cb,  
    void * data ) [static]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>delta time in seconds until the timer expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cb</td>
<td>callback function</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>optional user data (default: NULL)</td>
</tr>
</tbody>
</table>

Implements `Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)`

See also

`Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)`

### 33.145.2.11 time_to_wait()

double Fl_Timeout::time_to_wait (  
    double ttw ) [static]

Returns the delay in seconds until the next timer expires, limited by `ttw`.

This function calculates the time to wait for the FLTK event queue processing, depending on the given value `ttw`.

If at least one timer is active and its timeout value is smaller than `ttw` then this value is returned. `Fl::wait()` will wait no longer than until the next timer expires.

If no timer is active this returns the input value `ttw` unchanged.

If at least one timer is expired this returns 0.0 so the event processing does not wait.

Parameters

| in  | ttw         | time to wait from `Fl::wait()` etc. (upper limit) |

Generated by Doxygen
Returns
delay until next timeout or 0.0 (see description)

33.145.3 Member Data Documentation

33.145.3.1 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)

33.145.3.2 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.

33.145.3.3 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

33.146 Fl_Timer Class Reference

This is provided only to emulate the Forms Timer widget.
#include <Fl_Timer.H>
Inheritance diagram for Fl_Timer:
Public Member Functions

- char direction () const
  Gets or sets the direction of the timer.

- void direction (char d)
  Gets or sets the direction of the timer.

- Fl_Timer (uchar t, int x, int y, int w, int h, const char ∗l)
  Creates a new Fl_Timer widget using the given type, position, size, and label string.

- int handle (int) FL_OVERRIDE
  Handles the specified event.

- char suspended () const
  Gets or sets whether the timer is suspended.

- void suspended (char d)
  Gets or sets whether the timer is suspended.

- double value () const
  See void Fl_Timer::value(double)

- void value (double)
  Sets the current timer value.

- ~Fl_Timer ()
  Destroys the timer and removes the timeout.

Protected Member Functions

- void draw () FL_OVERRIDE
  Draws the widget.

Additional Inherited Members

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.

Constructor & Destructor Documentation

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.
33.146.3 Member Function Documentation

33.146.3.1 direction() [1/2]

```cpp
char Fl_Timer::direction() const [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

33.146.3.2 direction() [2/2]

```cpp
void Fl_Timer::direction(char d) [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

33.146.3.3 draw()

```cpp
void Fl_Timer::draw() [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:
```cpp
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw(); // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

33.146.3.4 handle()

```cpp
int Fl_Timer::handle(int event) [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| 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.

33.146.3.5 suspended()

```cpp
char Fl_Timer::suspended() const [inline]
```

Gets or sets whether the timer is suspended.

The documentation for this class was generated from the following files:

- Fl_Timer.H
- forms_timer.cxx

## Fl_Toggle_Button Class Reference

The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off.

```cpp
#include <Fl_Toggle_Button.H>
```

Inheritance diagram for Fl_Toggle_Button:

```
Fl_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

#### 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().

#### Constructor & Destructor Documentation

##### Fl_Toggle_Button()

```cpp
Fl_Toggle_Button::Fl_Toggle_Button {
    int X,
    int Y,
    int W,
    int H,
    const char *l = 0
}
```

Creates a new Fl_Toggle_Button widget using the given position, size, and label string.

The constructor creates the button using the given position, size, and label.

The inherited destructor deletes the toggle button.

The Button type() is set to FL_TOGGLE_BUTTON.

### Parameters

- **in X, Y, W, H** position and size of the widget
- **in L** widget label, default is no label

Generated by Doxygen
The documentation for this class was generated from the following files:

- Fl_Toggle_Button.H
- Fl_Button.cxx

### 33.148 Fl_Tooltip Class Reference

The Fl_Tooltip class provides tooltip support for all FLTK widgets.

```cpp
#include <Fl_Tooltip.H>
```

**Static Public Member Functions**

- static **Fl_Color** color ()
  
  Gets the background color for tooltips.

- static void color (**Fl_Color** c)
  
  Sets the background color for tooltips.

- static **Fl_Widget** *current ()
  
  Gets the current widget target.

- static void current (**Fl_Widget** *w)
  
  Sets the current widget target.

- static **Fl_Window** *current_window ()
  
  Returns the window that is used for tooltips.

- static **float** delay ()
  
  Gets the tooltip delay.

- static void delay (**float** f)
  
  Sets the tooltip delay.

- static void disable ()
  
  Same as enable(0), disables tooltips on all widgets.

- static void enable (**int** b=1)
  
  Enables tooltips on all widgets (or disables if b is false).

- static **int** enabled ()
  
  Returns non-zero if tooltips are enabled.

- static void enter_area (**Fl_Widget** *w, **int** X, **int** Y, **int** W, **int** H, **const char** *tip)
  
  You may be able to use this to provide tooltips for internal pieces of your widget.

- static **Fl_Font** font ()
  
  Gets the typeface for the tooltip text.

- static void font (**Fl_Font** i)
  
  Sets the typeface for the tooltip text.

- static **float** hidedelay ()
  
  Gets the time until an open tooltip hides again.

- static void hidedelay (**float** f)
  
  Sets the time until an open tooltip hides again.

- static **float** hoverdelay ()
  
  Gets the tooltip hover delay, the delay between tooltips.

- static void hoverdelay (**float** f)
  
  Sets the tooltip hover delay, the delay between tooltips.

- static **int** margin_height ()
  
  Gets the amount of extra space above and below the tooltip’s text.

- static void margin_height (**int** v)
  
  Sets the amount of extra space above and below the tooltip’s text.

- static **int** margin_width ()
  
  Gets the amount of extra space left/right of the tooltip’s text.
• static void margin_width (int v)
  * Sets the amount of extra space left/right of the tooltip's text.

• static Fl_Fontsize size ()
  * Gets the size of the tooltip text.

• static void size (Fl_Fontsize s)
  * Sets the size of the tooltip text.

• static Fl_Color textcolor ()
  * Gets the color of the text in the tooltip.

• static void textcolor (Fl_Color c)
  * Sets the color of the text in the tooltip.

• static int wrap_width ()
  * Gets the maximum width for tooltip's text before it word wraps.

• static void wrap_width (int v)
  * Sets the maximum width for tooltip's text before it word wraps.

Static Public Attributes

• static void (∗enter)(Fl_Widget ∗w) = nothing
• static void (∗exit)(Fl_Widget ∗w) = nothing

Friends

• class Fl_TooltipBox
• void Fl_Widget::copy_tooltip (const char ∗)
• void Fl_Widget::tooltip (const char ∗)

33.148.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets. It contains only static methods.

Figure 33.53 Fl_Tooltip Options

33.148.2 Member Function Documentation

33.148.2.1 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.
33.148.2.2 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.

33.148.2.3 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.

33.148.2.4 delay() [1/2]

static float Fl_Tooltip::delay ( ) [inline], [static]
Gets the tooltip delay.
The default delay is 1.0 seconds.

33.148.2.5 delay() [2/2]

static void Fl_Tooltip::delay (float f) [inline], [static]
Sets the tooltip delay.
The default delay is 1.0 seconds.

33.148.2.6 disable()

static void Fl_Tooltip::disable ( ) [inline], [static]
Same as enable(0), disables tooltips on all widgets.

33.148.2.7 enable()

static void Fl_Tooltip::enable (int b = 1) [inline], [static]
Enables tooltips on all widgets (or disables if b is false).

33.148.2.8 enabled()

static int Fl_Tooltip::enabled ( ) [inline], [static]
Returns non-zero if tooltips are enabled.

33.148.2.9 enter_area()

void Fl_Tooltip::enter_area (Fl_Widget * wid, int x, int y,
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).

### 33.148.2.10 font() [1/2]

```cpp
static Fl_Font Fl_Tooltip::font ( ) [inline], [static]
```

Gets the typeface for the tooltip text.

### 33.148.2.11 font() [2/2]

```cpp
static void Fl_Tooltip::font ( Fl_Font i ) [inline], [static]
```

Sets the typeface for the tooltip text.

### 33.148.2.12 hidedelay() [1/2]

```cpp
static float Fl_Tooltip::hidedelay ( ) [inline], [static]
```

Gets the time until an open tooltip hides again. The default delay is 12.0 seconds.

### 33.148.2.13 hidedelay() [2/2]

```cpp
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.

### 33.148.2.14 hoverdelay() [1/2]

```cpp
static float Fl_Tooltip::hoverdelay ( ) [inline], [static]
```

Gets the tooltip hover delay, the delay between tooltips. The default delay is 0.2 seconds.

### 33.148.2.15 hoverdelay() [2/2]

```cpp
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.

### 33.148.2.16 margin_height() [1/2]

```cpp
static int Fl_Tooltip::margin_height ( ) [inline], [static]
```

Gets the amount of extra space above and below the tooltip's text. Default is 3.
33.148.2.17 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.

33.148.2.18 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.

33.148.2.19 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.

33.148.2.20 size() [1/2]

static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]
Gets the size of the tooltip text.

33.148.2.21 size() [2/2]

static void Fl_Tooltip::size ( Fl_Fontsize s ) [inline], [static]
Sets the size of the tooltip text.

33.148.2.22 textcolor() [1/2]

static Fl_Color Fl_Tooltip::textcolor ( ) [inline], [static]
Gets the color of the text in the tooltip. The default is black.

33.148.2.23 textcolor() [2/2]

static void Fl_Tooltip::textcolor ( Fl_Color c ) [inline], [static]
Sets the color of the text in the tooltip. The default is black.

33.148.2.24 wrap_width() [1/2]

static int Fl_Tooltip::wrap_width ( ) [inline], [static]
Gets the maximum width for tooltip's text before it word wraps. Default is 400.

33.148.2.25 wrap_width() [2/2]

static void Fl_Tooltip::wrap_width ( int v ) [inline], [static]
Sets the maximum width for tooltip's text before it word wraps. Default is 400.
The documentation for this class was generated from the following files:

- Fl_Tooltip.H
- Fl.cxx
- Fl_Tooltip.cxx

33.149 Fl_Tree Class Reference

Tree widget.
#include <Fl_Tree.H>

Inheritance diagram for Fl_Tree:

```
<table>
<thead>
<tr>
<th>Fl_Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl_Group</td>
</tr>
<tr>
<td>Fl_Tree</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **Fl_Tree_Item * add (const char *path, Fl_Tree_Item *newitem=0)**
  Adds a new item, given a menu style 'path'.

- **Fl_Tree_Item * add (Fl_Tree_Item *parent_item, const char *name)**
  Add a new child item labeled 'name' to the specified 'parent_item'.

- **void calc_dimensions ()**
  Recalculate widget dimensions and scrollbar visibility, normally managed automatically.

- **void calc_tree ()**
  Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.

- **Fl_Tree_Item * callback_item ()**
  Gets the item that caused the callback.

- **void callback_item (Fl_Tree_Item *item)**
  Sets the item that was changed for this callback.

- **Fl_Tree_Reason callback_reason () const**
  Gets the reason for this callback.

- **void callback_reason (Fl_Tree_Reason reason)**
  Sets the reason for this callback.

- **void clear ()**
  Clear the entire tree's children, including the root.

- **void clear_children (Fl_Tree_Item *item)**
  Clear all the children for 'item'.

- **int close (const char *path, int docallback=1)**
  Closes the item specified by 'path'.

- **int close (Fl_Tree_Item *item, int docallback=1)**
  Closes the specified 'item'.

- **Fl_Image * closeicon () const**
  Returns the icon to be used as the 'close' icon.

- **void closeicon (Fl_Image *val)**
  Sets the icon to be used as the 'close' icon.

- **Fl_Color connectorcolor () const**
  Get the connector color used for tree connection lines.
- void **connectorcolor** (Fl_Color val)
  Set the connector color used for tree connection lines.

- **Fl_Tree_Connector** **connectorstyle** () const
  Returns the line drawing style for inter-connecting items.

- void **connectorstyle** (Fl_Tree_Connector val)
  Sets the line drawing style for inter-connecting items.

- int **connectorwidth** () const
  Gets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

- void **connectorwidth** (int val)
  Sets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.

- int **deselect** (const char *path, int docallback=1)
  Deselect an item specified by 'path'.

- int **deselect** (Fl_Tree_Item *item, int docallback=1)
  Deselect the specified item.

- int **deselect_all** (Fl_Tree_Item *item=0, int docallback=1)
  Deselect 'item' and all its children.

- void **display** (Fl_Tree_Item *item)
  Displays 'item', scrolling the tree as necessary.

- int **displayed** (Fl_Tree_Item *item)
  See if 'item' is currently displayed on-screen (visible within the widget).

- int **extend_selection** (Fl_Tree_Item *from, Fl_Tree_Item *to, int val=1, bool visible=false)
  Extend a selection between 'from' and 'to' depending on 'visible'.

- int **extend_selection_dir** (Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible)
  Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.

- Fl_Tree_Item * **find_clicked** (int yonly=0)
  Non-const version of Fl_Tree::find_clicked(int yonly) const.

- const Fl_Tree_Item * **find_clicked** (int yonly=0) const
  Find the item that was last clicked on.

- Fl_Tree_Item * **find_item** (const char *path)
  Non-const version of Fl_Tree::find_item(const char *path) const.

- const Fl_Tree_Item * **find_item** (const char *path) const
  Find the item, given a menu style path, e.g.

- Fl_Tree_Item * **first** ()
  Returns the first item in the tree, or 0 if none.

- Fl_Tree_Item * **first_selected_item** ()
  Returns the first selected item in the tree.

- Fl_Tree_Item * **first_visible** ()
  Returns the first open(), visible item in the tree, or 0 if none.

- Fl_Tree_Item * **first_visible_item** ()
  Returns the first open(), visible item in the tree, or 0 if none.

- Fl_Tree (int X, int Y, int W, int H, const char *L=0)
  Constructor.

- Fl_Tree_Item * **get_item_focus** () const
  Get the item that currently has keyboard focus.

- int **get_selected_items** (Fl_Tree_Item_Array &items)
  Returns the currently selected items as an array of 'ret_items'.

- int **handle** (int e) FL_OVERRIDE
  Standard FLTK event handler for this widget.

- int **hposition** () const
  Returns the horizontal scroll position as a pixel offset.
• void hposition (int pos)
  Sets the horizontal scroll offset to position `pos`.

• Fl_Tree_Item *insert (Fl_Tree_Item *item, const char *name, int pos)
  Insert a new item `name` into `item`'s children at position `pos`.

• Fl_Tree_Item *insert_above (Fl_Tree_Item *above, const char *name)
  Inserts a new item `name` above the specified Fl_Tree_Item `above`.

• 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) const
  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 integer `mode`.

• void item_labelbgcolor (Fl_Color val)
  Set the default label background color used for creating new items.

• Fl_Color item_labelbgcolor (void) const
  Get the default label background color used for creating new items.

• void item_labelfgcolor (Fl_Color val)
  Set the default label foreground color used for creating new items.

• Fl_Color item_labelfgcolor (void) const
  Get the default label foreground color used for creating new items.

• Fl_Font item_labelfont () const
  Get the default font face used for creating new items.

• void item_labelfont (Fl_Font val)
  Set the default font face used for creating new items.

• Fl_Fontsize item_labelsize () const
  Get the default label fontsize used for creating new items.

• void item_labelsize (Fl_Fontsize val)
  Set the default label font size used for creating new items.

• int item_pathname (char *pathname, int pathnamelen, const Fl_Tree_Item *item) const
  Return `pathname` of size `pathnamelen` for the specified `item`.

• Fl_Tree_Item_Reselect_Mode item_reselect_mode () const

Generated by Doxygen
Returns the current item re/selection mode.

- void item_reselect_mode (Fl_Tree_Item_Reselect_Mode mode)
  
  Sets the item re/selection mode.

- int labelmarginleft () const
  
  Get the amount of white space (in pixels) that should appear to the left of the label text.

- void labelmarginleft (int val)
  
  Set the amount of white space (in pixels) that should appear to the left of the label text.

- FL_Tree_Item * last ()
  
  Returns the last item in the tree.

- FL_Tree_Item * last_selected_item ()
  
  Returns the last selected item in the tree.

- FL_Tree_Item * last_visible ()
  
  Returns the last open(), visible item in the tree.

- FL_Tree_Item * last_visible_item ()
  
  Returns the last open(), visible item in the tree.

- int linespacing () const
  
  Get the amount of white space (in pixels) that should appear between items in the tree.

- void linespacing (int val)
  
  Sets the amount of white space (in pixels) that should appear between items in the tree.

- void load (class Fl_Preferences &)
  
  Load FLTK preferences.

- int marginbottom () const
  
  Get the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

- void marginbottom (int val)
  
  Sets the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.

- int marginleft () const
  
  Get the amount of white space (in pixels) that should appear between the widget’s left border and the tree’s contents.

- void marginleft (int val)
  
  Set the amount of white space (in pixels) that should appear between the widget’s left border and the left side of the tree’s contents.

- int margintop () const
  
  Get the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

- void margintop (int val)
  
  Sets the amount of white space (in pixels) that should appear between the widget’s top border and the top of the tree’s contents.

- FL_Tree_Item * next (FL_Tree_Item *item=0)
  
  Return the next item after ‘item’, or 0 if no more items.

- FL_Tree_Item * next_item (FL_Tree_Item *item, int dir=FL_Down, bool visible=false)
  
  Returns next item after ‘item’ in direction ‘dir’ depending on ‘visible’.

- FL_Tree_Item * next_selected_item (FL_Tree_Item *item=0, int dir=FL_Down)
  
  Returns the next selected item above or below ‘item’, depending on ‘dir’.

- FL_Tree_Item * next_visible_item (FL_Tree_Item *start, int dir)
  
  Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified ‘item’, or 0 if no more items.

- int open (const char *path, int docallback=1)
  
  Opens the item specified by ‘path’.

- int open (FL_Tree_Item *item, int docallback=1)
  
  Open the specified ‘item’.

- void open_toggle (FL_Tree_Item *item, int docallback=1)
Toggle the open state of 'item'.

- int openchild_marginbottom () const
  Get the amount of white space (in pixels) that should appear below an open child tree's contents.

- void openchild_marginbottom (int val)
  Set the amount of white space (in pixels) that should appear below an open child tree's contents.

- FL Image * openicon () const
  Returns the icon to be used as the 'open' icon.

- void openicon (FL Image *val)
  Sets the icon to be used as the 'open' icon.

- const Fl Tree_Prefs & prefs () const

- FL Tree_Item * prev (FL Tree_Item *item=0)
  Return the previous item before 'item', or 0 if no more items.

- void recalc_tree ()
  Schedule tree to recalc the entire tree size.

- int remove (FL Tree_Item *item)
  Remove the specified 'item' from the tree.

- void resize (int, int, int, int) FL_OVERRIDE
  Resizes the Fl Group widget and all of its children.

- FL Tree_Item * root ()
  Returns the root item.

- void root (FL Tree_Item *newitem)
  Sets the root item to 'newitem'.

- void root_label (const char *new_label)
  Set the label for the root item to 'new_label'.

- int scrollbar_size () const
  Gets the default size of scrollbars' troughs for this widget in pixels.

- void scrollbar_size (int size)
  Sets the pixel size of the scrollbars' troughs to 'size' for this widget, in pixels.

- int select (const char *path, int docallback=1)
  Select the item specified by 'path'.

- int select (FL Tree_Item *item, int docallback=1)
  Select the specified 'item'.

- int select_all (FL Tree_Item *item=0, int docallback=1)
  Select 'item' and all its children.

- int select_only (FL Tree_Item *selitem, int docallback=1)
  Select only the specified item, deselecting all others that might be selected.

- void select_toggle (FL Tree_Item *item, int docallback=1)
  Toggle the select state of the specified 'item'.

- FL Boxtype selectbox () const
  Sets the style of box used to draw selected items.

- void selectbox (FL Boxtype val)
  Gets the style of box used to draw selected items.

- FL Tree_Select selectmode () const
  Gets the tree's current selection mode.

- void selectmode (FL Tree_Select val)
  Sets the tree's selection mode.

- void set_item_focus (FL Tree_Item *item)
  Set the item that currently should have keyboard focus.

- void show_item (FL Tree_Item *item)
  Adjust the vertical scrollbar to show 'item' at the top of the display IF it is currently off-screen (for instance show_item_top()).
• void show_item (Fl_Tree_Item *item, int yoff)
  Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the Fl_Tree widget's display.

• void show_item_bottom (Fl_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

• void show_item_middle (Fl_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is in the middle of the display.

• void show_item_top (Fl_Tree_Item *item)
  Adjust the vertical scrollbar so that 'item' is at the top of the display.

• void show_self ()
  Print the tree as 'ascii art' to stdout.

• int showcollapse () const
  Returns 1 if the collapse icon is enabled, 0 if not.

• void showcollapse (int val)
  Set if we should show the collapse icon or not.

• int showroot () const
  Returns 1 if the root item is to be shown, or 0 if not.

• void showroot (int val)
  Set if the root item should be shown or not.

• Fl_Tree_Sort sortorder () const
  Set the default sort order used when items are added to the tree.

• void sortorder (Fl_Tree_Sort val)
  Gets the sort order used to add items to the tree.

• Fl_Image * usericon () const
  Returns the Fl_Image being used as the default user icon for all newly created items.

• void usericon (Fl_Image *val)
  Sets the Fl_Image to be used as the default user icon for all newly created items.

• int usericonmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the usericon.

• void usericonmarginleft (int val)
  Set the amount of white space (in pixels) that should appear to the left of the usericon.

• int vposition () const
  Returns the vertical scroll position as a pixel offset.

• void vposition (int pos)
  Sets the vertical scroll offset to position 'pos'.

• int widgetmarginleft () const
  Get the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

• void widgetmarginleft (int val)
  Set the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

• ~Fl_Tree ()
  Destructor.

Protected Member Functions

• void do_callback_for_item (Fl_Tree_Item *item, Fl_Tree_Reason reason)
  Do the callback for the specified 'item' using 'reason', setting the callback_item() and callback_reason().

• void draw () FL_OVERRIDE
  Standard FLTK draw() method, handles drawing the tree widget.

• void item_clicked (Fl_Tree_Item *val)
  Set the item that was last clicked.
Protected 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

33.149.1 Detailed Description

Tree widget.

![Figure 33.54 Fl_Tree example program](image)

Similar to `Fl_Browser`, `Fl_Tree` is a browser of `Fl_Tree_Item's arranged in a parented hierarchy, or 'tree'. Subtrees can be expanded or closed. Items can be added, deleted, inserted, sorted and re-ordered.

The tree items may also contain other FLTK widgets, like buttons, input fields, or even "custom" widgets.

The `callback()` is invoked depending on the value of `when()`:

- `FL_WHEN_RELEASE` – callback invoked when left mouse button is released on an item
• FL_WHEN_CHANGED – callback invoked when left mouse changes selection state

The simple way to define a tree:

```c
#include <FL/Fl_Tree.H>
[..]
Fl_Tree tree(X,Y,W,H);
  tree.begin();
  tree.add("Flintstones/Fred");
  tree.add("Flintstones/Wilma");
  tree.add("Flintstones/Pebbles");
  tree.add("Simpsons/Homer");
  tree.add("Simpsons/Marge");
  tree.add("Simpsons/Bart");
  tree.add("Simpsons/Lisa");
  tree.end();
```

**FEATURES**

Items can be added with `add()`, removed with `remove()`, completely cleared with `clear()`, inserted with `insert()` and `insert_above()`, selected/deselected with `select()` and `deselect()`, open/closed with `open()` and `close()`, positioned on the screen with `show_item_top()`, `show_item_middle()` and `show_item_bottom()`, item children can be swapped around with `Fl_Tree_Item::swap_children()`, items can be moved around with `Fl_Tree_Item::move()`, 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 added 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()`.

Generated by Doxygen
Various default preferences can be globally manipulated via \texttt{Fl_Tree_Prefs}, including colors, margins, icons, connection lines, etc.

**FONTS AND COLORS**

When adding new items to the tree, the new items get the defaults for fonts and colors from:

- \texttt{Fl_Tree::item_labelfont()} – The default item label font (default: FL_HELVETICA)
- \texttt{Fl_Tree::item_labelsize()} – The default item label size (default: FL_NORMAL_SIZE)
- \texttt{Fl_Tree::item_labelfgcolor()} – The default item label foreground color (default: FL_FOREGROUND_COLOR)
- \texttt{Fl_Tree::item_labelbgcolor()} – The default item label background color (default: 0xffffffff, which tree uses as 'transparent')

Each item (\texttt{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:

- \texttt{Fl_Tree_Item::labelfont()} – The item’s label font (default: FL_HELVETICA)
- \texttt{Fl_Tree_Item::labelsize()} – The item’s label size (default: FL_NORMAL_SIZE)
- \texttt{Fl_Tree_Item::labelfgcolor()} – The item’s label foreground color (default: FL_FOREGROUND_COLOR)
- \texttt{Fl_Tree_Item::labelbgcolor()} – The item’s label background color (default: 0xffffffff, which uses the tree’s own bg color)

**CALLBACKS**

The tree’s \texttt{callback()} will be invoked when items change state or are open/closed. \texttt{when()} controls when mouse/keyboard events invoke the callback. \texttt{callback_item()} and \texttt{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 \texttt{Fl_Tree::item_pathname()}, e.g.
char pathname[256] = "???";
tree->item_pathname(pathname, sizeof(pathname), item); // eg. "Parent/Child/Item"

To walk all the items of the tree from top to bottom:

// Walk all the items in the tree, and print their labels
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    printf("Item: %s\n", item->label());
}

To recursively walk all the children of a particular item, define a function that uses recursion:

// Find all of the item's children and print an indented report of their labels
void my_print_all_children(Fl_Tree_Item *item, int indent=0) {
    for ( int t=0; t<item->children(); t++ ) {
        printf("%s Item: %s\n", indent, "", item->child(t)->label());
        my_print_all_children(item->child(t), indent+4); // recurse
    }
}

To change the default label font and color when creating new items:

tree = new Fl_Tree(...);
tree->item_labelfont(FL_COURIER); // Use Courier font for all new items
tree->item_labelfgcolor(FL_RED); // Use red color for labels of all new items

// Now create the items in the tree using the above defaults.
tree->add("Aaa");
tree->add("Bbb");

To change the font and color of all existing items in the tree:

// Change the font and color of all items currently in the tree
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    item->labelfont(FL_COURIER);
    item->labelcolor(FL_RED);
}

DISPLAY DESCRIPTION

The following image shows the tree's various visual elements and the methods that control them:

![Figure 33.55 Fl_Tree elements]
The following shows the protected dimension variables 'tree inner' (tix..) and 'tree outer' (tox..):

![Figure 33.56 Fl_Tree inner/outer dimensions](image)

**KEYBOARD BINDINGS**

The following table lists keyboard bindings for navigating the tree:

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>FL_TREE_SELECT _MULTI</th>
<th>FL_TREE_SELECT _SINGLE</th>
<th>FL_TREE_SELECT _NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-A (Linux/Windows)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Command-A (Mac)</td>
<td>Select all items</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Space</td>
<td>Selects item</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Ctrl-Space</td>
<td>Toggle item</td>
<td>Toggle item</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Space</td>
<td>Extends selection</td>
<td>Selects item</td>
<td>N/A</td>
</tr>
<tr>
<td>Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Ctrl-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Shift-Enter</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
<td>Toggles open/close</td>
</tr>
<tr>
<td>Right / Left</td>
<td>Open/Close item</td>
<td>Open/Close item</td>
<td>Open/Close item</td>
</tr>
<tr>
<td>Up / Down</td>
<td>Move focus box up/down</td>
<td>Move focus box up/down</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift-Up / Shift-Down</td>
<td>Extend selection</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>

### 33.149.2 Member Function Documentation

#### 33.149.2.1 add() [1/2]

```cpp
Fl_Tree_Item & Fl_Tree::add (  
    const char * path,
```
Fl_Tree_Item * item = 0;

Adds a new item, given a menu style 'path'.
Any parent nodes that don't already exist are created automatically. Adds the item based on the value of sortorder().
If 'item' is NULL, a new item is created.
To specify items or submenus that contain slashes ('/' or '\') use an escape character to protect them, e.g.

```
':
tree->add("/Holidays/Photos/12\25\2010"); // Adds item "12/25/2010"
tree->add("/Pathnames/c:\Program Files\MyApp"); // Adds item "c:\Program Files\MyApp"
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>The path to the item, e.g. &quot;Flintstone/Fred&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>item</td>
<td>The new item to be added. If NULL, a new item is created with a name that is the last element in 'path'.</td>
</tr>
</tbody>
</table>

Returns

The new item added, or 0 on error.

Version

1.3.3

33.149.2.2  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

<table>
<thead>
<tr>
<th>in</th>
<th>parent_item</th>
<th>The parent item the new child item will be added to. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>The label for the new item</td>
</tr>
</tbody>
</table>

Returns

The new item added.

Version

1.3.0 release

33.149.2.3  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 resized or if the scrollbar's sizes are changed (affects tree widget's inner dimensions tix/y/w/h), and also used by calc_tree().

Generated by Doxygen
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)

### 33.149.2.5 callback_item() [1/2]

```cpp
Fl_Tree_Item * Fl_Tree::callback_item ( )
```

Gets the item that caused the callback.

The callback() can use this value to see which item changed.

### 33.149.2.6 callback_item() [2/2]

```cpp
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.

### 33.149.2.7 callback_reason() [1/2]

```cpp
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:

```cpp
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() ) {  
    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..  
  }
}
```
33.149.2.8 callback_reason() [2/2]

```cpp
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.

33.149.2.9 clear()

```cpp
void Fl_Tree::clear ( )
```

Clear the entire tree's children, including the root. The tree will be left completely empty.

33.149.2.10 clear_children()

```cpp
void Fl_Tree::clear_children (  
    Fl_Tree_Item * item )
```

Clear all the children for 'item'. Item may not be NULL.

33.149.2.11 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 <code>callback()</code> is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - <code>callback()</code> is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - <code>callback()</code> is invoked if item changed (default), <code>callback_reason()</code> will be <code>FL_TREE_REASON_CLOSED</code></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

`item_reselect_mode()` – enables `FL_TREE_REASON_RESELECTED` events
33.149.2.12 close() [2/2]

```c
int Fl_Tree::close (  
    Fl_Tree_Item ∗ item,  
    int docallback = 1 )
```

Closes the specified 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

### Parameters

| in | item  | -- the item to be closed. Must not be NULL. |
| in | docallback | -- A flag that determines if the callback() is invoked or not: |
|    |         | • 0 - callback() is not invoked |
|    |         | • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED |

### Returns

- 1 – item was closed
- 0 – item was already closed, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

33.149.2.13 closeicon() [1/2]

```c
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.

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

33.149.2.14 closeicon() [2/2]

```c
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. |

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()
33.149.2.15 connectorstyle()

```c
void Fl_Tree::connectorstyle ( 
    Fl_Tree_Connector val )
```

Sets the line drawing style for inter-connecting items.
See Fl_Tree_Connector for possible values.

33.149.2.16 deselect() [1/2]

```c
int Fl_Tree::deselect ( 
    const char * path, 
    int docallback = 1 )
```

Deselect an item specified by 'path'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. deselect("← Holidays/12/25/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>– the tree item’s pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

- Returns
  - 1 - OK: item’s state was changed
  - 0 - OK: item was already deselected, no change was made
  - -1 - ERROR: item was not found

33.149.2.17 deselect() [2/2]

```c
int Fl_Tree::deselect ( 
    Fl_Tree_Item * item, 
    int docallback = 1 )
```

Deselect the specified item.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>– the item to be deselected. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>– A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>
33.149.2.18  **deselect_all()**

```cpp
int Fl_Tree::deselect_all (  
    Fl_Tree_Item * item = 0,  
    int docallback = 1  
)
```

Deselect `item` and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter `docallback`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The item that will be deselected (along with all its children). If NULL, first() is used.</td>
</tr>
</tbody>
</table>
| 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.

33.149.2.19  **display()**

```cpp
void Fl_Tree::display (  
    Fl_Tree_Item * item  
)
```

Displays `item`, scrolling the tree as necessary.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The item to be displayed. If NULL, first() is used.</td>
</tr>
</tbody>
</table>

33.149.2.20  **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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The item to be checked. If NULL, first() is used.</td>
</tr>
</tbody>
</table>

Returns

- 0 - item was already deselected, no change was made
- 1 - item's state was changed
Returns

1 if displayed, 0 if scrolled off screen or no items are in tree.

### 33.149.2.21 draw()

```cpp
void Fl_Tree::draw ( ) [protected], [virtual]
```

Standard FLTK `draw()` method, handles drawing the tree widget.
Reimplemented from `Fl_Group`.

### 33.149.2.22 extend_selection()

```cpp
int Fl_Tree::extend_selection ( Fl_Tree_Item * from, Fl_Tree_Item * to, int val = 1, bool visible = false )
```

Extend a selection between 'from' and 'to' depending on 'visible'.
Similar to the more efficient `extend_selection_dir(Fl_Tree_Item*,Fl_Tree_Item*,int,dirt,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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from</code></td>
<td>Starting item</td>
</tr>
<tr>
<td><code>to</code></td>
<td>Ending item</td>
</tr>
<tr>
<td><code>val</code></td>
<td>Select or deselect items (0=deselect, 1=select, 2=toggle)</td>
</tr>
<tr>
<td><code>visible</code></td>
<td>true=affect only <code>open()</code>, visible items, false=affect open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

The number of items whose selection states were changed, if any.

#### Version

1.3.3 ABI feature

### 33.149.2.23 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

- **in from** Starting item
- **in to** Ending item
- **in dir** Direction to extend selection (FL_Up or FL_Down)
- **in val** 0=deselct, 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

#### 33.149.2.24 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

#### 33.149.2.25 find_item()

```cpp
const Fl_Tree_Item * Fl_Tree::find_item ( const char * path ) const
```

Find the item, given a menu style path, e.g. 
"/Parent/Child/item". There is both a const and non-const version of this method. Const version allows pure const methods to use this method to do lookups without causing compiler errors.
To specify items or submenus that contain slashes ('/' or '\') use an escape character to protect them, e.g.
tree->add("/Holidays/Photos/12\125\2010"); // Adds item "12/25/2010"

Parameters

`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()`

### 33.149.2.26 first()

`Fl_Tree_Item * Fl_Tree::first ( )`

Returns the first item in the tree, or 0 if none.

Use this to walk the tree in the forward direction, e.g.

```c
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) )
    printf("Item: %s\n", item->label());
```

Returns

First item in tree, or 0 if none (tree empty).

See also

`first(), next(), last(), prev()`

### 33.149.2.27 first_selected_item()

`Fl_Tree_Item * Fl_Tree::first_selected_item ( )`

Returns the first selected item in the tree.

Use this to walk the tree from top to bottom looking for all the selected items, e.g.

```c
// Walk tree forward, from top to bottom
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i) )
    printf("Selected item: %s\n", i->label());
```

Returns

The first selected item, or 0 if none.

See also

`first_selected_item(), last_selected_item(), next_selected_item()`
33.149.2.28 first_visible()

```
Fl_Tree_Item * Fl_Tree::first_visible ( )
```

Returns the first open(), visible item in the tree, or 0 if none.

**Deprecated** in 1.3.3 ABI – use `first_visible_item()` instead.

33.149.2.29 first_visible_item()

```
Fl_Tree_Item * Fl_Tree::first_visible_item ( )
```

Returns the first open(), visible item in the tree, or 0 if none.

Returns

First visible item in tree, or 0 if none.

See also

`first_visible_item()`, `last_visible_item()`, `next_visible_item()`

Version

1.3.3

33.149.2.30 get_selected_items()

```
int Fl_Tree::get_selected_items ( Fl_Tree_Item_Array & ret_items )
```

Returns the currently selected items as an array of `ret_items`.

Example:

```
// Get selected items as an array
Fl_Tree_Item_Array items;
tree->get_selected_items(items);
// Manipulate the returned array
for ( int t=0; t<items.total(); t++ ) { 
    Fl_Tree_Item item = items[t];
    ..do stuff with each selected item..
}
```

Parameters

| out | ret_items | The returned array of selected items. |

Returns

The number of items in the returned array.

See also

`first_selected_item()`, `next_selected_item()`

Version

1.3.3 ABI feature

Generated by Doxygen
33.149.2.31 handle()

```cpp
int Fl_Tree::handle ( int e ) [virtual]
```

Standard FLTK event handler for this widget.

**Todo** add `Fl_Widget_Tracker` (see `Fl_Browser_.cxx::handle()`)

Reimplemented from `Fl_Group`.

33.149.2.32 hposition() [1/2]

```cpp
int Fl_Tree::hposition ( ) const
```

Returns the horizontal scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the left edge of the screen.

See also

- `hposition(int)`, `vposition()`, `vposition(int)`

**Note**

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

33.149.2.33 hposition() [2/2]

```cpp
void Fl_Tree::hposition ( int pos )
```

Sets the horizontal scroll offset to position `pos`.
The position is how many pixels of the tree are scrolled off the left edge of the screen.

**Parameters**

| in | pos | The vertical position (in pixels) to scroll the tree to. |

See also

- `hposition()`, `vposition()`, `vposition(int)`

**Note**

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

33.149.2.34 insert()

```cpp
Fl_Tree_Item * Fl_Tree::insert ( Fl_Tree_Item * item, const char * name, int pos )
```

Insert a new item `name` into `item`'s children at position `pos`.
If `pos` is out of range the new item is

- prepended if `pos < 0` or
- appended if `pos > item->children()`.  

**Note:** `pos == children()` is not considered out of range: the item is appended to the child list.

**Example:**
tree->add("Aaa/000"); // "000" is index 0 in Aaa’s children
// How to use insert() to insert a new item between Aaa/111 + Aaa/222
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()

33.149.2.35 insert_above()

Fl_Tree_Item * Fl_Tree::insert_above (  
   Fl_Tree_Item * above,  
   const char * name )

Inserts a new item ‘name’ above the specified Fl_Tree_Item ‘above’.

Example:

tree->add("Aaa/000"); // "000" is index 0 in Aaa’s children
// How to use insert_above() to insert a new item above Aaa/222
if (item) tree->insert_above(item, "New item"); // insert new item above it

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>above</th>
<th>– the item above which to insert the new item. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>– the name of the new item</td>
</tr>
</tbody>
</table>

Returns

The new item added, or 0 if ‘above’ could not be found.

See also

insert()

33.149.2.36 is_close() [1/2]

int Fl_Tree::is_close (  
   const char * path ) const
See if item specified by 'path' is closed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_close("Holidays/12/25/2010").

Parameters

| in | path | the tree item's pathname (e.g. "Flintstones/Fred") |

Returns

- 1: OK: item is closed
- 0: OK: item is open
- -1: ERROR: item was not found

33.149.2.37  is_close() [2/2]

int Fl_Tree::is_close (Fl_Tree_Item *item) const
See if the specified 'item' is closed.

Parameters

| in | item | the item to be tested. Must not be NULL. |

Returns

- 1: item is closed
- 0: item is open

33.149.2.38  is_hscroll_visible()

int Fl_Tree::is_hscroll_visible ( ) const
See if the horizontal scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

33.149.2.39  is_open() [1/2]

int Fl_Tree::is_open (const char *path ) const
See if item specified by 'path' is open.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_open("Holidays/12/25/2010").
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

| in | path | the tree item's pathname (e.g. "Flintstones/Fred") |
Returns
- 1 - OK: item is open
- 0 - OK: item is closed
- -1 - ERROR: item was not found

See also
Fl_Tree_Item::visible_r()

### 33.149.2.40 is_open() [2/2]

```c
int Fl_Tree::is_open ( Fl_Tree_Item * item ) const
```

See if 'item' is open.
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

#### Parameters
- `in item` - the item to be tested. Must not be NULL.

#### Returns
- 1 : item is open
- 0 : item is closed

### 33.149.2.41 is_scrollbar()

```c
int Fl_Tree::is_scrollbar ( Fl_Widget * w )
```

See if widget 'w' is one of the Fl_Tree widget's scrollbars.
Use this to skip over the scrollbars when walking the child() array. Example:

```c
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

### 33.149.2.42 is_selected() [1/2]

```c
int Fl_Tree::is_selected ( const char * path )
```

Generated by Doxygen
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

33.149.2.43  is_selected() [2/2]

int Fl_Tree::is_selected ( Fl_Tree_Item * item ) const

See if the specified 'item' is selected.

Parameters

| in | item | the item to be tested. Must not be NULL. |

Returns

- 1 : item selected
- 0 : item deselected

33.149.2.44  is_vscroll_visible()

int Fl_Tree::is_vscroll_visible ( ) const

See if the vertical scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

33.149.2.45  item_clicked() [1/2]

Fl_Tree_Item * Fl_Tree::item_clicked ( )

Return the item that was last clicked.

Valid only from within the callback().

Returns

The item clicked, or 0 if none. 0 may also be used to indicate several items were clicked/changed.

Deprecated in 1.3.3 ABI – use callback_item() instead.
33.149.2.46 item_clicked() [2/2]

    void Fl_Tree::item_clicked (  
        Fl_Tree_Item * item  ) [protected]  

Set the item that was last clicked.  
Should only be used by subclasses needing to change this value. Normally Fl_Tree manages this value.  

**Deprecated** in 1.3.3 ABI – use callback_item() instead.

33.149.2.47 item_draw_mode() [1/3]

    Fl_Tree_Item_Draw_Mode Fl_Tree::item_draw_mode ( ) const  

Get the 'item draw mode' used for the tree.

**Version**  
1.3.1 ABI feature

33.149.2.48 item_draw_mode() [2/3]

    void Fl_Tree::item_draw_mode (  
        Fl_Tree_Item_Draw_Mode mode  )  

Set the 'item draw mode' used for the tree to 'mode'.  
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

**Version**  
1.3.1 ABI feature

33.149.2.49 item_draw_mode() [3/3]

    void Fl_Tree::item_draw_mode (  
        int mode  )  

Set the 'item draw mode' used for the tree to integer 'mode'.  
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

**Version**  
1.3.1 ABI feature

33.149.2.50 item_labelbgcolor() [1/2]

    void Fl_Tree::item_labelbgcolor (  
        Fl_Color val  )  

Set the default label background color used for creating new items.  
A special case is made for color 0xffffffff (default) which is treated as 'transparent'. To change the background color on a per-item basis, use Fl_Tree_Item::labelbgcolor(Fl_Color)

33.149.2.51 item_labelbgcolor() [2/2]

    Fl_Color Fl_Tree::item_labelbgcolor (  
        void ) const  

Get the default label background color used for creating new items.  
If the color is 0xffffffff, it is 'transparent'.

Generated by Doxygen
### item_labelfgcolor()

```c
void Fl_Tree::item_labelfgcolor (  
    Fl_Color val  
)
```

Set the default label foreground color used for creating new items.
To change the foreground color on a per-item basis, use `Fl_Tree_Item::labelfgcolor(Fl_Color)`.

### item_labelfont()

```c
void Fl_Tree::item_labelfont (  
    Fl_Font val  
)
```

Set the default font face used for creating new items.
To change the font face on a per-item basis, use `Fl_Tree_Item::labelfont(Fl_Font)`.

### item_labelsize()

```c
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)`.

### item_pathname()

```c
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="")

**See also**

`find_item()`

### item_reselect_mode()

```c
Fl_Tree_Item_Reselect_Mode Fl_Tree::item_reselect_mode ( ) const
```

Returns the current item re/selection mode.

**Version**

1.3.1 ABI feature
item_reselect_mode() [2/2]

```cpp
void Fl_Tree::item_reselect_mode(
    Fl_Tree_Item_Reselect_Mode mode)
```

Sets the item re/selection mode.
See `Fl_Tree_Item_Reselect_Mode` for possible values.

Version
1.3.1 ABI feature

last()

```cpp
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.

```cpp
for ( Fl_Tree_Item *item = tree->last(); item; item = tree->prev() )
    printf("Item: %s\n", item->label());
```

Returns
Last item in the tree, or 0 if none (tree empty).

See also
first(), next(), last(), prev()

last_selected_item()

```cpp
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.

```cpp
// Walk tree in reverse, from bottom to top
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
    printf("Selected item: %s\n", i->label());
```

Returns
The last selected item, or 0 if none.

See also
first_selected_item(), last_selected_item(), next_selected_item()

Version
1.3.3

last_visible()

```cpp
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.
33.149.2.61 last_visible_item()

Fl_Tree_Item * Fl_Tree::last_visible_item ( )

Returns the last open(), visible item in the tree.

Returns
   Last visible item in the tree, or 0 if none.

See also
   first_visible_item(), last_visible_item(), next_visible_item() 

Version
   1.3.3

33.149.2.62 load()

void Fl_Tree::load ( 
   class Fl_Preferences & prefs )

Load FLTK preferences.
Read a preferences database into the tree widget.
A preferences database is a hierarchical collection of data which can be directly loaded into the tree view for inspection.

Parameters
   in  prefs  the Fl_Preferences database

33.149.2.63 next()

Fl_Tree_Item * Fl_Tree::next ( 
   Fl_Tree_Item * item = 0 )

Return the next item after 'item', or 0 if no more items.
Use this code to walk the entire tree:

::
   for ( Fl_Tree_Item *i = tree->first(); i; i = tree->next(i) )
      printf("Item: %s\n", i->label());
   :

Parameters
   in  item  The item to use to find the next item. If NULL, returns 0.

Returns
   Next item in tree, or 0 if at last item.

See also
   first(), next(), last(), prev()
33.149.2.64 next_item()

Fl_Tree_Item * Fl_Tree::next_item (  
    Fl_Tree_Item * item,  
    int dir = FL_Down,  
    bool visible = false  
)  

Returns next item after 'item' in direction 'dir' depending on 'visible'.
Next item will be above (if dir==FL_Up) or below (if dir==FL_Down). If 'visible' is true, only items whose parents are open() will be returned. If 'visible' is false, even items whose parents are close()ed will be returned.
If item is 0, the return value will be the result of this truth table:

<table>
<thead>
<tr>
<th>visible=true</th>
<th>visible=false</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir=FL_Up:</td>
<td>last_visible_item()</td>
</tr>
<tr>
<td>dir=FL_Down:</td>
<td>first_visible_item()</td>
</tr>
</tbody>
</table>

Example use:

```c
// Walk down the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_item(i, FL_Down, true) )
    printf("Item: %s
", 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
", 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
", 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
", i->label());
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item to use to find the next item. If NULL, returns 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>Can be FL_Up or FL_Down (default=FL_Down or 'next')</td>
</tr>
<tr>
<td>in</td>
<td>visible</td>
<td>true=return only open(), visible items,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false=return open or closed items (default)</td>
</tr>
</tbody>
</table>

Returns

Next item in tree in the direction and visibility specified, or 0 if no more items of specified visibility in that direction.

See also

first(), last(), next(),
first_visible_item(), last_visible_item(), next_visible_item(),
first_selected_item(), last_selected_item(), next_selected_item()  

Version

1.3.3

33.149.2.65 next_selected_item()

Fl_Tree_Item * Fl_Tree::next_selected_item (  
    Fl_Tree_Item * item = 0,  
    int dir = FL_Down  
)  

Returns the next selected item above or below 'item', depending on 'dir'.

Generated by Doxygen
If 'item' is 0, search starts at either first() or last(), depending on 'dir': first() if 'dir' is FL_Down (default), last() if 'dir' is FL_Up.

Use this to walk the tree looking for all the selected items, e.g.

```c
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i, FL_Down) )
  printf("Item: %s\n", i->label());
// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
  printf("Item: %s\n", i->label());
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item above or below which we’ll find the next selected item. If NULL, first() is used if FL_Down, last() if FL_Up. (default=NULL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default)</td>
</tr>
</tbody>
</table>

Returns

The next selected item, or 0 if there are no more selected items.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

33.149.2.66 next_visible_item()

Fl_Tree_Item * Fl_Tree::next_visible_item (          
  Fl_Tree_Item * item, 
  int dir )

Returns next open(), visible item above (dir==FL_Up) or below (dir==FL_Down) the specified 'item', or 0 if no more items.

If 'item' is 0, returns last() if 'dir' is FL_Up, or first() if dir is FL_Down.

```c
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_visible_item(i, FL_Down) )
  printf("Item: %s\n", i->label());
// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_visible_item(i, FL_Up) )
  printf("Item: %s\n", i->label());
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>The item above/below which we’ll find the next visible item</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dir</td>
<td>The direction to search. Can be FL_Up or FL_Down.</td>
</tr>
</tbody>
</table>

Returns

The item found, or 0 if there’s no visible items above/below the specified item.
Version
1.3.3

33.149.2.67  open() [1/2]

int Fl_Tree::open (const char *path, int docallback = 1)

Opens the item specified by 'path'.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. open("Holidays/12\25\2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>— the tree item's pathname (e.g. &quot;Flintstones/Fred&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>— A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>

Returns

• 1 – OK: item opened
• 0 – OK: item was already open, no change
• -1 – ERROR: item was not found

See also
open(), close(), is_open(), is_close(), callback_item(), callback_reason()

33.149.2.68  open() [2/2]

int Fl_Tree::open (Fl_Tree_Item *item, int docallback = 1)

Open the specified 'item'.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>item</th>
<th>— the item to be opened. Must not be NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>docallback</td>
<td>— A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

- 1 – item was opened
- 0 – item was already open, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

### 33.149.2.69 open_toggle()

```cpp
void Fl_Tree::open_toggle (Fl_Tree_Item *item, int docallback = 1)
```

Toggles the open state of 'item'.

Invokes the callback depending on the value of optional parameter 'docallback'. Handles calling `redraw()` if anything changed.

The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>The item whose open state is to be toggled. Must not be NULL.</td>
</tr>
<tr>
<td><code>docallback</code></td>
<td>A flag that determines if the callback() is invoked or not:</td>
</tr>
<tr>
<td></td>
<td>• 0 - callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>• 1 - callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED</td>
</tr>
</tbody>
</table>

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

### 33.149.2.70 openicon() [1/2]

```cpp
Fl_Image *Fl_Tree::openicon () const
```

Returns the icon to be used as the 'open' icon.

If none was set, the internal default is returned, a simple '[+]' icon.

### 33.149.2.71 openicon() [2/2]

```cpp
void Fl_Tree::openicon (Fl_Image *val)
```

Sets the icon to be used as the 'open' icon.

This overrides the built in default '[+]' icon.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>val</code></td>
<td>The new image, or zero to use the default [+ icon].</td>
</tr>
</tbody>
</table>
33.149.2.72 prev()

```cpp
Fl_Tree_Item * Fl_Tree::prev (Fl_Tree_Item * item = 0)
```

Return the previous item before 'item', or 0 if no more items. This can be used to walk the tree in reverse, e.g.:

```c
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->prev(item) )
    printf("Item: %s\n", item->label());
```

**Parameters**

| in  | item | The item to use to find the previous item. If NULL, returns 0. |

**Returns**

Previous item in tree, or 0 if at first item.

**See also**

first(), next(), last(), prev()

---

33.149.2.73 recalc_tree()

```cpp
void Fl_Tree::recalc_tree ( )
```

Schedule tree to recalc the entire tree size.

**Note**

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

---

33.149.2.74 remove()

```cpp
int Fl_Tree::remove (Fl_Tree_Item * item)
```

Remove the specified 'item' from the tree. *item* may not be NULL. If it has children, all those are removed too. If item being removed has focus, no item will have focus.

**Returns**

0 if done, -1 if 'item' not found.

---

33.149.2.75 resize()

```cpp
void Fl_Tree::resize (int X, int Y, int W, int H ) [virtual]
```

Resizes the Fl_Group widget and all of its children. The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)
See also

Fl_Group::resizable(Fl_Widget*)
Fl_Group::resizable()
Fl_Widget::resize(int,int,int)

Reimplemented from Fl_Group.

33.149.2.76 root()

void Fl_Tree::root ( Fl_Tree_Item * newitem )

Sets the root item to 'newitem'. If a root item already exists, clear() is called first to clear it before replacing it with newitem.

Use this to install a custom item (derived from Fl_Tree_Item) as the root of the tree. This allows the derived class to implement custom drawing by overriding Fl_Tree_Item::draw_item_content().

Version

1.3.3

33.149.2.77 root_label()

void Fl_Tree::root_label ( const char * new_label )

Set the label for the root item to 'new_label'. Makes an internally managed copy of 'new_label'.

33.149.2.78 scrollbar_size() [1/2]

int Fl_Tree::scrollbar_size () const

Gets the default size of scrollbars' troughs for this widget in pixels. If this value is zero (default), this widget will use the global Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

33.149.2.79 scrollbar_size() [2/2]

void Fl_Tree::scrollbar_size ( int size )

Sets the pixel size of the scrollbars' troughs to 'size' for this widget, in pixels. Normally you should not need this method, and should use the global Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, and is the default behavior. Normally this is what you want.

Only use this method if you really need to override just THIS instance of the widget's scrollbar size. (This need should be rare.) Setting size to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.
Parameters

| in | size | Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global `Fl::scrollbar_size()` |

See also

`Fl::scrollbar_size()`

### 33.149.2.80 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**

| 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_SELECTED` |

**Returns**

- 1 : OK: item's state was changed
- 0 : OK: item was already selected, no change was made
- -1 : ERROR: item was not found

### 33.149.2.81 select() [2/2]

```cpp
int Fl_Tree::select (Fl_Tree_Item * item, int docallback = 1)
```

Select the specified `item`.
Use `deselect()` to deselect it.
Invokes the callback depending on the value of optional parameter `docallback`.
Handles calling `redraw()` if anything changed.
The callback can use `callback_item()` and `callback_reason()` respectively to determine the item changed and the reason the callback was called.

**Parameters**

| in | item | – the item to be selected. Must not be NULL. |
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
<th>-- A flag that determines if the callback() is invoked or not:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>- the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>- 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

#### 33.149.2.82 select_all()

```c
int Fl_Tree::select_all (Fl_Tree_Item * item = 0, int docallback = 1)
```

Select 'item' and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

### Parameters

<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>0</td>
<td>- the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>- the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED</td>
</tr>
</tbody>
</table>

### Returns

Count of how many items were actually changed to the selected state.

#### 33.149.2.83 select_only()

```c
int Fl_Tree::select_only (Fl_Tree_Item * selitem, int docallback = 1)
```

Select only the specified item, deselecting all others that might be selected.
If 'selitem' is 0, first() is used.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>selitem</th>
<th>The item to be selected. If NULL, first() is used.</th>
</tr>
</thead>
</table>
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>docallback</th>
<th>− A flag that determines if the callback() is invoked or not:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>the callback() is invoked for each item that changed state (default), callback_reason() will be 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.

33.149.2.84 select_toggle()

void Fl_Tree::select_toggle ( Fl_Tree_Item *item, int docallback = 1 )

Toggle the select state of the specified 'item'.
Invokes the callback depending on the value of optional parameter 'docallback'.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

<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>0</td>
<td>the callback() is not invoked</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>the callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED</td>
</tr>
</tbody>
</table>

33.149.2.85 selectbox() [1/2]

Fl_Boxtype Fl_Tree::selectbox ( ) const

Sets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme().

33.149.2.86 selectbox() [2/2]

void Fl_Tree::selectbox ( Fl_Boxtype val )

Gets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme().

33.149.2.87 selectmode() [1/2]

Fl_Tree_Select Fl_Tree::selectmode ( ) const

Gets the tree's current selection mode.
See Fl_Tree_Select for possible values.
33.149.2.88  selectmode() [2/2]

void Fl_Tree::selectmode ( Fl_Tree_Select val )

Sets the tree's selection mode.
See Fl_Tree_Select for possible values.

33.149.2.89  set_item_focus()

void Fl_Tree::set_item_focus ( Fl_Tree_Item * item )

Set the item that currently should have keyboard focus.
Handles calling redraw() to update the focus box (if it is visible).

Parameters

| in  | item | The item that should take focus. If NULL, none will have focus. |

33.149.2.90  show_item() [1/2]

void Fl_Tree::show_item ( Fl_Tree_Item * item )

Adjust the vertical scrollbar to show 'item' at the top of the display IF it is currently off-screen (for instance show_item_top()).
If it is already on-screen, no change is made.

Parameters

| in  | item | The item to be shown. If NULL, first() is used. |

See also

show_item_top(), show_item_middle(), show_item_bottom()

33.149.2.91  show_item() [2/2]

void Fl_Tree::show_item ( Fl_Tree_Item * item, int yoff )

Adjust the vertical scrollbar so that 'item' is visible 'yoff' pixels from the top of the Fl_Tree widget's display.
For instance, yoff=0 will position the item at the top.
If yoff is larger than the vertical scrollbar's limit, the value will be clipped. So if yoff=100, but scrollbar's max is 50, then 50 will be used.

Parameters

| in  | item | The item to be shown. If NULL, first() is used. |
| in  | yoff | The pixel offset from the top for the displayed position. |

See also

show_item_top(), show_item_middle(), show_item_bottom()
33.149.2.92 show_item_bottom()

void Fl_Tree::show_item_bottom ( Fl_Tree_Item * item )
Adjust the vertical scrollbar so that 'item' is at the bottom of the display.

Parameters

| in  | item | The item to be shown. If NULL, first() is used. |

33.149.2.93 show_item_middle()

void Fl_Tree::show_item_middle ( Fl_Tree_Item * item )
Adjust the vertical scrollbar so that 'item' is in the middle of the display.

Parameters

| in  | item | The item to be shown. If NULL, first() is used. |

33.149.2.94 show_item_top()

void Fl_Tree::show_item_top ( Fl_Tree_Item * item )
Adjust the vertical scrollbar so that 'item' is at the top of the display.

Parameters

| in  | item | The item to be shown. If NULL, first() is used. |

33.149.2.95 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

33.149.2.96 showcollapse()

int Fl_Tree::showcollapse ( ) const
Returns 1 if the collapse icon is enabled, 0 if not.
See also

showcollapse(int)
### showcollapse() [2/2]

```cpp
void Fl_Tree::showcollapse (int val)
```

Set if we should show the collapse icon or not. If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via `open()` and `close()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1: shows collapse icons (default), 0: hides collapse icons.</td>
</tr>
</tbody>
</table>

### showroot()

```cpp
void Fl_Tree::showroot (int val)
```

Set if the root item should be shown or not.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 – show the root item (default), 0 – hide the root item.</td>
</tr>
</tbody>
</table>

### sortorder()

```cpp
Fl_Tree_Sort Fl_Tree::sortorder ( ) const
```

Set the default sort order used when items are added to the tree. See `Fl_Tree_Sort` for possible values.

### usericon() [1/2]

```cpp
Fl_Image * Fl_Tree::usericon ( ) const
```

Returns the `Fl_Image` being used as the default user icon for all newly created items. Returns zero if no icon has been set, which is the default.

### usericon() [2/2]

```cpp
void Fl_Tree::usericon (Fl_Image * val)
```

Sets the `Fl_Image` to be used as the default user icon for all newly created items. If you want to specify user icons on a per-item basis, use `Fl_Tree_Item::usericon()` instead.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>– The new image to be used, or zero to disable user icons.</td>
</tr>
</tbody>
</table>

### vposition() [1/2]

```cpp
int Fl_Tree::vposition ( ) const
```

Returns the vertical scroll position as a pixel offset. The position returned is how many pixels of the tree are scrolled off the top edge of the screen.
See also

\texttt{vposition(int), hposition(), hposition(int)}

\subsection*{33.149.2.103 \texttt{vposition} () [2/2]}

\begin{verbatim}
void Fl_Tree::vposition (  
    int pos  
)
\end{verbatim}

Sets the vertical scroll offset to position \texttt{pos}.
The position is how many pixels of the tree are scrolled off the top edge of the screen.

\begin{itemize}
\item \texttt{pos} \hspace{1cm} The vertical position (in pixels) to scroll the tree to.
\end{itemize}

See also

\texttt{vposition(), hposition(), hposition(int)}

The documentation for this class was generated from the following files:

\begin{itemize}
\item \texttt{Fl_Tree.H}
\item \texttt{Fl_Tree.cxx}
\end{itemize}

\section*{33.150 \texttt{Fl\_Tree\_Item} Class Reference}

Tree widget item.

\begin{verbatim}
#include <Fl_Tree_Item.H>
\end{verbatim}

\textbf{Public Member Functions}

\begin{itemize}
\item \textbf{void activate (int val\textasciitilde1)}
  \begin{center}
  Change the item's activation state to the optionally specified 'val'.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * add (const Fl\_Tree\_Prefs &prefs, char **arr)}
  \begin{center}
  Descend into the path specified by 'arr', and add a new child there.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * add (const Fl\_Tree\_Prefs &prefs, char **arr, Fl\_Tree\_Item *newitem)}
  \begin{center}
  Descend into path specified by 'arr' and add 'newitem' there.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * add (const Fl\_Tree\_Prefs &prefs, const char *new_label)}
  \begin{center}
  Add a new child to this item with the name 'new_label' and defaults from 'prefs'.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * add (const Fl\_Tree\_Prefs &prefs, const char *new_label, Fl\_Tree\_Item *newitem)}
  \begin{center}
  Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * child (int index)}
  \begin{center}
  Return the child item for the given 'index'.
  \end{center}
\item \textbf{const Fl\_Tree\_Item * child (int t) const}
  \begin{center}
  Return the const child item for the given 'index'.
  \end{center}
\item \textbf{int children () const}
  \begin{center}
  Return the number of children this item has.
  \end{center}
\item \textbf{void clear_children ()}
  \begin{center}
  Clear all the children for this item.
  \end{center}
\item \textbf{void close ()}
  \begin{center}
  Close this item and all its children.
  \end{center}
\item \textbf{void deactivate ()}
  \begin{center}
  Deactivate the item; the callback() won't be invoked when clicked.
  \end{center}
\item \textbf{\texttt{Fl\_Tree\_Item} * deparent (int index)}
\end{itemize}
Deparent child at index position \texttt{\textasciitilde{pos}}.

• int \texttt{depth (const)}
  Returns how many levels deep this item is in the hierarchy.

• void \texttt{deselect ()}
  Disable the item's selection state.

• int \texttt{deselect_all ()}
  Deselect item and all its children.

• void \texttt{draw (int X, int &Y, int W, Fl_Tree_Item \texttt{*itemfocus}, int &tree_item_xmax, int lastchild=1, int render=1)}
  Draw this item and its children.

• virtual int \texttt{draw_item_content (int render)}
  Draw the item content.

• int \texttt{event_on_collapse_icon (const Fl_Tree_Prefs &prefs) const}
  Was the event on the 'collapse' button of this item?

• int \texttt{event_on_item (const Fl_Tree_Prefs &prefs) const}
  Was event anywhere on the item?

• int \texttt{event_on_label (const Fl_Tree_Prefs &prefs) const}
  Was event on the \texttt{label()} of this item?

• int \texttt{event_on_user_icon (const Fl_Tree_Prefs &prefs) const}
  Was the event on the 'user icon' of this item, if any?

• int \texttt{find_child (const char \texttt{*name})}
  Return the index of the immediate child of this item that has the label \texttt{\textasciitilde{name}}.

• int \texttt{find_child (Fl_Tree_Item \texttt{*item})}
  Find the index number for the specified \texttt{\textasciitilde{item}} in the current item's list of children.

• Fl_Tree_Item * \texttt{find_child_item (char **arr)}
  Non-const version of \texttt{Fl_Tree_Item::find_child_item(char \textasciitilde{**arr}) const}.

• const Fl_Tree_Item * \texttt{find_child_item (char **an) const}
  Find child item by descending array \texttt{\textasciitilde{arr}} of names.

• Fl_Tree_Item * \texttt{find_child_item (const char \texttt{*name}) const}
  Return the /immediate/ child of current item that has the label \texttt{\textasciitilde{name}}.

• Fl_Tree_Item * \texttt{find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0)}
  Non-const version of \texttt{Fl_Tree_Item::find_clicked(const Fl_Tree_Prefs \texttt{&}, int yonly) const}.

• const Fl_Tree_Item * \texttt{find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0) const}
  Find the item that the last event was over.

• Fl_Tree_Item * \texttt{find_item (char **arr)}
  Non-const version of \texttt{Fl_Tree_Item::find_item(char \textasciitilde{**arr}) const}.

• const Fl_Tree_Item * \texttt{find_item (char **an) const}
  Find item by descending array of \texttt{\textasciitilde{names}}.

• Fl_Tree_Item (const Fl_Tree_Item \texttt{*o})
  Copy constructor.

• Fl_Tree_Item (const Fl_Tree_Prefs &prefs)
  Constructor.

• Fl_Tree_Item (Fl_Tree \texttt{*tree})
  Constructor.

• int \texttt{h (const)}
  The item's height.

• int \texttt{has_children () const}
  See if this item has children.

• Fl_Tree_Item * \texttt{insert (const Fl_Tree_Prefs &prefs, const char \texttt{*new_label}, int pos=0)}
  Insert a new item named \texttt{\textasciitilde{new_label}} into current item's children at a specified position \texttt{\textasciitilde{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 as 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.*

### 33.150.1 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 33.57 Fl_Tree_Item's internal dimensions.](image)

### 33.150.2 Constructor & Destructor Documentation

#### 33.150.2.1 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.
33.150.2.2 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

33.150.3 Member Function Documentation

33.150.3.1 activate()

void Fl_Tree_Item::activate (  
    int val = 1  ) [inline]

Change the item's activation state to the optionally specified 'val'.  
When deactivated, the item will be 'grayed out'; the callback() won't be invoked if the user clicks on the label. If a widget() is associated with the item, its activation state will be changed as well.  
If 'val' is not specified, the item will be activated.

33.150.3.2 add() [1/4]

Fl_Tree_Item * Fl_Tree_Item::add (  
    const Fl_Tree_Prefs & prefs,  
    char ** arr  
)

Descend into the path specified by 'arr', and add a new child there.  
Should be used only by Fl_Tree's internals. Adds the item based on the value of prefs.sortorder().  

Returns
the item added.

Version
1.3.0 release

33.150.3.3 add() [2/4]

Fl_Tree_Item * Fl_Tree_Item::add (  
    const Fl_Tree_Prefs & prefs,  
    char ** arr,  
    Fl_Tree_Item * newitem  
)

Descend into path specified by 'arr' and add 'newitem' there.  
Should be used only by Fl_Tree's internals. If item is NULL, a new item is created. Adds the item based on the value of prefs.sortorder().  

Returns
the item added.

Version
1.3.3 ABI feature
33.150.3.4 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

33.150.3.5 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

33.150.3.6 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

33.150.3.7 child()

cast 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'.

Generated by Doxygen
33.150.3.8 deactivate()

void Fl_Tree_Item::deactivate ( ) [inline]
Deactivate the item; the callback() won't be invoked when clicked.
Same as activate(0)

33.150.3.9 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()/nextSibling() set to NULL.

Returns

• pointer to orphaned item on success
• NULL on error (could not deparent the item)

See also

reparent()

33.150.3.10 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.

33.150.3.11 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".

33.150.3.12 draw()

void Fl_Tree_Item::draw ( int X, int & Y, int W, Fl_Tree_Item * itemfocus, int & tree_item_xmax, int lastchild = 1, int render = 1 )
Draw this item and its children.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>X</th>
<th>Vertical position for item being drawn</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>Y</td>
<td>Vertical position for item being drawn, returns new position for next item</td>
</tr>
<tr>
<td>in</td>
<td>W</td>
<td>Recommended width for item</td>
</tr>
<tr>
<td>in</td>
<td>itemfocus</td>
<td>The tree's current focus item (if any)</td>
</tr>
<tr>
<td>in, out</td>
<td>tree_item_xmax</td>
<td>The tree's running xmax (right-most edge so far). Mainly used by parent tree when render==0 to calculate tree's max width.</td>
</tr>
<tr>
<td>in</td>
<td>lastchild</td>
<td>Is this item the last child in a subtree?</td>
</tr>
<tr>
<td>in</td>
<td>render</td>
<td>Whether or not to render the item: 0: no rendering, just calculate size w/out drawing. 1: render item as well as size calc</td>
</tr>
</tbody>
</table>

Generated by Doxygen
33.150.3.13 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>in</th>
<th>x1</th>
<th>The left hand X position of the horizontal connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x2</td>
<td>The right hand X position of the horizontal connector</td>
</tr>
<tr>
<td>in</td>
<td>y</td>
<td>The vertical position of the horizontal connector</td>
</tr>
<tr>
<td>in</td>
<td>prefs</td>
<td>The Fl_Tree prefs</td>
</tr>
</tbody>
</table>

33.150.3.14 draw_item_content()

```cpp
int Fl_Tree_Item::draw_item_content (  
    int render ) [virtual]
```

Draw the item content.

This method can be overridden to implement custom drawing by filling the label_[xywh]() area with content.

A minimal example of how to override draw_item_content() and draw just a normal item’s background and label ourselves:

```cpp
class MyTreeItem : public Fl_Tree_Item {
public:
    MyTreeItem() { }
    ~MyTreeItem() { }

    // DRAW OUR CUSTOM CONTENT FOR THE ITEM
    int draw_item_content(int render) {
        // Our item's dimensions + text content
        int X=label_x(), Y=label_y(), W=label_w(), H=label_h();
        const char *text = label() ? label() : "";
        // Rendering? Do any drawing that's needed
        if ( render ) {  
            // Draw bg -- a filled rectangle
            fl_color(drawbgcolor()); fl_rectf(X,Y,W,H);
            // Draw label
            fl_font(labelfont(), labelsize()); // use item's label font/size
            fl_color(drawfgcolor()); // use recommended fg color
            fl_draw(text, X,Y,W,H, FL_ALIGN_LEFT); // draw the item's label
        }
        // Rendered or not, we must calculate content's max X position
        int lw=0, lh=0;
        fl_measure(text, lw, lh); // get width of label text
        return X + lw; // return X + label width
    }
};
```

You can draw anything you want inside draw_item_content() using any of the fl_draw.H functions, as long as it’s within the label’s xywh area.

To add instances of your custom item to the tree, you can use:

```cpp
// Example #1: using add()
MyTreeItem *bart = new MyTreeItem(...); // class derived from Fl_Tree_Item
item = tree->add("/Simpsons/Bart", bart); // Add item as /Simpsons/Bart
```

...or you can insert or replace existing items:

```cpp
// Example #2: using replace()
MyTreeItem *marge = new MyTreeItem(...); // class derived from Fl_Tree_Item
item = tree->add("/Simpsons/Marge"); // create item
item->replace(mi); // replace it with our own
```
Parameters

| in | render | Whether we should render content (1), or just tally the geometry (0). Fl_Tree may want only to find the widest item in the tree for scrollbar calculations. |

Returns

the right-most X coordinate, or 'xmax' of content we drew, i.e. the "scrollable" content. The tree uses the largest xmax to determine the maximum width of the tree's content (needed for e.g. computing the horizontal scrollbar's size).

Version

1.3.3 ABI feature

33.150.3.15 draw_vertical_connector()

void Fl_Tree_Item::draw_vertical_connector (  
  int x,  
  int y1,  
  int y2,  
  const Fl_Tree_Prefs & prefs ) [protected]

Internal: Vertical connector line based on preference settings.

Parameters

| 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 |

33.150.3.16 drawbgcolor()

Fl_Color Fl_Tree_Item::drawbgcolor ( ) const [protected]

Returns the recommended background color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI

33.150.3.17 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
33.150.3.18 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

33.150.3.19 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.

33.150.3.20 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

33.150.3.21 find_child_item() [2/2]

```cpp
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

33.150.3.22 find_clicked()

```cpp
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

| in  | prefs   | The parent tree's Fl_Tree_Prefs |
| in  | only    | – 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X. |

Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

33.150.3.23 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

33.150.3.24 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.

33.150.3.25 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()
33.150.3.26  insert_above()

\texttt{Fl_Tree_Item * Fl_Tree_Item::insert_above (}
\texttt{const Fl_Tree_Prefs & prefs,}
\texttt{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.

33.150.3.27  label()

\texttt{void Fl_Tree_Item::label (}
\texttt{const char * name )}

Set the label to 'name'.

Makes and manages an internal copy of 'name'.

33.150.3.28  label_h()

\texttt{int Fl_Tree_Item::label_h ( ) const [inline]}

The item's label height.

Version

1.3.3

33.150.3.29  label_w()

\texttt{int Fl_Tree_Item::label_w ( ) const [inline]}

The item's maximum label width to right edge of \texttt{Fl\_Tree}'s inner width within scrollbars.

Version

1.3.3

33.150.3.30  label_x()

\texttt{int Fl_Tree_Item::label_x ( ) const [inline]}

The item's label x position relative to the window.

Version

1.3.3

33.150.3.31  label_y()

\texttt{int Fl_Tree_Item::label_y ( ) const [inline]}

The item's label y position relative to the window.

Version

1.3.3
33.150.3.32 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. 

33.150.3.33 labelbgcolor() [2/2]

`void Fl_Tree_Item::labelbgcolor ( Fl_Color val ) [inline]`

Set item's label background color.

A special case is made for color 0xffffffff which uses the parent tree's bg color.

33.150.3.34 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)`

33.150.3.35 move() [2/2]

`int Fl_Tree_Item::move ( int to, int from )`

Move an item within its parent using index numbers.

Item is moved 'to' its new position 'from' its old position.

Returns

- 0: Success
- -1: range error (e.g. if 'to' or 'from' out of range).

(Other return values reserved for future use)
### move_above()

```cpp
int Fl_Tree_Item::move_above (Fl_Tree_Item * item)
```

Moves the current item above the specified `item`. This is the equivalent of calling `move(item,0,0)`.

**Returns**

- 0 on success.
- On error returns a negative value; see `move(Fl_Tree_Item*,int,int)` for possible error codes.

**See also**

- `move_below()`, `move_into()`, `move(int,int)`, `move(Fl_Tree_Item*,int,int)`

### move_below()

```cpp
int Fl_Tree_Item::move_below (Fl_Tree_Item * item)
```

Moves the current item below the specified `item`. This is the equivalent of calling `move(item,1,0)`.

**Returns**

- 0 on success.
- On error returns a negative value; see `move(Fl_Tree_Item*,int,int)` for possible error codes.

**See also**

- `move_above()`, `move_into()`, `move(int,int)`, `move(Fl_Tree_Item*,int,int)`

### 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)`
33.150.3.39  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.

33.150.3.40  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

33.150.3.41  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.

33.150.3.42  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

33.150.3.43  parent()

```cpp
void Fl_Tree_Item::parent ( Fl_Tree_Item * val ) [inline]
```

Set the parent for this item.
Should only be used by `Fl_Tree`'s internals.
33.150.3.44  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

33.150.3.45  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).

33.150.3.46  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()

33.150.3.47  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.

33.150.3.48  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.
33.150.3.49  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

33.150.3.50  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

33.150.3.51  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.

33.150.3.52  reparent()

int Fl_Tree_Item::reparent (Fl_Tree_Item ∗newchild, int pos )
Reparent specified item as a child of ourself at position 'pos'.
Typically 'newchild' was recently orphaned with deparent().

Returns
• 0: on success
• -1: on error (e.g. if 'pos' out of range) with no changes made.

See also
deparent()
33.150.3.53  replace()

Fl_Tree_Item * Fl_Tree_Item::replace (  
    Fl_Tree_Item * newitem)

Replace the current item with a new item.
The current item is destroyed if successful. No checks are made to see if an item with the same name exists.
This method can be used to, for example, install 'custom' items into the tree derived from Fl_Tree_Item; see
draw_item_content().

Parameters

in  newitem  The new item to replace the current item

Returns

newitem on success, NULL if could not be replaced.

See also

Fl_Tree_Item::draw_item_content(), Fl_Tree::root(Fl_Tree_Item*)

Version

1.3.3 ABI feature

33.150.3.54  replace_child()

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

33.150.3.55  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.
33.150.3.56  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".

33.150.3.57  show_self()

    void Fl_Tree_Item::show_self ( const char * indent = "" ) const
Print the tree as 'ascii art' to stdout.
Used mainly for debugging.

33.150.3.58  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()

33.150.3.59  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.

33.150.3.60  swap_children() [2/2]

    void Fl_Tree_Item::swap_children ( int ax,
    int bx )
Swap two of our children, given two child index values 'ax' and 'bx'.
Use e.g. for sorting.
This method is FAST, and does not involve lookups.
No range checking is done on either index value.

Parameters

| in  | ax, bx | the index of the items to swap |

33.150.3.61  tree() [1/2]

    Fl_Tree * Fl_Tree_Item::tree ( ) [inline]
Return the tree for this item.

Version

1.3.4

33.150.3.62  tree() [2/2]

const Fl_Tree * Fl_Tree_Item::tree ( ) const [inline]

Return the tree for this item.

Version

1.3.3

33.150.3.63  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>in</th>
<th>index</th>
<th>Our index# in the parent.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Special case if index=-1: become an orphan; null out all parent/sibling associations.</td>
</tr>
</tbody>
</table>

33.150.3.64  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.

33.150.3.65  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_Pixmap *usr_icon = new Fl_Pixmap(...); // your usericon
Fl_Pixmap *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
33.150.3.66  usericon()

void Fl_Tree_Item::usericon ( 
    Fl_Image * val ) [inline]

Set the item's user icon to an Fl_Image.
Use '0' to disable. No internal copy is made, caller must manage icon's memory.
Note, if you expect your items to be deactivated(), use userdeicon(Fl_Image+) to set up a 'grayed out' version of
your icon to be used for display.

See also
    userdeicon(Fl_Image+)

33.150.3.67  visible_r()

int Fl_Tree_Item::visible_r ( ) const
See if item and all its parents are open() and visible().
Returns
  1 – item and its parents are open() and visible()  0 – item (or one of its parents) are invisible or close()ed.

The documentation for this class was generated from the following files:

• Fl_Tree_Item.H
• Fl_Tree_Item.cxx

33.151  Fl_Tree_Item_Array Class Reference

Manages an array of Fl_Tree_Item pointers.
#include <Fl_Tree_Item_Array.H>

Public Member Functions

• void add (Fl_Tree_Item *val)
   Add an item* to the end of the array.
• void clear ()
   Clear the entire array.
• int deparent (int pos)
   Deparent item at 'pos' from our list of children.
• Fl_Tree_Item_Array (const Fl_Tree_Item_Array *o)
   Copy constructor. Makes new copy of array, with new instances of each item.
• Fl_Tree_Item_Array (int new_chunksize=10)
   Constructor; creates an empty array.
• void insert (int pos, Fl_Tree_Item *new_item)
   Insert an item at index position pos.
• int manage_item_destroy () const
• void manage_item_destroy (int val)
   Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.
• int move (int to, int from)
   Move item at 'from' to new position 'to' in the array.
• Fl_Tree_Item * operator[] (int i)
   Return the item and index i.
• const Fl_Tree_Item * operator[] (int i) const
   Const version of operator[](int i)
• int remove (Fl_Tree_Item *item)
Remove the item from the array.

- void remove (int index)
  Remove the item at.

- int reparent (Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos)
  Reparent specified item as a child of ourself.

- void replace (int pos, Fl_Tree_Item *new_item)
  Replace the item at index with newitem.

- void swap (int ax, int bx)
  Swap the two items at index positions ax and bx.

- int total () const
  Return the total items in the array, or 0 if empty.

- ~Fl_Tree_Item_Array()
  Destructor. Calls each item's destructor, destroys internal _items array.

### 33.151.1 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).

### 33.151.2 Constructor & Destructor Documentation

#### 33.151.2.1 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.

### 33.151.3 Member Function Documentation

#### 33.151.3.1 add()

void Fl_Tree_Item_Array::add (Fl_Tree_Item *val)
Add an item* to the end of the array.

Assumes the item was created with ‘new’, and will remain allocated. Fl_Tree_Item_Array will handle calling the item’s destructor when the array is cleared or the item remove()'ed.

#### 33.151.3.2 clear()

void Fl_Tree_Item_Array::clear ()
Clear the entire array.

Each item will be deleted (destructors will be called), and the array will be cleared. total() will return 0.
33.151.3.3 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)

33.151.3.4 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 \geq total(), the item is appended to the array.
If \p pos < 0, the item is prepended (works like pos == 0).

33.151.3.5 manage_item_destroy()

```cpp
void Fl_Tree_Item_Array::manage_item_destroy (int val) [inline]
```

Option to control if `Fl_Tree_Item_Array`'s destructor will also destroy the `Fl_Tree_Item`'s. If set: items and item array is destroyed. If clear: only the item array is destroyed, not items themselves.

33.151.3.6 move()

```cpp
int Fl_Tree_Item_Array::move (int to, int from)
```

Move item at 'from' to new position 'to' in the array. Due to how the moving an item shuffles the array around, a positional 'move' implies things that may not be obvious:

- When 'from' moved lower in tree, appears BELOW item that was at 'to'.
- When 'from' moved higher in tree, appears ABOVE item that was at 'to'.

`\returns 0` on success, `-1` on range error (e.g. if 'to' or 'from' out of range)

33.151.3.7 remove() [1/2]

```cpp
int Fl_Tree_Item_Array::remove (Fl_Tree_Item * item)
```

Remove the item from the array.

`\returns 0` if removed, or `-1` if the item was not in the array.

33.151.3.8 remove() [2/2]

```cpp
void Fl_Tree_Item_Array::remove (int index)
```

Remove the item at.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>index</strong></td>
<td>from the array. The item will be deleted (if non-NULL), so its destructor will be called.</td>
</tr>
</tbody>
</table>

### 33.151.3.9 reparent()

```cpp
int Fl_Tree_Item_Array::reparent (
        Fl_Tree_Item * item,
        Fl_Tree_Item * newparent,
        int pos )
```

Reparent specified item as a child of ourself. Typically 'newchild' was recently orphaned with deparent().

\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)

### 33.151.3.10 replace()

```cpp
void Fl_Tree_Item_Array::replace ( 
        int index, 
        Fl_Tree_Item * newitem )
```

Replace the item at index with newitem. Old item at index position will be destroyed, and the new item will take it's place, and stitched into the linked list.

The documentation for this class was generated from the following files:

- Fl_Tree_Item_Array.H
- Fl_Tree_Item_Array.cxx

### 33.152 Fl_Tree_Prefs Class Reference

Tree widget's preferences.

```cpp
#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
  Get the tree connection line's width.
• void set_labelmarginleft(int val)
  Set the label's left margin value in pixels.
Obsolete: Return the label's size in pixels. Please use item_labelsize() instead.

- void *labelsize*(Fl_Fontsize val)
  
  Obsolete: Set the label's size in pixels to val. Please use item_labelsize(Fl_Fontsize) instead.

- int *linespacing* () const
  
  Get the line spacing value in pixels.

- void *linespacing*(int val)
  
  Set the line spacing value in pixels.

- int *marginbottom* () const
  
  Get the bottom margin's value in pixels.

- void *marginbottom*(int val)
  
  Set the bottom margin's value in pixels. This is the extra distance the vertical scroller lets you travel.

- int *marginleft* () const
  
  Get the left margin's value in pixels.

- void *marginleft*(int val)
  
  Set the left margin's value in pixels.

- int *margintop* () const
  
  Get the top margin's value in pixels.

- void *margintop*(int val)
  
  Set the top margin's value in pixels.

- int *openchild_marginbottom* () const
  
  Get the margin below an open child in pixels.

- void *openchild_marginbottom*(int val)
  
  Set the margin below an open child in pixels.

- Fl_Image *opendeicon* () const
  
  Return the deactivated version of the open icon, if any.

- Fl_Image *openicon* () const
  
  Get the current default 'open' icon.

- void *openicon*(Fl_Image *val)
  
  Sets the default icon to be used as the 'open' icon when items are add(jed) to the tree.

- Fl_Boxtype *selectbox* () const
  
  Get the default selection box's box drawing style as an Fl_Boxtype.

- void *selectbox*(Fl_Boxtype val)
  
  Set the default selection box's box drawing style to val.

- Fl_Tree_Select *selectmode* () const
  
  Get the selection mode used for the tree.

- void *selectmode*(Fl_Tree_Select val)
  
  Set the selection mode used for the tree to val.

- char *showcollapse* () const
  
  Returns 1 if the collapse icon is enabled, 0 if not.

- void *showcollapse*(int val)
  
  Set if we should show the collapse icon or not.

- int *showroot* () const
  
  Returns 1 if the root item is to be shown, or 0 if not.

- void *showroot*(int val)
  
  Set if the root item should be shown or not.

- Fl_Tree_Sort *sortorder* () const
  
  Get the default sort order value.

- void *sortorder*(Fl_Tree_Sort val)
  
  Set the default sort order value.

- Fl_Image *userdeicon* () const
  
  Return the deactivated version of the user icon, if any.
• Fl_Image * usericon() const
  Gets the default 'user icon' (default is 0)
• void usericon(Fl_Image *val)
  Sets the default 'user icon' Returns the Fl_Image of the icon, or 0 if none (default).
• int usericonmarginleft() const
  Get the user icon's left margin value in pixels.
• void usericonmarginleft(int val)
  Set the user icon's left margin value in pixels.
• int widgetmarginleft() const
  Get the widget()'s left margin value in pixels.
• void widgetmarginleft(int val)
  Set the widget's left margin value in pixels.
• ~Fl_Tree_Prefs()
  Fl_Tree_Prefs destructor.

33.152.1 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.

33.152.2 Member Function Documentation

33.152.2.1 closedeicon()
Fl_Image * Fl_Tree_Prefs::closedeicon() const [inline]
Return the deactivated version of the close icon, if any.
Returns 0 if none.

33.152.2.2 closeicon()
void Fl_Tree_Prefs::closeicon(
  Fl_Image * val) [inline]
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.

33.152.2.3 item_draw_mode()
void Fl_Tree_Prefs::item_draw_mode (
  Fl_Tree_Item_Draw_Mode val) [inline]
Set the 'item draw mode' used for the tree to val.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode
for possible values.

33.152.2.4 item_labelbgcolor() [1/2]
Fl_Color Fl_Tree_Prefs::item_labelbgcolor (}
  void ) const [inline]
Get the default label background color. This returns the Fl_Tree::color() unless item_labelbgcolor() has been set explicitly.

### 33.152.2.5 item_labelbgcolor() [2/2]

```cpp
void Fl_Tree_Prefs::item_labelbgcolor (Fl_Color val) [inline]
```

Set the default label background color. Once set, overrides the default behavior of using Fl_Tree::color().

### 33.152.2.6 marginbottom()

```cpp
int Fl_Tree_Prefs::marginbottom ( ) const [inline]
```

Get the bottom margin's value in pixels. This is the extra distance the vertical scroller lets you travel.

### 33.152.2.7 opendeicon()

```cpp
Fl_Image * Fl_Tree_Prefs::opendeicon ( ) const [inline]
```

Return the deactivated version of the open icon, if any. Returns 0 if none.

### 33.152.2.8 openicon() [1/2]

```cpp
Fl_Image * Fl_Tree_Prefs::openicon ( ) const [inline]
```

Get the current default 'open' icon. Returns the Fl_Image* of the icon, or 0 if none.

### 33.152.2.9 openicon() [2/2]

```cpp
void Fl_Tree_Prefs::openicon (Fl_Image * val)
```

Sets the default icon to be used as the 'open' icon when items are added to the tree. This overrides the built in default '[+] icon.

**Parameters**

| in val | - The new image, or zero to use the default [+] icon. |

### 33.152.2.10 selectmode()

```cpp
void Fl_Tree_Prefs::selectmode (Fl_Tree_Select val) [inline]
```

Set the selection mode used for the tree to val. This affects how items in the tree are selected when clicked on and dragged over by the mouse. See Fl_Tree_Select for possible values.

### 33.152.2.11 showcollapse()

```cpp
void Fl_Tree_Prefs::showcollapse (int val) [inline]
```

Set if we should show the collapse icon or not. If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>val</th>
<th>1: shows collapse icons (default), 0: hides collapse icons.</th>
</tr>
</thead>
</table>

### 33.152.2.12 showroot()

```cpp
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>

### 33.152.2.13 sortorder()

```cpp
void Fl_Tree_Prefs::sortorder ( Fl_Tree_Sort val ) [inline]
```

Set the default sort order value.

Defines the order new items appear when add()ed to the tree. See Fl_Tree_Sort for possible values.

### 33.152.2.14 userdeicon()

```cpp
Fl_Image * Fl_Tree_Prefs::userdeicon ( ) const [inline]
```

Return the deactivated version of the user icon, if any.

Returns 0 if none.

The documentation for this class was generated from the following files:

- Fl_Tree_Prefs.H
- Fl_Tree_Prefs.cxx

### 33.153 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.

```cpp
#include <Fl_Valuator.H>
```

Generated by Doxygen
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 $\frac{1.0}{10^\text{digits}}$.

- void **range** (double a, double b)
  
  Sets the minimum and maximum values for the valuator.

- double **round** (double)
  
  Round the passed value to the nearest step increment.

- double **step** () const
  
  Gets or sets the step value.

- void **step** (double a, int b)
  
  See double *Fl_Valuator::step()* const

- void **step** (double s)
  
  See double *Fl_Valuator::step()* const.

- void **step** (int a)
  
  See double *Fl_Valuator::step()* const

- double **value** () const
  
  Gets the floating point(double) value.

- int **value** (double)
  
  Sets the current value.
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

33.153.1 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 33.58 Valuators derived from Fl_Valuators](image_url)

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.
33.153.2 Constructor & Destructor Documentation

33.153.2.1 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.

33.153.3 Member Function Documentation

33.153.3.1 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 %.1f 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.

33.153.3.2 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.

33.153.3.3 maximum() [1/2]

double Fl_Valuator::maximum ( )  const  [inline]

Gets the maximum value for the valuator.

33.153.3.4 maximum() [2/2]

void Fl_Valuator::maximum (  
    double a )  [inline]

Sets the maximum value for the valuator.

33.153.3.5 minimum() [1/2]

double Fl_Valuator::minimum ( )  const  [inline]

Gets the minimum value for the valuator.
33.153.3.6 minimum() [2/2]

```cpp
void Fl_Valuator::minimum (double a) [inline]
```
Sets the minimum value for the valuator.

33.153.3.7 precision()

```cpp
void Fl_Valuator::precision (int digits)
```
Sets the step value to $1.0 / 10^{\text{digits}}$.
Precision digits is limited to 0...9 to avoid internal overflow errors. Values outside this range are clamped.

**Note**

For negative values of digits the step value is set to $A = 1.0$ and $B = 1$, i.e. $1.0/1 = 1$.

33.153.3.8 range()

```cpp
void Fl_Valuator::range (double a, double b) [inline]
```
Sets the minimum and maximum values for the valuator.
When the user manipulates the widget, the value is limited to this range. This clamping is done after rounding to the step value (this makes a difference if the range is not a multiple of the step).
The minimum may be greater than the maximum. This has the effect of "reversing" the object so the larger values are in the opposite direction. This also switches which end of the filled sliders is filled.
Some widgets consider this a "soft" range. This means they will stop at the range, but if the user releases and grabs the control again and tries to move it further, it is allowed.
The range may affect the display. You must `redraw()` the widget after changing the range.

33.153.3.9 round()

```cpp
double Fl_Valuator::round (double v)
```
Round the passed value to the nearest step increment. Does nothing if step is zero.

33.153.3.10 step()

```cpp
double Fl_Valuator::step () const [inline]
```
Gets or sets the step value. As the user moves the mouse the value is rounded to the nearest multiple of the step value. This is done before clamping it to the range. For most widgets the default step is zero. For precision the step is stored as the ratio of a double $A$ and an integer $B = A/B$. You can set these values directly. Currently setting a floating point value sets the nearest $A/1$ or $1/B$ value possible.

33.153.3.11 value() [1/2]

```cpp
double Fl_Valuator::value () const [inline]
```
Gets the floating point(double) value.
See int value(double)
33.153.3.12 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).

33.153.3.13 value_damage()

```cpp
void Fl_Valuator::value_damage () [protected], [virtual]
```

Asks for partial redraw.
Reimplemented in `Fl_Adjuster`.
The documentation for this class was generated from the following files:

- Fl_Valuator.H
- Fl_Valuator.cxx

33.154 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`:

```
FI_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) FL_OVERRIDE
  
  *Handles the specified event.*

- **void** resize (int, int, int, int) FL_OVERRIDE
  
  *Changes the size or position of the widget.*

- **int** shortcut () const
  
  *Returns the current shortcut key for the Input.*

- **void** shortcut (int s)
  
  *Sets the shortcut key to s.*

- **char** soft () const
  
  *If "soft" is turned on, the user is allowed to drag the value outside the range.*

- **void** soft (char s)
  
  *See void `Fl_Value_Input::soft(char s)`*
• **Fl_Color textcolor () const**  
  *Gets the color of the text in the value box.*

• **void textcolor (Fl_Color n)**  
  *Sets the color of the text in the value box.*

• **Fl_Font textfont () const**  
  *Gets the typeface of the text in the value box.*

• **void textfont (Fl_Font s)**  
  *Sets the typeface of the text in the value box.*

• **Fl_Fontsize textsize () const**  
  *Sets the size of the text in the value box.*

• **void textsize (Fl_Fontsize s)**  
  *Sets the size of the text in the value box.*

**Public Attributes**

• **Fl_Input input**

**Protected Member Functions**

• **void draw () FL_OVERRIDE**  
  *Draws the widget.*

**Additional Inherited Members**

### 33.154.1 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 33.59 Fl_Value_Input**

See also

`Fl_Widget::shortcut_label(int)`

### 33.154.2 Constructor & Destructor Documentation

---

*Generated by Doxygen*
33.154.2.1 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.

33.154.3 Member Function Documentation

33.154.3.1 cursor_color() [1/2]

Fl_Color Fl_Value_Input::cursor_color ( ) const [inline]  

Gets the color of the text cursor.  
The text cursor is black by default.

33.154.3.2 cursor_color() [2/2]

void Fl_Value_Input::cursor_color (  
  Fl_Color n ) [inline]  

Sets the color of the text cursor.  
The text cursor is black by default.

33.154.3.3 draw()

void Fl_Value_Input::draw ( ) [protected], [virtual]  

Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrew 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.

33.154.3.4 handle()

int Fl_Value_Input::handle (  
  int event ) [virtual]  

Handles the specified event.  
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short- 
circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

| in   | event | the kind of event received |

Return values

| 0    | if the event was not used or understood |
Return values

\texttt{true} if the event was used and can be deleted

See also

\texttt{Fl\_Event}

Reimplemented from \texttt{Fl\_Widget}.

### 33.154.3.5 resize()

\begin{verbatim}
void Fl_Value_Input::resize (  
    int x,  
    int y,  
    int w,  
    int h ) [virtual]
\end{verbatim}

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 \textit{not} call the \texttt{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 \texttt{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.

\texttt{position(X, Y)} is a shortcut for \texttt{resize(X, Y, w(), h())}, and \texttt{size(W, H)} is a shortcut for \texttt{resize(x(), y(), W, H)}.

Parameters

\begin{itemize}
    \item \texttt{x, y} new position relative to the parent window
    \item \texttt{w, h} new size
\end{itemize}

See also

\texttt{position(int,int)}, \texttt{size(int,int)}

Reimplemented from \texttt{Fl\_Widget}.

### 33.154.3.6 shortcut() [1/2]

\begin{verbatim}
int Fl_Value_Input::shortcut ( ) const [inline]
\end{verbatim}

Returns the current shortcut key for the Input.

See also

\texttt{Fl_Value_Input::shortcut(int)}

### 33.154.3.7 shortcut() [2/2]

\begin{verbatim}
void Fl_Value_Input::shortcut (  
    int s ) [inline]
\end{verbatim}

Sets the shortcut key to \texttt{s}.

Setting this overrides the use of ‘&’ in the \texttt{label()}. The value is a bitwise OR of a key and a set of shift flags, for example \texttt{FL\_ALT | \textasciitilde{a'}, FL\_ALT | (FL\_F + 10)}, or just ‘a’. A value of 0 disables the shortcut.

The key can be any value returned by \texttt{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 \texttt{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).
33.154.3.8 **soft()**

```cpp
c 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.

33.154.3.9 **textcolor()**

```cpp
c Fl_Color Fl_Value_Input::textcolor ( ) const [inline]
```

Gets the color of the text in the value box.

33.154.3.10 **textfont()** [1/2]

```cpp
c Fl_Font Fl_Value_Input::textfont ( ) const [inline]
```

Gets the typeface of the text in the value box.

33.154.3.11 **textfont()** [2/2]

```cpp
void Fl_Value_Input::textfont ( 
    Fl_Font s ) [inline]
```

Sets the typeface of the text in the value box.

33.154.3.12 **textsize()** [1/2]

```cpp
c Fl_Fontsize Fl_Value_Input::textsize ( ) const [inline]
```

Gets the size of the text in the value box.

33.154.3.13 **textsize()** [2/2]

```cpp
void Fl_Value_Input::textsize ( 
    Fl_Fontsize s ) [inline]
```

Sets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Input.H
- Fl_Value_Input.cxx

### 33.155 Fl_Value_Output Class Reference

The `Fl_Value_Output` widget displays a floating point value.

```cpp
#include <Fl_Value_Output.H>
```

Inheritance diagram for `Fl_Value_Output`:
Public Member Functions

- **Fl_Value_Output (int x, int y, int w, int h, const char *l=0)**
  
  Creates a new Fl_Value_Output widget using the given position, size, and label string.

- **int handle (int) FL_OVERRIDE**
  
  Handles the specified event.

- **uchar soft () const**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **void soft (uchar s)**
  
  If "soft" is turned on, the user is allowed to drag the value outside the range.

- **Fl_Color textcolor () const**
  
  Sets the color of the text in the value box.

- **void textcolor (Fl_Color s)**
  
  Gets the color of the text in the value box.

- **Fl_Font textfont () const**
  
  Gets the typeface of the text in the value box.

- **void textfont (Fl_Font s)**
  
  Sets the typeface of the text in the value box.

- **Fl_Fontsize textsize () const**
  
  Gets the size of the text in the value box.

- **void textsize (Fl_Fontsize s)**

Protected Member Functions

- **void draw () FL_OVERRIDE**
  
  Draws the widget.

Additional Inherited Members

33.155.1 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 33.60 Fl_Value_Output

33.155.2 Constructor & Destructor Documentation
33.155.2.1 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.

33.155.3 Member Function Documentation

33.155.3.1 draw()

void Fl_Value_Output::draw ( ) [protected], [virtual]

Draws the widget.  
Never call this function directly. FLTK will schedule redrawing whenever needed.  
If your widget must be redrawn as soon as possible, call redraw() instead.  
Override this function to draw your own widgets.  
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:  
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar  
s->draw(); // calls Fl_Scrollbar::draw()

Implements Fl_Widget.

33.155.3.2 handle()

int Fl_Value_Output::handle (  
    int event ) [virtual]

Handles the specified event.  
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.  
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.  
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

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.

33.155.3.3 soft() [1/2]

uchar Fl_Value_Output::soft ( ) const [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

33.155.3.4 soft() [2/2]
void Fl_Value_Output::soft (uchar s) [inline]
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

33.155.3.5 textcolor() [1/2]
Fl_Color Fl_Value_Output::textcolor() const [inline]
Sets the color of the text in the value box.

33.155.3.6 textcolor() [2/2]
void Fl_Value_Output::textcolor(Fl_Color s) [inline]
Gets the color of the text in the value box.

33.155.3.7 textfont() [1/2]
Fl_Font Fl_Value_Output::textfont() const [inline]
Gets the typeface of the text in the value box.

33.155.3.8 textfont() [2/2]
void Fl_Value_Output::textfont(Fl_Font s) [inline]
Sets the typeface of the text in the value box.

33.155.3.9 textsize()
Fl_Fontsize Fl_Value_Output::textsize() const [inline]
Gets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Output.H
- Fl_Value_Output.cxx

33.156 Fl_Value_Slider Class Reference

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.
#include <Fl_Value_Slider.H>
Inheritance diagram for Fl_Value_Slider:
Public Member Functions

- **Fl_Value_Slider** (int x, int y, int w, int h, const char ∗l=0)
  
  Creates a new Fl_Value_Slider widget using the given position, size, and label string.

- **int handle (int)** FL_OVERRIDE
  
  Handles the specified event.

- **Fl_Color textcolor () const**
  
  Gets the color of the text in the value box.

- **void textcolor (Fl_Color s)**
  
  Sets the color of the text in the value box.

- **Fl_Font textfont () const**
  
  Gets the typeface of the text in the value box.

- **void textfont (Fl_Font s)**
  
  Sets the typeface of the text in the value box.

- **Fl_Fontsize textsize () const**
  
  Gets the size of the text in the value box.

- **void textsize (Fl_Fontsize s)**
  
  Sets the size of the text in the value box.

- **int value_height () const**
  
  Gets the height of the value box in pixels (vertical mode only).

- **void value_height (int s)**
  
  Sets the height of the value box in pixels (vertical mode only).

- **int value_width () const**
  
  Gets the width of the value box in pixels (horizontal mode only).

- **void value_width (int s)**
  
  Sets the width of the value box in pixels (horizontal mode only).

Protected Member Functions

- **void draw ()** FL_OVERRIDE
  
  Draws the widget.

Additional Inherited Members

33.156.1 Detailed Description

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.
33.156.2 Constructor & Destructor Documentation

33.156.2.1 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.

33.156.3 Member Function Documentation

33.156.3.1 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.

33.156.3.2 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

<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_Slider.

33.156.3.3 value_height() [1/2]

```cpp
int Fl_Value_Slider::value_height ( ) const [inline]
```

Gets the height of the value box in pixels (vertical mode only).

Since

1.4.0

33.156.3.4 value_height() [2/2]

```cpp
void Fl_Value_Slider::value_height ( int s ) [inline]
```

Sets the height of the value box in pixels (vertical mode only).

Limited range checking is applied but drawing errors may occur if the size `s` is set too high or too low, particularly if the widget is resized (later).

The programmer is responsible for setting sensible values and widget sizes.

The default value set by the constructor is 25.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in s</code></td>
<td>new height of the value box</td>
</tr>
</tbody>
</table>

Since

1.4.0

33.156.3.5 value_width() [1/2]

```cpp
int Fl_Value_Slider::value_width ( ) const [inline]
```

Gets the width of the value box in pixels (horizontal mode only).

Since

1.4.0

33.156.3.6 value_width() [2/2]

```cpp
void Fl_Value_Slider::value_width ( int s ) [inline]
```

Sets the width of the value box in pixels (horizontal mode only).

Limited range checking is applied but drawing errors may occur if the size `s` is set too high or too low, particularly if the widget is resized (later).
The programmer is responsible for setting sensible values and widget sizes. The default value set by the constructor is 35.
Parameters

\begin{verbatim}
in  s  new width of the value box
\end{verbatim}

Since

1.4.0

The documentation for this class was generated from the following files:

- `Fl_Value_Slider.H`
- `Fl_Value_Slider.cxx`

### 33.157 Fl_Widget Class Reference

`Fl_Widget` is the base class for all widgets in FLTK.

```
#include <Fl_Widget.H>
```

Inheritance diagram for `Fl_Widget`:

![Inheritance Diagram](image)

**Public Member Functions**

- `void _clear_fullscreen ()`
- `void _set_fullscreen ()`
- `void activate ()`
  
  *Activates the widget.*
- `unsigned int active () const`
  
  *Returns whether the widget is active.*

Generated by Doxygen
• int active_r () const  
  Returns whether the widget and all of its parents are active.

• Fl_Align align () const  
  Gets the label alignment.

• void align (Fl_Align alignment)  
  Sets the label alignment.

• long argument () const  
  Gets the current user data (long) argument that is passed to the callback function.

• void argument (long v)  
  Sets the current user data (long) argument that is passed to the callback function.

• virtual class Fl_Gl_Window ∗ as_gl_window ()  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual class Fl_Gl_Window const ∗ as_gl_window () const  
  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

• virtual Fl_Group ∗ as_group ()  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Group const ∗ as_group () const  
  Returns an Fl_Group pointer if this widget is an Fl_Group.

• virtual Fl_Window ∗ as_window ()  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• virtual Fl_Window const ∗ as_window () const  
  Returns an Fl_Window pointer if this widget is an Fl_Window.

• void bind_deimage (Fl_Image ∗ img)  
  Sets the image to use as part of the widget label when in the inactive state.

• void bind_deimage (int f)  
  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

• void bind_image (Fl_Image ∗ img)  
  Sets the image to use as part of the widget label when in the active state.

• void bind_image (int f)  
  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

• Fl_Boxtype box () const  
  Gets the box type of the widget.

• void box (Fl_Boxtype new_box)  
  Sets the box type for the widget.

• Fl_Callback_p callback () const  
  Gets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb)  
  Sets the current callback function for the widget.

• void callback (Fl_Callback ∗ cb, void ∗ p)  
  Sets the current callback function for the widget.

• void callback (Fl_Callback0 ∗ cb)  
  Sets the current callback function for the widget.

• void callback (Fl_Callback1 ∗ cb, long p=0)  
  Sets the current callback function for the widget.

• unsigned int changed () const  
  Checks if the widget value changed since the last callback.

• void clear_active ()  
  Marks the widget as inactive without sending events or changing focus.

• void clear_changed ()  
  Marks the value of the widget as unchanged.

• void clear_damage (uchar c=0)  
  Clears or sets the damage flags.

• void clear_output ()  
  Sets a widget to accept input.
- `void clear_visible ()`
  Hides the widget.
- `void clear_visible_focus ()`
  Disables keyboard focus navigation with this widget.
- `Fl_Color color () const`
  Gets the background color of the widget.
- `void color (Fl_Color bg)`
  Sets the background color of the widget.
- `void color (Fl_Color bg, Fl_Color sel)`
  Sets the background and selection color of the widget.
- `Fl_Color color2 () const`
  For back compatibility only.
- `void color2 (unsigned a)`
  For back compatibility only.
- `int contains (const Fl_Widget ∗w) const`
  Checks if w is a child of this widget.
- `void copy_label (const char ∗new_label)`
  Sets the current label.
- `void copy_tooltip (const char ∗text)`
  Sets the current tooltip text.
- `uchar damage () const`
  Returns non-zero if draw() needs to be called.
- `void damage (uchar c)`
  Sets the damage bits for the widget.
- `void damage (uchar c, int x, int y, int w, int h)`
  Sets the damage bits for an area inside the widget.
- `int damage_resize (int, int, int, int)`
  Internal use only.
- `void deactivate ()`
  Deactivates the widget.
- `Fl_Image ∗deimage ()`
  Gets the image that is used as part of the widget label when in the inactive state.
- `const Fl_Image ∗deimage () const`
  Gets the image that is used as part of the widget label when in the inactive state.
- `void deimage (Fl_Image &img)`
  Sets the image to use as part of the widget label when in the inactive state.
- `void deimage (Fl_Image ∗img)`
  Sets the image to use as part of the widget label when in the inactive state.
- `int deimage_bound () const`
  Returns whether the inactive image is managed by the widget.
- `void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with default arguments.
- `void do_callback (Fl_Widget ∗widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
- `void do_callback (Fl_Widget ∗widget, void ∗arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)`
  Calls the widget callback function with arbitrary arguments.
- `virtual void draw ()=0`
  Draws the widget.
- `void draw_label (int, int, int, Fl_Align) const`
  Draws the label in an arbitrary bounding box with an arbitrary alignment.
Gets the widget height.

- virtual int handle (int event)  
  Handles the specified event.

- virtual void hide ()  
  Makes a widget invisible.

- FL_Imageximage ()  
  Gets the image that is used as part of the widget label when in the active state.

- const FL_Imageximage () const  
  Gets the image that is used as part of the widget label when in the active state.

- void image (FL_Imagex&img)  
  Sets the image to use as part of the widget label when in the active state.

- void image (FL_Imagex*img)  
  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const  
  Returns whether the image is managed by the widget.

- int inside (const FL_Widgetx*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 charxlabel () const  
  Gets the current label text.

- void label (const charx*text)  
  Sets the current label pointer.

- void label (FL_Labeltype a, const charx*b)  
  Shortcut to set the label text and type in one call.

- FL_Color labelcolor () const  
  Gets the label color.

- void labelcolor (FL_Color c)  
  Sets the label color.

- FL_Font labelfont () const  
  Gets the font to use.

- void labelfont (FL_Font f)  
  Sets the font to use.

- FL_Fontsize labelsize () const  
  Gets the font size in pixels.

- void labelsize (FL_Fontsize pix)  
  Sets the font size in pixels.

- FL_Labeltype labeltype () const  
  Gets the label type.

- void labeltype (FL_Labeltype a)  
  Sets the label type.

- void measure_label (int &ww, int &hh) const  
  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const  
  Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)  
  Sets whether this widget needs a keyboard.

- unsigned int output () const  
  Returns if a widget is used for output only.

- FL_Groupx*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 *)
  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_Value 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 *, const bool require_alt=false)
  Returns true if the given text t contains the entered ’&x’ shortcut.

Protected Types

• enum {
  INACTIVE = 1<<0, INVISIBLE = 1<<1, OUTPUT = 1<<2, NOBORDER = 1<<3,
  FORCE_POSITION = 1<<4, NON_MODAL = 1<<5, SHORTCUT_LABEL = 1<<6, CHANGED = 1<<7,
  OVERRIDE = 1<<8, VISIBLE_FOCUS = 1<<9, COPIED_LABEL = 1<<10, CLIP_CHILDREN = 1<<11,
  MENU_WINDOW = 1<<12, TOOLTIP_WINDOW = 1<<13, MODAL = 1<<14, NO_OVERLAY = 1<<15,
  GROUP_RELATIVE = 1<<16, COPIED_TOOLTIP = 1<<17, FULLSCREEN = 1<<18, MAC_USE_ACCENTS_MENU
  = 1<<19,
  NEEDS_KEYBOARD = 1<<20, IMAGE_BOUND = 1<<21, DEIMAGE_BOUND = 1<<22, USERFLAG3
  = 1<<29,
  USERFLAG2 = 1<<30, USERFLAG1 = 1<<31
}
Protected Member Functions

- void **clear_flag** (unsigned int c)
  Clears a flag in the flags mask.

- void **draw_backdrop** () const
  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box** () const
  Draws the widget box according its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const
  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const
  Draws a box of type t, of color c at the position X,Y and size W,H.

- void **draw_focus** () const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int X, int Y, int W, int H) const
  Draws a focus rectangle around the widget.

- void **draw_focus** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const
  Draws a focus box for the widget at the given position and size.

- void **draw_label** () const
  Draws the widget's label at the defined label position.

- void **draw_label** (int, int, int, int) const
  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)
  Creates a widget at the given position and size.

- unsigned int **flags** () const
  Gets the widget flags mask.

- void **h** (int v)
  Internal use only.

- void **set_flag** (unsigned int c)
  Sets a flag in the flags mask.

- void **w** (int v)
  Internal use only.

- void **x** (int v)
  Internal use only.

- void **y** (int v)
  Internal use only.

Friends

- class Fl_Group

33.157.1 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.

33.157.2 Member Enumeration Documentation
### 33.157.2.1 anonymous enum

**anonymous enum** [protected]

flags possible values enumeration.

See activate(), output(), visible(), changed(), set_visible_focus()

**Enumerator**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INACTIVE</td>
<td>the widget can't receive focus, and is disabled but potentially visible</td>
</tr>
<tr>
<td>INVISIBLE</td>
<td>the widget is not drawn, but can receive a few special events</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>for output only</td>
</tr>
<tr>
<td>NOBORDER</td>
<td>don't draw a decoration (Fl_Window)</td>
</tr>
<tr>
<td>FORCE_POSITION</td>
<td>don't let the window manager position the window (Fl_Window)</td>
</tr>
<tr>
<td>NON_MODAL</td>
<td>this is a hovering toolbar window (Fl_Window)</td>
</tr>
<tr>
<td>SHORTCUT_LABEL</td>
<td>the label contains a shortcut we need to draw</td>
</tr>
<tr>
<td>CHANGED</td>
<td>the widget value changed</td>
</tr>
<tr>
<td>OVERRIDE</td>
<td>position window on top (Fl_Window)</td>
</tr>
<tr>
<td>VISIBLE_FOCUS</td>
<td>accepts keyboard focus navigation if the widget can have the focus</td>
</tr>
<tr>
<td>COPIED_LABEL</td>
<td>the widget label is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>CLIP_CHILDREN</td>
<td>all drawing within this widget will be clipped (Fl_Group)</td>
</tr>
<tr>
<td>MENU_WINDOW</td>
<td>a temporary popup window, dismissed by clicking outside (Fl_Window)</td>
</tr>
<tr>
<td>TOOLTIP_WINDOW</td>
<td>a temporary popup, transparent to events, and dismissed easily (Fl_Window)</td>
</tr>
<tr>
<td>MODAL</td>
<td>a window blocking input to all other windows (Fl_Window)</td>
</tr>
<tr>
<td>NO_OVERLAY</td>
<td>window not using a hardware overlay plane (Fl_Menu_Window)</td>
</tr>
<tr>
<td>GROUP_RELATIVE</td>
<td>Reserved, not implemented. DO NOT USE.</td>
</tr>
<tr>
<td>COPIED_TOOLTIP</td>
<td>the widget tooltip is internally copied, its destruction is handled by the widget</td>
</tr>
<tr>
<td>FULLSCREEN</td>
<td>a fullscreen window (Fl_Window)</td>
</tr>
<tr>
<td>MAC_USE_ACCENTS_MENU</td>
<td>On the macOS platform, pressing and holding a key on the keyboard opens an accented-character menu window (Fl_Input_, Fl_Text_Editor)</td>
</tr>
<tr>
<td>NEEDS_KEYBOARD</td>
<td>set on touch screen devices if a widget needs a keyboard when it gets the focus. Reserved, not yet used in 1.4.0. See also Fl_Widget::needs_keyboard()</td>
</tr>
<tr>
<td>IMAGE_BOUND</td>
<td>binding the image to the widget will transfer ownership, so that the widget will delete the image when it is no longer needed</td>
</tr>
<tr>
<td>DEIMAGE_BOUND</td>
<td>bind the inactive image to the widget, so the widget deletes the image when it is no longer needed</td>
</tr>
<tr>
<td>USERFLAG3</td>
<td>reserved for 3rd party extensions</td>
</tr>
<tr>
<td>USERFLAG2</td>
<td>reserved for 3rd party extensions</td>
</tr>
<tr>
<td>USERFLAG1</td>
<td>reserved for 3rd party extensions</td>
</tr>
</tbody>
</table>

### 33.157.3 Constructor & Destructor Documentation

#### 33.157.3.1 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in x, y</td>
<td>the position of the widget relative to the enclosing window</td>
</tr>
<tr>
<td>in w, h</td>
<td>size of the widget in pixels</td>
</tr>
<tr>
<td>in label</td>
<td>optional text for the widget label</td>
</tr>
</tbody>
</table>

33.157.2 ~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.

33.157.4 Member Function Documentation

33.157.4.1 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()

33.157.4.2 active()

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()
33.157.4.3  active_r()

int Fl_Widget::active_r ( ) const

Returns whether the widget and all of its parents are active.

Return values

0  if this or any of the parent widgets are inactive

See also

active(), activate(), deactivate()

33.157.4.4  align() [1/2]

Fl_Align Fl_Widget::align ( ) const  [inline]

Gets the label alignment.

Returns

label alignment

See also

label(), align(Fl_Align), Fl_Align

33.157.4.5  align() [2/2]

void Fl_Widget::align ( Fl_Align alignment )  [inline]

Sets the label alignment.

This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER, which centers the label inside the widget.

Parameters

| in | alignment | new label alignment |

See also

align(), Fl_Align

33.157.4.6  argument() [1/2]

long Fl_Widget::argument ( ) const  [inline]

Gets the current user data (long) argument that is passed to the callback function.

Note

On platforms with sizeof(long) < sizeof(void*), particularly on Windows 64-bit platforms, this method can truncate stored addresses (void*) to the size of a long value. Use with care and only if you are sure that the stored user_data value fits in a long value because it was stored with argument(long) or another method using only long values. You may want to use user_data() instead.
33.157.4.7 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()
```

33.157.4.8 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.

33.157.4.9 as_group()

```cpp
virtual Fl_Group * Fl_Widget::as_group () [inline], [virtual]
```

Returns an Fl_Group pointer if this widget is an Fl_Group.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Group. If it returns non-NULL, then the widget in question is derived from Fl_Group, and you can use the returned pointer to access its children or other Fl_Group-specific methods.

Example:

```cpp
void my_callback (Fl_Widget *w, void *) {
    Fl_Group *g = w->as_group();
    if (g)
        printf ("This group has %d children\n", g->children());
    else
        printf ("This widget is not a group!\n");
}
```

Return values

```
NULL if this widget is not derived from Fl_Group.
```
Note
This method is provided to avoid dynamic_cast.

See also
Fl_Widget::as_window(), Fl_Widget::as_gl_window()

Reimplemented in Fl_Group.

### 33.157.4.10 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.

### 33.157.4.11 bind_deimage() [1/2]

void Fl_Widget::bind_deimage ( Fl_Image * img )

Sets the image to use as part of the widget label when in the inactive state.

Parameters

- **in **img the new image for the deactivated widget

Note
The image will be bound to the widget. When the widget is deleted, the image will be deleted as well.

See also
void deimage(Fl_Image* img)

### 33.157.4.12 bind_deimage() [2/2]

void Fl_Widget::bind_deimage ( int f ) [inline]

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
Parameters

|   | 1: mark the image as bound, 0: mark the image as managed by the user |

See also

deimage_bound(), bind_image()

### 33.157.4.13 bind_image() [1/2]

```cpp
void Fl_Widget::bind_image (Fl_Image * img)
```

Sets the image to use as part of the widget label when in the active state. The image will be bound to the widget. When the widget is deleted, the image will be deleted as well. Calling `bind_image()` with a new image will delete the old image if it was bound, and then set the new image, and bind that. If old and new image are the same, nothing happens. Calling `bind_image()` with NULL will delete the old image if it was bound and not set a new image.

Parameters

| in | `img` | the new image for the label |

See also

void image(Fl_Image* img)

### 33.157.4.14 bind_image() [2/2]

```cpp
void Fl_Widget::bind_image (int f) [inline]
```

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

Parameters

|   | 1: mark the image as bound, 0: mark the image as managed by the user |

See also

image_bound(), bind_deimage()

### 33.157.4.15 box() [1/2]

```cpp
Fl_Boxtype Fl_Widget::box ( ) const [inline]
```

Gets the box type of the widget.

Returns

the current box type

See also

box(Fl_Boxtype), Fl_Boxtype
33.157.4.16 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

33.157.4.17 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

33.157.4.18 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 |

33.157.4.19 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    |
33.157.4.20 callback() [4/5]

```c
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 |

33.157.4.21 callback() [5/5]

```c
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   |

33.157.4.22 changed()

```c
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)

33.157.4.23 clear_active()

```c
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()

### 33.157.4.24 clear_changed()

```cpp
void Fl_Widget::clear_changed () [inline]
```

Marks the value of the widget as unchanged.

See also

damaged(), set_changed()

### 33.157.4.25 clear_damage()

```cpp
void Fl_Widget::clear_damage ( uchar c = 0 ) [inline]
```

Clears or sets the damage flags. Damage flags are cleared when parts of the widget drawing is repaired. The optional argument \( c \) specifies the bits that are set after the call (default: 0) and not the bits that are cleared!

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()

### 33.157.4.26 clear_output()

```cpp
void Fl_Widget::clear_output () [inline]
```

Sets a widget to accept input.

See also

set_output(), output()

### 33.157.4.27 clear_visible()

```cpp
void Fl_Widget::clear_visible () [inline]
```

Hides the widget. You must still redraw the parent to see a change in the window. Normally you want to use the hide() method instead.

### 33.157.4.28 clear_visible_focus()

```cpp
void Fl_Widget::clear_visible_focus () [inline]
```

Disables keyboard focus navigation with this widget. Normally, all widgets participate in keyboard focus navigation.
33.157.4.29  color() [1/3]

\texttt{Fl\_Color \ F1\_Widget::color ( ) const [inline]}

Gets the background color of the widget.

Returns

current background color

See also

\texttt{color(Fl\_Color), color(Fl\_Color, Fl\_Color)}

33.157.4.30  color() [2/3]

\texttt{void \ F1\_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 \texttt{fl\_rgb\_color()}. The default for most widgets is \texttt{FL\_BACKGROUND\_COLOR}. Use \texttt{Fl::set\_color()} to redefine colors in the color map.

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{bg} \\
\hline
\end{tabular}

background color

See also

\texttt{color(), color(Fl\_Color, Fl\_Color), selection\_color(Fl\_Color)}

33.157.4.31  color() [3/3]

\texttt{void \ F1\_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

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textbf{bg} \\
\hline
background color
\hline
\textbf{in} & \textbf{sel} \\
\hline
selection color
\hline
\end{tabular}

See also

\texttt{color(unsigned), selection\_color(unsigned)}

33.157.4.32  color2() [1/2]

\texttt{Fl\_Color \ F1\_Widget::color2 ( ) const [inline]}

See also

\texttt{set\_visible\_focus(), visible\_focus(), visible\_focus(int)}
For back compatibility only.

**Deprecated** Use `selection_color()` instead.

### 33.157.4.33 color2() [2/2]

```cpp
void Fl_Widget::color2 ( unsigned a ) [inline]
```

For back compatibility only.

**Deprecated** Use `selection_color(unsigned)` instead.

### 33.157.4.34 contains()

```cpp
int Fl_Widget::contains ( const Fl_Widget * w ) const
```

Checks if `w` is a child of this widget.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>w</th>
<th>potential child widget</th>
</tr>
</thead>
</table>

**Returns**

Returns 1 if `w` is a child of this widget, or is equal to this widget. Returns 0 if `w` is NULL.

### 33.157.4.35 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**

<table>
<thead>
<tr>
<th>in</th>
<th>new_label</th>
<th>the new label text</th>
</tr>
</thead>
</table>

**See also**

- `label()`

### 33.157.4.36 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()`

### 33.157.4.37 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)`

### 33.157.4.38 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)`

### 33.157.4.39 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
33.157.4.40 deactivate()

```cpp
void Fl_Widget::deactivate ( )
```

Deactivates the widget.

Inactive widgets will be drawn "grayed out", e.g. with less contrast than the active widget. Inactive widgets will
not receive any keyboard or mouse button events. Other events (including FL_ENTER, FL_MOVE, FL_LEAVE,
FL_SHORTCUT, and others) will still be sent. A widget is only active if active() is true on it and all of its parents.
Changing this value will send FL_DEACTIVATE to the widget if active_r() is true.

Currently you cannot deactivate Fl_Window widgets.

See also

```cpp
activate(), active(), active_r()
```

33.157.4.41 default_callback()

```cpp
void Fl_Widget::default_callback ( 
    Fl_Widget * widget,
    void * data ) [static]
```

The default callback for all widgets that don't set a callback.
This callback function puts a pointer to the widget on the queue returned by Fl::readqueue(). This is the default for
all widgets if you don't set a callback.

You can avoid the overhead of this default handling if you set the callback to NULL explicitly.
Relying on the default callback and reading the callback queue with Fl::readqueue() is not recommended. If you
need a callback, you should set one with Fl_Widget::callback(Fl_Callback *cb, void *data) or one of its variants.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>the Fl_Widget given to the callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>user data associated with that callback</td>
</tr>
</tbody>
</table>

See also

```cpp
callback(), Fl::readqueue()
```

```cpp
do_callback(Fl_Widget *widget, void *data)
```

33.157.4.42 deimage() [1/4]

```cpp
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

33.157.4.43 deimage() [2/4]

```cpp
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.

See also

```cpp
damage(), clear_damage(uchar)
```
Returns

the current image for the deactivated widget

33.157.4.44  deimage() [3/4]

void Fl_Widget::deimage {
    Fl_Image & img
}

Sets the image to use as part of the widget label when in the inactive state.

Parameters

in  img  the new image for the deactivated widget

See also

void deimage(Fl_Image* img)

33.157.4.45  deimage() [4/4]

void Fl_Widget::deimage {
    Fl_Image * img
}

Sets the image to use as part of the widget label when in the inactive state.

Parameters

in  img  the new image for the deactivated widget

Note

The caller is responsible for making sure img is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

See also

void bind_deimage(Fl_Image* img)

33.157.4.46  deimage_bound()

int Fl_Widget::deimage_bound ( ) const [inline]

Returns whether the inactive image is managed by the widget.

Return values

0  if the image is not bound to the widget
1  if the image will be deleted when the widget is deleted

See also

image_bound(), bind_deimage()
33.157.4.47  do_callback() [1/3]

void Fl_Widget::do_callback {
    FL_CALLBACK_REASON reason = FL_REASON_UNKNOWN } [inline]

Calls the widget callback function with default arguments.
This is the same as calling
do_callback(this, user_data(), reason);

Parameters

| in  | reason | give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN |

See also

    callback()
    do_callback(Fl_Widget *widget, void *data, FL_CALLBACK_REASON reason), FL_CALLBACK_REASON

33.157.4.48  do_callback() [2/3]

void Fl_Widget::do_callback {
    Fl_Widget * widget,
    long arg,
    FL_CALLBACK_REASON reason = FL_REASON_UNKNOWN } [inline]

Calls the widget callback function with arbitrary arguments.

Parameters

| in  | widget | call the callback with widget as the first argument |
| in  | arg    | call the callback with arg as the user data (second) argument |
| in  | reason | give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN |

See also

    callback()
    do_callback(Fl_Widget *widget, void *data), FL_CALLBACK_REASON

33.157.4.49  do_callback() [3/3]

void Fl_Widget::do_callback {
    Fl_Widget * widget,
    void * arg = 0,
    FL_CALLBACK_REASON reason = FL_REASON_UNKNOWN }

Calls the widget callback function with arbitrary arguments.
All overloads of do_callback() call this method. It does nothing if the widget's callback() is NULL. It clears the widget's changed flag after the callback was called unless the callback is the default callback. Hence it is not necessary to call clear_changed() after calling do_callback() in your own widget's handle() method.
A reason must be set for widgets if different actions can trigger the same callback.

Note

   It is legal to delete the widget in the callback (i.e. in user code), but you must not access the widget in the handle() method after calling do_callback() if the widget was deleted in the callback. We recommend to use Fl_Widget_Tracker to check whether the widget was deleted in the callback.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>call the callback with widget as the first argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>use arg as the user data (second) argument</td>
</tr>
<tr>
<td>in</td>
<td>reason</td>
<td>give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN</td>
</tr>
</tbody>
</table>

See also

- default_callback()
- callback()
- class Fl_Widget_Tracker
- Fl::callback_reason()

#### 33.157.4.50 draw()

```cpp
virtual void Fl_Widget::draw () [pure virtual]
```

**draw()**

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()
```


#### 33.157.4.51 draw_focus() [1/3]

```cpp
void Fl_Widget::draw_focus ( ) const [inline], [protected]
```

**draw_focus()**

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`

#### 33.157.4.52 draw_focus() [2/3]

```cpp
void Fl_Widget::draw_focus ( ) const [inline], [protected]
```

**draw_focus()**

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`
33.157.4.53  draw_focus() [3/3]

void Fl_Widget::draw_focus (  
    Fl_Boxtype bt,  
    int X,  
    int Y,  
    int W,  
    int H,  
    Fl_Color bg ) const [protected]

Draws a focus box for the widget at the given position and size.
This method does nothing if

• the global option Fl::visible_focus() or

• the per-widget option visible_focus() is false (off).

This means that Fl_Widget::draw_focus() or one of the more specialized methods can be called without checking
these visible focus options.

Note

This method must only be called if the widget has the focus. This is not tested internally.

The boxtype bt is used to calculate the inset so the focus box is drawn inside the box borders.
The default focus box drawing color is black. The background color bg is used to determine a better visible color if
necessary by using fl_contrast() with the given background color.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bt</th>
<th>Boxttype that needs to be considered (frame width)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>X,Y,W,H</td>
<td>Bounding box</td>
</tr>
<tr>
<td>in</td>
<td>bg</td>
<td>Background color</td>
</tr>
</tbody>
</table>

See also

Fl_Widget::draw_focus()
Fl_Widget::draw_focus(Fl_Boxtype, int, int, int) const

33.157.4.54  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.

33.157.4.55  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

33.157.4.56  draw_label() [3/3]

void Fl_Widget::draw_label (  
    int X,  
    int Y,  

Generated by Doxygen
Draws the label in an arbitrary bounding box with an arbitrary alignment. Anybody can call this to force the label to draw anywhere.

33.157.4.57 \texttt{h()} [1/2]

\begin{verbatim}
int \texttt{Fl_Widget::h () const [inline]}
\end{verbatim}

Gets the widget height.

Returns

the height of the widget in pixels.

33.157.4.58 \texttt{h()} [2/2]

\begin{verbatim}
void \texttt{Fl_Widget::h ( int v ) [inline], [protected]}
\end{verbatim}

Internal use only.
Use \texttt{position(int,int), size(int,int)} or \texttt{resize(int,int,int,int)} instead.

33.157.4.59 \texttt{handle()}

\begin{verbatim}
int \texttt{Fl_Widget::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 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited \texttt{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

<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

\texttt{Fl_Event}


33.157.4.60 \texttt{hide()}

\begin{verbatim}
void \texttt{Fl_Widget::hide () [virtual]}
\end{verbatim}

Makes a widget invisible.
See also

`show(), visible(), visible_r()`

Reimplemented in `Fl_Browser`, `Fl_Double_Window`, `Fl_Gl_Window`, `Fl_Overlay_Window`, and `Fl_Window`.

### 33.157.4.61 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

### 33.157.4.62 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

### 33.157.4.63 image() [3/4]

```cpp
void Fl_Widget::image ( Fl_Image & img )
```

Sets the image to use as part of the widget label when in the active state.

Parameters

| in  | img | the new image for the label |

See also

`void image(Fl_Image* img)`

### 33.157.4.64 image() [4/4]

```cpp
void Fl_Widget::image ( Fl_Image * img )
```

Sets the image to use as part of the widget label when in the active state.

The caller is responsible for making sure `img` is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

Calling `image()` with a new image will delete the old image if it was bound, and set the new image without binding it.

If old and new are the same, the image will not be deleted, but it will be unbound.

Calling `image()` with NULL will delete the old image if it was bound and not set a new image.

Parameters

| in  | img | the new image for the label |
See also

bind_image(Fl_Image* img)

33.157.4.65 image_bound()

int Fl_Widget::image_bound ( ) const [inline]

Returns whether the image is managed by the widget.

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>if the image is not bound to the widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if the image will be deleted when the widget is deleted</td>
</tr>
</tbody>
</table>

See also
deimage_bound(), bind_image()

33.157.4.66 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()

33.157.4.67 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

<table>
<thead>
<tr>
<th>0</th>
<th>current label was assigned with label().</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>current label was assigned with copy_label().</td>
</tr>
</tbody>
</table>

33.157.4.68 label()[1/3]

const char * Fl_Widget::label ( ) const [inline]

Gets the current label text.
Returns
    a pointer to the current label text

See also
    label(const char *), copy_label(const char *)

33.157.4.69 label() [2/3]

void Fl_Widget::label (const char * text )

Sets the current label pointer.
The label is shown somewhere on or next to the widget. See Labels and Label Types for details about what can be
put in a label. The passed pointer is stored unchanged in the widget (the string is not copied), so if you need to set
the label to a formatted value, make sure the buffer is static, global, or allocated. The copy_label() method can be
used to make a copy of the label string automatically.

Parameters

See also
    copy_label()

33.157.4.70 label() [3/3]

void Fl_Widget::label (Fl_Labeltype a, const char * b ) [inline]

Shortcut to set the label text and type in one call.

See also
    label(const char *), labeltype(Fl_Labeltype)

33.157.4.71 label_shortcut()

unsigned int Fl_Widget::label_shortcut (const char * t ) [static]

Returns the Unicode value of the '&x' shortcut in a given text.
The given text t (usually a widget's label or a menu text) is searched for a '&x' shortcut label, and if found, the
Unicode value (code point) of the '&x' shortcut is returned.

Parameters

Returns
    Unicode (UCS-4) value of shortcut in t or 0.
Labelcolor() [1/2]

**Fl_Color Fl_Widget::labelcolor () const [inline]**

Gets the label color.
The default color is FL_FOREGROUND_COLOR.

Returns

the current label color

Labelcolor() [2/2]

**void Fl_Widget::labelcolor (Fl_Color c) [inline]**

Sets the label color.
The default color is FL_FOREGROUND_COLOR.

Parameters

| in | c | the new label color |

Labelfont() [1/2]

**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

Labelfont() [2/2]

**void Fl_Widget::labelfont (Fl_Font f) [inline]**

Sets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Parameters

| in | f | the new font for the label |


See also

Fl_Widget

33.157.4.76 labelsize() [1/2]

Fl_Fontsize Fl_Widget::labelsize ( ) const [inline]

Gets the font size in pixels.
The default size is 14 pixels.

Returns

the current font size

33.157.4.77 labelsize() [2/2]

void Fl_Widget::labelsize ( Fl_Fontsize pix ) [inline]

Sets the font size in pixels.

Parameters

| in | pix | the new font size |

See also

Fl_Fontsize labelsize()

33.157.4.78 labeltype() [1/2]

Fl_Labeltype Fl_Widget::labeltype ( ) const [inline]

Gets the label type.

Returns

the current label type.

See also

Fl_Labeltype

33.157.4.79 labeltype() [2/2]

void Fl_Widget::labeltype ( Fl_Labeltype a ) [inline]

Sets the label type.
The label type identifies the function that draws the label of the widget. This is generally used for special effects such as embossing or for using the label() pointer as another form of data such as an icon. The value FL_NORMAL_LABEL prints the label as plain text.

Parameters

| in | a | new label type |

Generated by Doxygen
See also

***Fl_Labeltype***

### 33.157.4.80 measure_label()

```cpp
void Fl_Widget::measure_label (  
    int & ww,  
    int & hh ) const [inline]
```

Sets width `ww` and height `hh` accordingly with the label size. Labels with images will return `w()` and `h()` of the image. This calls `fl_measure()` internally. For more information about the arguments `ww` and `hh` and word wrapping

See also

`fl_measure(const char*, int&, int&, int)`

### 33.157.4.81 needs_keyboard() [1/2]

```cpp
bool Fl_Widget::needs_keyboard ( ) const [inline]
```

Returns whether this widget needs a keyboard.

Returns

true or false

See also

`needs_keyboard(bool)`

### 33.157.4.82 needs_keyboard() [2/2]

```cpp
void Fl_Widget::needs_keyboard (  
    bool needs ) [inline]
```

Sets whether this widget needs a keyboard. Set this on touch screen devices if a widget needs a keyboard when it gets the focus.

**Note**

This flag can be set but is not yet **used** in FLTK 1.4.0. It is intended to be used in the future on real touch devices.

**Parameters**

| in needs | Set this to true or false |

### 33.157.4.83 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.

---

*Generated by Doxygen*
Return values

0  if the widget is used for input and output

See also

set_output(), clear_output()

33.157.4.84  parent() [1/2]

Fl_Group * Fl_Widget::parent () const [inline]
Returns a pointer to the parent widget.
Usually this is a Fl_Group or Fl_Window.

Return values

NULL  if the widget has no parent

See also

Fl_Group::add(Fl_Widget*)

33.157.4.85  parent() [2/2]

void Fl_Widget::parent ( Fl_Group * p ) [inline]
Internal use only - "for hacks only".
It is **STRONGLY recommended** not to use this method, because it short-circuits Fl_Group's normal widget adding and removing methods, if the widget is already a child widget of another Fl_Group.
Use Fl_Group::add(Fl_Widget*) and/or Fl_Group::remove(Fl_Widget*) instead.

33.157.4.86  position()

void Fl_Widget::position ( int X, int Y ) [inline]
Repositions the window or widget.
position(X, Y) is a shortcut for resize(X, Y, w(), h()).

Parameters

| in | X,Y | new position relative to the parent window |

See also

resize(int,int,int), size(int,int)

33.157.4.87  redraw()

void Fl_Widget::redraw ( )
Schedules the drawing of the widget.
Marks the widget as needing its draw() routine called.
### 33.157.4.88 redraw_label()

```cpp
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.

### 33.157.4.89 resize()

```cpp
void Fl_Widget::resize(int x, int y, int w, int h) [virtual]
```

Changes the size or position of the widget. This is a virtual function so that the widget may implement its own handling of resizing. The default version does not call the `redraw()` method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position. Some window managers under X11 call `resize()` a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations. `position(X, Y)` is a shortcut for `resize(X, Y, w(), h())`, and `size(W, H)` is a shortcut for `resize(x(), y(), W, H)`. Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>new position relative to the parent window</td>
</tr>
<tr>
<td>w,h</td>
<td>new size</td>
</tr>
</tbody>
</table>

See also

- `position(int, int)`, `size(int, int)`


### 33.157.4.90 selection_color() [1/2]

```cpp
Fl_Color Fl_Widget::selection_color() const [inline]
```

Gets the selection color.

Returns

the current selection color

See also

- `selection_color(Fl_Color)`, `color(Fl_Color, Fl_Color)`

### 33.157.4.91 selection_color() [2/2]

```cpp
void Fl_Widget::selection_color(Fl_Color a) [inline]
```

Sets the selection color.

The selection color is defined for Forms compatibility and is usually used to color the widget when it is selected, although some widgets use this color for other purposes. You can set both colors at once with `color(Fl_Color bg, Fl_Color sel)`.
Parameters

```{lst-c}
in a the new selection color
```

See also

```{lst-c}
selection_color(), color(Fl_Color, Fl_Color)
```

### 33.157.4.92 set_active()

```{lst-c}
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

```{lst-c}
activate()
```

### 33.157.4.93 set_changed()

```{lst-c}
void Fl_Widget::set_changed ( ) [inline]
```
Marks the value of the widget as changed.

See also

```{lst-c}
changed(), clear_changed()
```

### 33.157.4.94 set_output()

```{lst-c}
void Fl_Widget::set_output ( ) [inline]
```
Sets a widget to output only.

See also

```{lst-c}
output(), clear_output()
```

### 33.157.4.95 set_visible()

```{lst-c}
void Fl_Widget::set_visible ( ) [inline]
```
Makes the widget visible. You must still redraw the parent widget to see a change in the window. Normally you want to use the show() method instead.

### 33.157.4.96 set_visible_focus()

```{lst-c}
void Fl_Widget::set_visible_focus ( ) [inline]
```
Enables keyboard focus navigation with this widget. Note, however, that this will not necessarily mean that the widget will accept focus, but for widgets that can accept focus, this method enables it if it has been disabled.

See also

```{lst-c}
visible_focus(), clear_visible_focus(), visible_focus(int)
```
33.157.4.97 shortcut_label() [1/2]

```cpp
int Fl_Widget::shortcut_label ( ) const [inline]
```

Returns whether the widget's label uses '&' to indicate shortcuts.

See also

```cpp
void shortcut_label(int value)
```

33.157.4.98 shortcut_label() [2/2]

```cpp
void Fl_Widget::shortcut_label ( 
    int value ) [inline]
```

Sets whether the widget's label uses '&' to indicate shortcuts.

By default, all objects of classes Fl_Menu_ (and derivatives), Fl_Button (and derivatives), Fl_Text_Display, Fl_Value_Input, and Fl_Input_ (and derivatives) use character '&' in their label, unless '&' is repeated, to indicate shortcuts: '&' does not appear in the drawn label, the next character after '&' in the label is drawn underlined, and typing this character triggers the corresponding menu window, button, or other widget. If the label contains 2 consecutive '&', only one is drawn and the next character is not underlined and not used as a shortcut. If value is set to 0, all these labels don't process character '&' as indicating a shortcut: '&' is drawn in the label, the next character is not underlined and does not define a shortcut.

33.157.4.99 show()

```cpp
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

```cpp
hide(), visible(), visible_r()
```


33.157.4.100 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 |

---
33.157.4.101  take_focus()

int Fl_Widget::take_focus ( )

Gives the widget the keyboard focus. Tries to make this widget be the Fl::focus() widget, by first sending it an FL_FOCUS event, and if it returns non-zero, setting Fl::focus() to this widget. You should use this method to assign the focus to a widget.

Returns

true if the widget accepted the focus.

33.157.4.102  takesevents()

unsigned int Fl_Widget::takesevents ( ) const [inline]

Returns if the widget is able to take events. This is the same as (active() && 'output() && visible()) but is faster.

Return values

0 if the widget takes no events

33.157.4.103  test_shortcut() [1/2]

int Fl_Widget::test_shortcut ( )

Returns true if the widget's label contains the entered '&x' shortcut. This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The widget's label is searched for a '&x' shortcut, and if found, this is compared with the entered key value. Fl::event_text() is used to get the entered key value.

Returns

true, if the entered text matches the widget's '&x' shortcut, false (0) otherwise.

Note

Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

33.157.4.104  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

Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

33.157.4.105 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*)

33.157.4.106 tooltip() [2/2]

void Fl_Widget::tooltip ( const char * text )

Sets the current tooltip text.

Sets a string of text to display in a popup tooltip window when the user hovers the mouse over the widget. The string is not copied, so make sure any formatted string is stored in a static, global, or allocated buffer. If you want a copy made and managed for you, use the copy_tooltip() method, which will manage the tooltip string automatically. If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group's tooltip instead. To avoid this behavior, you can set the child's tooltip to an empty string ("").

Parameters

| [in] text | New tooltip text (no copy is made) |

See also

copytooltip(const char*), tooltip()

33.157.4.107 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()

33.157.4.108  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)

33.157.4.109  type() [1/2]

`uchar Fl_Widget::type ( ) const [inline]`

Gets the widget type.

Returns the widget type value, which gives some information about the derived widget class to which the object belongs. Noticeably, the condition `type() >= FL_WINDOW` indicates a widget is an `Fl_Window` or derived object.

33.157.4.110  type() [2/2]

`void Fl_Widget::type ( uchar t ) [inline]`

Sets the widget type.

See also

type()

33.157.4.111  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
Sets the user data for this widget.
Sets the new user data (void *) argument that is passed to the callback function.
33.157.4.113  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()

33.157.4.114  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()

33.157.4.115  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()

33.157.4.116  visible_r()

`int Fl_Widget::visible_r ( ) const`

Returns whether a widget and all its parents are visible.
Return values

| 0 | if the widget or any of its parents are invisible. |

See also

- `show()`, `hide()`, `visible()`

### 33.157.4.117 `w()` [1/2]

```cpp
int Fl_Widget::w ( ) const [inline]
```

Gets the widget width.

Returns

- the width of the widget in pixels.

### 33.157.4.118 `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)` instead.

### 33.157.4.119 `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)`, `Fl_When`, `do_callback()`, `Fl::callback_reason()`

### 33.157.4.120 `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_WHEN_CLOSED**: If the user requests that the widget is closed, the callback is called with **FL_REASON_CLOSED**. The **Fl_Tabs** widget checks this flag on its children to determine whether to display a close button on the tab of that widget.

**Fl_Widget::when()** is a set of bitflags used by subclasses of **Fl_Widget** to decide when to do the callback. If the value is zero then the callback is never done. Other values are described in the individual widgets. This field is in the base class so that you can scan a panel and **do_callback()** on all the ones that don't do their own callbacks in response to an "OK" button.

**Parameters**

```
in  i  set of flags
```

**See also**

- **Fl_When**, **do_callback()**, **Fl::callback_reason()**

### 33.157.4.121 window()

**Fl_Window * Fl_Widget::window ( ) const**

Returns a pointer to the nearest parent window up the widget hierarchy. This will return sub-windows if there are any, or the parent window if there's no sub-windows. If this widget IS the top-level window, NULL is returned.

**Return values**

- **NULL** if no window is associated with this widget.

**Note**

- for an **Fl_Window** widget, this returns its parent window (if any), not this window.

**See also**

- **top_window()**

### 33.157.4.122 x() [1/2]

**int Fl_Widget::x ( ) const [inline]**

Gets the widget position in its window.

**Returns**

- the x position relative to the window

### 33.157.4.123 x() [2/2]

**void Fl_Widget::x ( int v ) [inline], [protected]**

Internal use only. Use **position(int,int)**, **size(int,int)** or **resize(int,int,int,int)** instead.
33.157.4.124 y() [1/2]

int Fl_Widget::y () const [inline]
Gets the widget position in its window.

Returns

the y position relative to the window.

33.157.4.125 y() [2/2]

void Fl_Widget::y (int v) [inline], [protected]

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,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

33.158 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
  Fl_Copy_Surface Fl_EPS_File_Surface Fl_Image_Surface Fl_Paged_Device Fl_SVG_File_Surface Fl_PostScript_File_Device Fl_Printer
```

Public Member Functions

- void **draw**(Fl_Widget *widget, int delta_x=0, int delta_y=0)
  Draws the widget on the drawing surface.
- virtual void **draw_decorated_window**(Fl_Window *win, int x_offset=0, int y_offset=0)
  Draws a window with its title bar and frame if any.
- virtual void **origin**(int *x, int *y)
  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

33.158.1 Detailed Description

A surface on which any FLTK widget can be drawn.

33.158.2 Constructor & Destructor Documentation

33.158.2.1 Fl_Widget_Surface()

Fl_Widget_Surface::Fl_Widget_Surface (Fl_Graphics_Driver * d) [protected]

The constructor.

Parameters

d can be nul.

33.158.3 Member Function Documentation

33.158.3.1 draw()

void Fl_Widget_Surface::draw (Fl_Widget * widget,
               int delta_x = 0,
               int delta_y = 0)

Draws the widget on the drawing surface.

The widget's position on the surface is determined by the last call to origin() and by the optional delta_x and delta_y arguments. Its dimensions are in points unless there was a previous call to scale().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>widget</th>
<th>Any FLTK widget (e.g., standard, custom, window).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delta_x, delta_y</td>
<td>Optional horizontal and vertical offsets for positioning the widget top left relatively to the current origin of graphics.</td>
</tr>
</tbody>
</table>
33.158.3.2 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.

33.158.3.3 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**

| in    | x, y | Horizontal and vertical positions in the drawing surface of the desired origin of graphics. |
|-------|------|-------------------------------------------------------------------------------------------------


33.158.3.4 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. |
|-------|------|-------------------------------------------------------------------------------------------------


33.158.3.5 print_window_part()

```cpp
void Fl_Widget_Surface::print_window_part (  
    Fl_Window * win,  
    int x,  
    int y,  
    int w,  
    int h,  
    int delta_x = 0,  
    int delta_y = 0 )
```

Draws a rectangular part of an on-screen window.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>win</code></td>
<td>The window from where to capture. Can be an <code>Fl_Gl_Window</code>. Sub-windows that intersect the rectangle are also captured.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>The rectangle left</td>
</tr>
<tr>
<td><code>y</code></td>
<td>The rectangle top</td>
</tr>
<tr>
<td><code>w</code></td>
<td>The rectangle width</td>
</tr>
<tr>
<td><code>h</code></td>
<td>The rectangle height</td>
</tr>
<tr>
<td><code>delta_x, delta_y</code></td>
<td>Optional horizontal and vertical offsets from current graphics origin where to draw the top left of the captured rectangle.</td>
</tr>
</tbody>
</table>

### 33.158.3.6 printable_rect()

```cpp
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


### 33.158.3.7 translate()

```cpp
void Fl_Widget_Surface::translate ( int x, int y ) [virtual]
```

Translates the current graphics origin accounting for the current rotation. Each `translate()` call must be matched by an `untranslate()` call. Successive `translate()` calls add up their effects.


### 33.158.3.8 untranslate()

```cpp
void Fl_Widget_Surface::untranslate ( void ) [virtual]
```

Undoes the effect of a previous `translate()` call.


The documentation for this class was generated from the following files:

- `Fl_Widget_Surface.H`
- `Fl_Widget_Surface.cxx`

### 33.159 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.

```cpp
#include <Fl.H>
```
Public Member Functions

- **int deleted ()**
  Returns 1, if the watched widget has been deleted.

- **int exists ()**
  Returns 1, if the watched widget exists (has not been deleted).

- **Fl_Widget_Tracker (Fl_Widget *w)**
  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.

### 33.159.1 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();
  }
}
```

### 33.159.2 Member Function Documentation

#### 33.159.2.1 deleted()

```c++
int Fl_Widget_Tracker::deleted () [inline]
```

Returns 1, if the watched widget has been deleted.

This is a convenience method. You can also use something like

```c++
if (wp.widget() == 0) // ...
```

where `wp` is an Fl_Widget_Tracker object.

#### 33.159.2.2 exists()

```c++
int Fl_Widget_Tracker::exists () [inline]
```

Returns 1, if the watched widget exists (has not been deleted).

This is a convenience method. You can also use something like

```c++
if (wp.widget() != 0) // ...
```

where `wp` is an Fl_Widget_Tracker object.
33.159.2.3  widget()

`Fl_Widget * Fl_Widget_Tracker::widget ( ) [inline]`

Returns a pointer to the watched widget.
This pointer is NULL, if the widget has been deleted.
The documentation for this class was generated from the following files:

- Fl.H
- Fl.cxx

### 33.160  Fl_Window Class Reference

This widget produces an actual window.

```
#include <Fl_Window.H>
```

Inheritance diagram for Fl_Window:

```
Fl_Window
   |     |
   v     v
Fl_Group
   |     |
   v     v
Fl_Widget
   |     |
   v     v
Fl_Double_Window
   |     |
   v     v
Fl_Gl_Window
   |     |
   v     v
Fl_Single_Window
   |     |
   v     v
Fl_Cairo_Window
   |     |
   v     v
Fl_Overlay_Window
   |     |
   v     v
Fl_Glut_Window
   |     |
   v     v
Fl_Menu_Window
```

#### Public Types

- `typedef struct HICON__ * HICON`

#### Public Member Functions

- `virtual class Fl_Double_Window * as_double_window ( )`
  
  Returns non-null if this is an Fl_Double_Window object.
- `virtual class Fl_Overlay_Window * as_overlay_window ( )`
  
  Returns non-null if this is an Fl_Overlay_Window object.
- `Fl_Window const * as_window ( ) const FL_OVERRIDE`
  
  Returns an Fl_Window pointer if this widget is an Fl_Window.
- `unsigned int border () const`
  
  Returns whether the window possesses a border.
- `void border (int b)`
  
  Sets whether or not the window manager border is around the window.
- `void clear_border ()`
  
  Fast inline function to turn the window manager border off.
- `void clear_modal_states ()`
  
  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- `void copy_label (const char *a)`
  
  Sets the window titlebar label to a copy of a character string.
- `void cursor (const Fl_RGB_Image *, int, int)`
  
  Changes the cursor for this window using the provided image as cursor's shape.
• void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
• void cursor (Fl_Cursor)
  Changes the cursor for this window.
• int decorated_h () const
  Returns the window height including any window title bar and any frame added by the window manager.
• int decorated_w () const
  Returns the window width including any frame added by the window manager.
• void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)
  For back compatibility only.
• void default_cursor (Fl_Cursor)
  Sets the default window cursor.
• Fl_Window (int w, int h, const char ∗title=0)
  Creates a window from the given width w, height h, and title.
• Fl_Window (int x, int y, int w, int h, const char ∗title=0)
  Creates a window from the given position (x, y), size (w, h) and title.
• void free_position ()
  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
• void fullscreen ()
  Makes the window completely fill one or more screens, without any window manager border visible.
• unsigned int fullscreen_active () const
  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
• void fullscreen_off ()
  Turns off any side effects of fullscreen()
• void fullscreen_off (int X, int Y , int W, int H)
  Turns off any side effects of fullscreen() and does resize(x,y,w,h).
• void fullscreen_screens (int top, int bottom, int left, int right)
  Sets which screens should be used when this window is in fullscreen mode.
• int handle (int) FL_OVERRIDE
  Handles the specified event.
• void hide () FL_OVERRIDE
  Removes the window from the screen.
• void hotspot (const Fl_Widget &p, int offscreen=0)
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
• void hotspot (const Fl_Widget ∗, int offscreen=0)
  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
• void hotspot (int x, int y, int offscreen=0)
  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
• const void ∗ icon () const
  Gets the current icon window target dependent data.
• void icon (const Fl_RGB_Image ∗)
  Sets or resets a single window icon.
• void icon (const void ∗ic)
  Platform-specific method to set the window icon usable on Windows and X11 only.
• void iconize ()
  Iconifies the window.
• const char ∗ iconlabel () const
  See void Fl_Window::iconlabel(const char ∗)
• void **iconlabel** (const char ∗)  
  Sets the icon label.
• void **icons** (const FL_RGB_Image ∗[], int)  
  Sets the window icons.
• void **icons** (HICON big_icon, HICON small_icon)  
  Sets the window icons using HICON handles (Windows platform only).
• const char ∗ **label** () const  
  See void Fl_Window::label(const char ∗)
• void **label** (const char ∗)  
  Sets the window title bar label.
• void **label** (const char ∗ label, const char ∗ iconlabel)  
  Sets the icon label.
• void **make_current** ()  
  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
• unsigned int **menu_window** () const  
  Returns true if this window is a menu window.
• unsigned int **modal** () const  
  Returns true if this window is modal.
• unsigned int **non_modal** () const  
  Returns true if this window is modal or non-modal.
• fl_uintptr_t **os_id** ()  
  Returns a platform-specific identification of a shown window, or 0 if not shown.
• unsigned int **override** () const  
  Returns non zero if OVERRIDE flag is set, 0 otherwise.
• void **resize** (int X, int Y, int W, int H) FL_OVERRIDE  
  Changes the size and position of the window.
• int **screen_num** ()  
  The number of the screen containing the mapped window.
• void **screen_num** (int screen_num)  
  Set the number of the screen where to map the window.
• void **set_menu_window** ()  
  Marks the window as a menu window.
• void **set_modal** ()  
  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
• void **set_non_modal** ()  
  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
• void **set_override** ()  
  Activates the flags NOBORDER|OVERRIDE.
• void **set_tooltip_window** ()  
  Marks the window as a tooltip window.
• const FL_Image ∗ **shape** ()  
  Returns the image controlling the window shape or NULL.
• void **shape** (const FL_Image &b)  
  Set the window's shape with an FL_Image.
• void **shape** (const FL_Image =img)  
  Assigns a non-rectangular shape to the window.
• void **show** () FL.Override  
  Puts the window on the screen.
• void show (int argc, char **argv)
  Puts the window on the screen with show() and parses command-line arguments.

• int shown ()
  Returns non-zero if show() has been called (but not hide() ).

• void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)
  Sets the allowable range the user can resize this window to.

• unsigned int tooltip_window () const
  Returns true if this window is a tooltip window.

• void 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 () FL_OVERRIDE
  Draws the widget.

• virtual void flush ()
  Forces the window to be drawn, this window is also made current and calls draw().

• int force_position () const
  Returns the internal state of the window's FORCE_POSITION flag.

• void force_position (int force)
Sets an internal flag that tells FLTK and the window manager to honor position requests.

- void free_icons ()
  Deletes all icons previously attached to the window.

- int is_resizable ()
  Protected method to determine whether a window is resizable.

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

33.160.1 Detailed Description

This widget produces an actual window. This can either be a main window, with a border and title and all the window management controls, or a "subwindow" inside a window. This is controlled by whether or not the window has a parent(). Once you create a window, you usually add children Fl_Widget's to it by using window->add(child) for each new widget. See Fl_Group for more information on how to add and remove children.

There are several subclasses of Fl_Window that provide double-buffering, overlay, menu, and OpenGL support. The window's callback is done if the user tries to close a window using the window manager and Fl::modal() is zero or equal to the window. Fl_Window has a default callback that calls Fl_Window::hide(). Callback reasons can be FL_REASON_CANCELLED if the Escape key was pressed, or FL_REASON_CLOSED when the close button is clicked. FL_WHEN... flags are ignored.

33.160.2 Constructor & Destructor Documentation

33.160.2.1 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)

Generated by Doxygen
33.160.2.2 Fl_Window() [2/2]

Fl_Window::Fl_Window (  
    int x,  
    int y,  
    int w,  
    int h,  
    const char * title = 0 )

Creates a window from the given position (x, y), size (w, h) and title.
On a multi-screen system, the values computed by Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n) can be used
to discover the coordinates of the area of screen #n. When these screens have various scale factor values, an (x, y) pair may not be enough to specify the targeted screen for the window, because the same (x,y) pair can belong to several screens. In that situation, a call to Fl_Window::screen_num(int) is to be used to identify unambiguously the targeted screen.

See also

    Fl_Window(int w, int h, const char *title)  
    Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n)

Note

Under Wayland, it's generally not possible for the client app to control the position of a window in the system. It's only possible to specify on what screen should the compositor place a fullscreen window.

33.160.2.3 ~Fl_Window()

Fl_Window::~Fl_Window ( ) [virtual]
The destructor also deletes all the children.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Window and all of its children can be automatic (local) variables, but you must declare the Fl_Window first so that it is destroyed last.

33.160.3 Member Function Documentation

33.160.3.1 as_double_window()

virtual class Fl_Double_Window * Fl_Window::as_double_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Double_Window object.
Reimplemented in Fl_Double_Window.

33.160.3.2 as_overlay_window()

virtual class Fl_Overlay_Window * Fl_Window::as_overlay_window ( ) [inline], [virtual]
Return non-null if this is an Fl_Overlay_Window object.
Reimplemented in Fl_Overlay_Window.

33.160.3.3 as_window() [1/2]

Fl_Window const * Fl_Window::as_window ( ) const [inline], [virtual]
Reimplemented from Fl_Widget.
33.160.3.4  as_window() [2/2]

Fl_Window * Fl_Window::as_window() [inline], [virtual]

Returns an Fl_Window pointer if this widget is an Fl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer to access its children or other Fl_Window-specific methods.

Return values

- **NULL** if this widget is not derived from Fl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

- Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented from Fl_Widget.

33.160.3.5  border()

void Fl_Window::border (int b)

Sets whether or not the window manager border is around the window.

The default value is true. *With some X window managers, this does not work after show() has been called.*

33.160.3.6  clear_border()

void Fl_Window::clear_border () [inline]

Fast inline function to turn the window manager border off.

It only works before show() is called.

33.160.3.7  clear_modal_states()

void Fl_Window::clear_modal_states () [inline]

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

Note that there are three states for a window: modal, non-modal, and normal.

You can not change the "modality" of a window whilst it is shown, so it is necessary to first hide() the window, change its "modality" as required, then re-show the window for the new state to take effect.

This method can also be used to change a "modal" window into a "non-modal" one. On several supported platforms, the "modal" state over-rides the "non-modal" state, so the "modal" state must be cleared before the window can be set into the "non-modal" state. In general, the following sequence should work:

```
win->hide();
win->clear_modal_states();
// Set win to new state as desired, or leave "normal", e.g...
win->set_non_modal();
win->show();
```

Note

Under some window managers, the sequence of hiding the window and changing its modality will often cause it to be re-displayed at a different position when it is subsequently shown. This is an irritating feature but appears to be unavoidable at present. As a result we would advise to use this method only when absolutely necessary.

See also

- void set_modal(), void set_non_modal()
33.160.3.8  current()

Fl_Window * Fl_Window::current ( )  [static]
Returns the last window that was made current.

See also

Fl_Window::make_current()

33.160.3.9  cursor()  [1/3]

void Fl_Window::cursor ( const Fl_RGB_Image * image, int hotx, int hoty )
Changes the cursor for this window using the provided image as cursor’s shape.
The window must be show()’n for this function to have any effect. This always calls the system. If you are changing
the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor
is different.
The default cursor will be used if the provided image cannot be used as a cursor.

Parameters

<table>
<thead>
<tr>
<th>image</th>
<th>Sets the cursor size and shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>hotx,hoty</td>
<td>Sets the cursor’s active location relatively to top-left of image when clicking</td>
</tr>
</tbody>
</table>

See also

cursor(Fl_Cursor), default_cursor()

33.160.3.10  cursor()  [2/3]

void Fl_Window::cursor ( Fl_Cursor c, Fl_Color , Fl_Color = FL_WHITE )
For back compatibility only.
Same as Fl_Window::cursor(Fl_Cursor)

33.160.3.11  cursor()  [3/3]

void Fl_Window::cursor ( Fl_Cursor c )
Changes the cursor for this window.
The window must be show()’n for this function to have any effect. This always calls the system. If you are changing
the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor
is different.
The type Fl_Cursor is an enumeration defined in <FL/Enumerations.H>.

See also

cursor(const Fl_RGB_Image*, int, int), default_cursor()

33.160.3.12  decorated_h()

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 \texttt{h()} if applied to a subwindow, or if window is not yet mapped.

\textbf{Note}\n
Under X11, FLTK is able to compute the size of window titlebars and borders only if these decoration elements are strictly X11-based. When that's not the case, \texttt{decorated\_h()} returns the same value as \texttt{h()} and \texttt{decorated\_w()} as \texttt{w()}, and FLTK cannot access window decorations.

Under X11 again, the values returned by \texttt{decorated\_h()} and \texttt{decorated\_w()} may not be reliable \textbf{during a resize operation}. The size of decoration elements of a window is best computed when the window is first mapped.

\subsection*{33.160.3.13 \texttt{decorated\_w()}}

\begin{verbatim}
int Fl_Window::decorated\_w ( ) const
\end{verbatim}

Returns the window width including any frame added by the window manager. Same as \texttt{w()} if applied to a subwindow, or if window is not yet mapped.

See also

\texttt{decorated\_h()}.

\subsection*{33.160.3.14 \texttt{default\_cursor() [1/2]}}

\begin{verbatim}
void Fl_Window::default\_cursor ( Fl_Cursor c,
                                Fl_Color ,
                                Fl_Color = FL\_WHITE )
\end{verbatim}

For back compatibility only.

same as \texttt{Fl\_Window::default\_cursor(Fl\_Cursor)}

\subsection*{33.160.3.15 \texttt{default\_cursor() [2/2]}}

\begin{verbatim}
void Fl_Window::default\_cursor ( Fl_Cursor c )
\end{verbatim}

Sets the default window cursor.

This is the cursor that will be used after the mouse pointer leaves a widget with a custom cursor set.

See also

\texttt{cursor(const Fl\_RGB\_Image\*, int, int), default\_cursor()}

\subsection*{33.160.3.16 \texttt{default\_icon()}}

\begin{verbatim}
void Fl_Window::default\_icon ( const Fl\_RGB\_Image \* icon ) [static]
\end{verbatim}

Sets a single default window icon.

If \texttt{icon} is NULL the current default icons are removed.

\textbf{Parameters}\n
\begin{verbatim}
in icon default icon for all windows subsequently created or NULL
\end{verbatim}
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.

### 33.160.3.17 default_icons() [1/2]

```cpp
void Fl_Window::default_icons (const Fl_RGB_Image * icons[], int count) [static]
```

Sets the default window icons.

The default icons are used for all windows that don’t have their own icons set before `show()` is called. You can change the default icons whenever you want, but this only affects windows that are created (and shown) after this call.

The given images in `icons` are copied. You can use a local variable or free the images immediately after this call.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>icons</code></td>
<td>default icons for all windows subsequently created</td>
</tr>
<tr>
<td><code>count</code></td>
<td>number of images in <code>icons</code>. Set to 0 to remove the current default icons</td>
</tr>
</tbody>
</table>

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::icon(const Fl_RGB_Image *)`
- `Fl_Window::icons(const Fl_RGB_Image *[ ], int)`

### 33.160.3.18 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>big_icon</code></td>
<td>default large icon for all windows subsequently created</td>
</tr>
<tr>
<td><code>small_icon</code></td>
<td>default small icon for all windows subsequently created</td>
</tr>
</tbody>
</table>
33.160 Fl_Window Class Reference

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)

33.160.3.19 default_size_range()

```c
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:**

```c
Fl_Window win(400, 400);
win.resizable(win);
// win.size_range(100, 100, 0, 0);
The minimum size of the resizable is 100, hence the minimum size of the total window is also 100 in both directions.
Fl_Window win(400, 400);
Fl_Box box(20, 20, 360, 360);
win.resizable(box);
// win.size_range(140, 140, 0, 0);
The calculated minimum width and height would be 20 + 100 + 20 in both dimensions.
Fl_Window win(400, 400);
Fl_Box box(200, 0, 500, 300); // note: width 500 too large: clipped
win.resizable(box);
// win.size_range(300, 200, 0, 0);
The width of the resizable is clipped to 200, hence the minimum size of the total window is also 200 (fix) + 100 (min. resizable) in x direction. The minimum value in y direction is 100 (resizable) + 100 (fixed part).
The calculation is based on clipping the resizable widget to the window area to prevent programming errors and the assumption that the resizable widget can be shrunk to 100x100 or its original size, whichever is smaller. If this is not what you want, please use Fl_Window::size_range() explicitly so you can set any appropriate range.
```
33.160.3.20  default_xclass() [1/2]

const char * Fl_Window::default_xclass() const [static]
Returns the default xclass.

See also

   Fl_Window::default_xclass(const char *)

33.160.3.21  default_xclass() [2/2]

void Fl_Window::default_xclass(const char * xc) const [static]
Sets the default window xclass.
The default xclass is used for all windows that don't have their own xclass set before show() is called. You can change the default xclass whenever you want, but this only affects windows that are created (and shown) after this call.
The given string xc is copied. You can use a local variable or free the string immediately after this call.
If you don't call this, the default xclass for all windows will be "FLTK". You can reset the default xclass by specifying NULL for xc.
If you call Fl_Window::xclass(const char *) for any window, then this also sets the default xclass, unless it has been set before.

Parameters

| in  | xc | default xclass for all windows subsequently created |

See also

   Fl_Window::xclass(const char *)

33.160.3.22  draw()

void Fl_Window::draw() const [protected], [virtual]
Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method from within your own draw() method, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

   Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
   s->draw();
   // calls Fl_Scrollbar::draw()

Reimplemented from Fl_Group.
Reimplemented in Fl_Glut_Window.

33.160.3.23  flush()

void Fl_Window::flush() const [protected], [virtual]
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented in Fl_Double_Window, Fl_Gl_Window, and Fl_Overlay_Window.

33.160.3.24  force_position() [1/2]

int Fl_Window::force_position() const [inline], [protected]
Returns the internal state of the window's FORCE_POSITION flag.
Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if flag is set</td>
</tr>
<tr>
<td>0</td>
<td>otherwise</td>
</tr>
</tbody>
</table>

See also

force_position(int)

### 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**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>force</td>
</tr>
</tbody>
</table>

#### free_icons()

```cpp
void Fl_Window::free_icons ( ) [protected]
```

Deletes all icons previously attached to the window.

See also

Fl_Window::icons(const Fl_RGB_Image *icons[], int count)

#### free_position()

```cpp
void Fl_Window::free_position ( ) [inline]
```

Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window. This is for Forms compatibility only.

**Deprecated** please use force_position(0) instead

#### 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

Fl_Window::fullscreen_screens()
33.160.3.29  fullscreen_screens()

void Fl_Window::fullscreen_screens {
    int top,
    int bottom,
    int left,
    int right
}

Sets which screens should be used when this window is in fullscreen mode. The window will be resized to the top of the screen with index top, the bottom of the screen with index bottom, etc. If this method is never called, or if any argument is < 0, then the window will be resized to fill the screen it is currently on.

See also

    void Fl_Window::fullscreen()

33.160.3.30  handle()

int Fl_Window::handle (  
    int event  ) [virtual]

Handles the specified event. You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget. When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise. Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>the kind of event received</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th></th>
<th>if the event was not used or understood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>if the event was used and can be deleted</td>
</tr>
</tbody>
</table>

See also

    Fl_Event

Reimplemented from Fl_Group.
Reimplemented in Fl_Glut_Window.

33.160.3.31  hide()

void Fl_Window::hide ( ) [virtual]

Removes the window from the screen. If the window is already hidden or has not been shown then this does nothing and is harmless.
Reimplemented from Fl_Widget.

33.160.3.32  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()

### icon() [1/3]

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

Gets the current icon window target dependent data.

**Deprecated** in 1.3.3

### icon() [2/3]

```cpp
void Fl_Window::icon ( 
    const Fl_RGB_Image * icon )
```

Sets or resets a single window icon.

A window icon can be changed while the window is shown, but this may be platform and/or window manager dependent. To be sure that the window displays the correct window icon you should always set the icon before the window is shown.

If a window icon has not been set for a particular window, then the default window icon (see links below) or the system default icon will be used.

This method makes an internal copy of the icon pixel buffer, so once set, the Fl_RGB_Image instance can be freed by the caller.

**Parameters**

| in | icon | icon for this window, NULL to reset window icon. |

See also

- `Fl_Window::default_icon(const Fl_RGB_Image *)`
- `Fl_Window::default_icons(const Fl_RGB_Image *, int)`
- `Fl_Window::icons(const Fl_RGB_Image *, int)`

**Note**

See Window icons for the Wayland platform.

### 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)`.
### 33.160.3.36 iconize()

```c
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.

### 33.160.3.37 icons()[1/2]

```c
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><code>icons</code></th>
<th>icons for this window</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>count</code></td>
<td>number of images in <code>icons</code>. Set to 0 to remove the current 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.

### 33.160.3.38 icons()[2/2]

```c
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><code>big_icon</code></th>
<th>large window icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>small_icon</code></td>
<td>small window icon</td>
</tr>
</tbody>
</table>

### 33.160.3.39 is_resizable()

```c
int Fl_Window::is_resizable () [protected]
```

Generated by Doxygen
Protected method to determine whether a window is resizable. If \texttt{size_range()} has not yet been called this method calculates the default size range values by calling \texttt{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

\begin{verbatim}
non-zero if the window is resizable
\end{verbatim}

Return values

\begin{table}[h]
\begin{tabular}{|c|c|}
\hline
0 & the window is not resizable \tabularnewline
1 & the window is resizable in horizontal direction (w) \tabularnewline
2 & the window is resizable in vertical direction (h) \tabularnewline
3 & the window is resizable in both directions (w and h) \tabularnewline
\hline
\end{tabular}
\end{table}

See also

\texttt{default_size_range()}

### 33.160.3.40 make_current()

\begin{verbatim}
void Fl_Window::make_current ()
\end{verbatim}

Sets things up so that the drawing functions in \texttt{<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. \textbf{Danger: incremental update is very hard to debug and maintain!}

This method only works for the \texttt{Fl_Window} and \texttt{Fl_Gl_Window} derived classes.

### 33.160.3.41 modal()

\begin{verbatim}
unsigned int Fl_Window::modal () const [inline]
\end{verbatim}

Returns true if this window is modal.

### 33.160.3.42 os_id()

\begin{verbatim}
fl_uintptr_t Fl_Window::os_id ()
\end{verbatim}

Returns a platform-specific identification of a shown window, or 0 if not shown.

\textbf{Note}

This identification may differ from the platform-specific reference of an \texttt{Fl_Window} object used by functions \texttt{fl_x11_xid()}, \texttt{fl_mac_xid()}, \texttt{fl_x11_find()}, and \texttt{fl_mac_find()}.

- X11 platform: the window's XID.
- macOS platform: The window number of the window's window device.
- other platforms: 0.

### 33.160.3.43 resize()

\begin{verbatim}
void Fl_Window::resize (int X, int Y,
\end{verbatim}
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`.

### 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.

### set_menu_window()

```cpp
void Fl_Window::set_menu_window ( )
```

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.

### set_modal()

```cpp
void Fl_Window::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).

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()`.

### set_non_modal()

```cpp
void Fl_Window::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.

There are three states for a window: modal, non-modal, and normal.

### set_tooltip_window()

```cpp
void Fl_Window::set_tooltip_window ( )
```

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.

33.160.3.49 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)

33.160.3.50 shape() [2/2]

void Fl_Window::shape (const Fl_Image * img)

Assigns a non-rectangular shape to the window.

This function gives an arbitrary shape (not just a rectangular region) to an Fl_Window. An Fl_Image of any dimension can be used as mask; it is rescaled to the window’s dimension as needed.

The layout and widgets inside are unaware of the mask shape, and most will act as though the window’s rectangular bounding box is available to them. It is up to you to make sure they adhere to the bounds of their masking shape.

The img argument can be an Fl_Bitmap, Fl_Pixmap, Fl_RGB_Image or Fl_Shared_Image:

- With Fl_Bitmap or Fl_Pixmap, the shaped window covers the image part where bitmap bits equal one, or where the pixmap is not fully transparent.
- With an Fl_RGB_Image with an alpha channel (depths 2 or 4), the shaped window covers the image part that is not fully transparent.
- With an Fl_RGB_Image of depth 1 (gray-scale) or 3 (RGB), the shaped window covers the non-black image part.
- With an Fl_Shared_Image, the shape is determined by rules above applied to the underlying image. The shared image should not have been scaled through Fl_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
33.160.3.51  show() [1/2]

void Fl_Window::show ( ) [virtual]
Puts the window on the screen.
This has the side effect of opening the display, if not done before.
If the window is already shown then it is restored and raised to the top. This is really convenient because your
program can call show() at any time, even if the window is already up. It also means that show() serves the purpose
of raise() in other toolkits.
Fl_Window::show(int argc, char **argv) is used for top-level windows and allows standard arguments to be parsed
from the command-line.

Note
For some obscure reasons Fl_Window::show() resets the current group by calling Fl_Group::current(0). The
comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have
unwanted side effects if you show() an unrelated window (maybe for an error message or warning) while
building a window or any other group widget.

Todo
Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break
"already broken" programs though if they rely on this "feature".

See also
Fl_Window::show(int argc, char **argv)
Reimplemented from Fl_Widget.

33.160.3.52  show() [2/2]

void Fl_Window::show (  
   int argc,  
   char ** argv )
Puts the window on the screen with show() and parses command-line arguments.
This call should be used for top-level windows, at least for the first (main) window. It allows standard arguments to
be parsed, as done by Fl::args(int, char **), from the command-line. You can use argv and argc from main(int
argc, char **argv) for this call.
This call also sets up some system-specific internal variables, that is, it sets FL_SELECTION_COLOR and
calls Fl::background(), Fl::background2(), Fl::foreground() with default or X resources-given values, and calls
Fl::scheme(const char *) for the current scheme. On X11, it also calls Fl::dnd_text_ops(int), Fl_Tooltip::enable(int),
Fl::visible_focus(int) with X resources-given values.

Parameters

<table>
<thead>
<tr>
<th>argc</th>
<th>command-line argument count, usually from main()</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>command-line argument vector, usually from main()</td>
</tr>
</tbody>
</table>

See also
virtual void Fl_Window::show()
    Fl::args(int, char **)

33.160.3.53  shown()

int Fl_Window::shown ( ) [inline]
Returns non-zero if show() has been called (but not hide() ).
You can tell if a window is iconified with (w->shown() && lw->visible()).
33.160.3.54 size_range()

```cpp
default 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.

We recommend to call `size_range()` if you have a `resizable()` widget in a main window, and to call it after setting the `resizable()` and before `show()`ing the window for best cross platform compatibility.

If this function is not called, FLTK tries to figure out the range when the window is shown. Please see the protected method `default_size_range()` for details.

It is undefined what happens if the current window size does not fit in the constraints passed to `size_range()`.

**Note**

This only works for top-level windows and the exact behavior can be platform specific. To work correctly across all platforms `size_range()` must be called after setting the `resizable()` widget of the window and before the window is `show()`.

Calling `size_range()` after the window has been shown may work on some but not all platforms. If you need to change the `size_range()` after the window has been shown, then you should consider to `hide()` and `show()` the window again, i.e. call `hide(), size_range(), and show()` in this order.

Typical usage: call `size_range(minWidth, minHeight);`

after setting the resizable widget and before calling `show()`. This ensures that the window cannot be resized smaller than the given values by user interaction.

`maxWidth` and `maxHeight` might be useful in some special cases but less frequently used.

The other optional parameters `deltaX, deltaY, and aspectRatio` are not recommended because they may not work on all platforms and may even under X11 not be supported by all Window Managers.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>minWidth, minHeight</code></th>
<th>The smallest the window can be. Either value must be greater than 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>maxWidth, maxHeight</code></td>
<td>The largest the window can be. If either is equal to the minimum then you cannot resize in that direction. If either is zero then FLTK picks a maximum size in that direction such that the window will fill the screen.</td>
</tr>
<tr>
<td>in</td>
<td><code>deltaX, deltaY</code></td>
<td>These are size increments. The window will be constrained to widths of <code>minWidth + N * deltaX</code>, where N is any non-negative integer. If these are less or equal to 1 they are ignored (this is always ignored on Windows).</td>
</tr>
<tr>
<td>in</td>
<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>

33.160.3.55 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_for_expose();  // wait, until displayed
Fl::flush();  // make sure everything gets drawn
// do more initialization work that needs some time here ...
Fl::run();  // start FLTK event loop
```

Note that the window will not be responsive until the event loop is started with `Fl::run()`.

### 33.160.3.56 `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 *)
```

### 33.160.3.57 `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 *)
```
33.160.4  Member Data Documentation

33.160.4.1  current_

Fl_Window * Fl_Window::current_ [static], [protected]
Stores the last window that was made current.
See current() const
The documentation for this class was generated from the following files:

- Fl_Window.H
- Fl_arg.cxx
- fl_cursor.cxx
- Fl_Window.cxx
- Fl_Window_fullscreen.cxx
- Fl_Window_hotspot.cxx
- Fl_Window_iconize.cxx

33.161  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

- **void draw ()** FL_OVERRIDE
  Draws the wizard border and visible child.
Additional Inherited Members

33.161.1 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.

33.161.2 Constructor & Destructor Documentation

33.161.2.1 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.

33.161.3 Member Function Documentation

33.161.3.1 draw()

void Fl_Wizard::draw (  
    void ) [protected], [virtual]

Draws the wizard border and visible child.
Reimplemented from Fl_Group.

33.161.3.2 next()

void Fl_Wizard::next ( )

This method shows the next child of the wizard.
If the last child is already visible, this function does nothing.
The documentation for this class was generated from the following files:

- Fl_Wizard.H  
- Fl_Wizard.cxx

33.162 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:
Public Member Functions

- **Fl_XBM_Image** (const char *filename)
  
  The constructor loads the named XBM file from the given name filename.

Additional Inherited Members

33.162.1 Detailed Description

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

33.162.2 Constructor & Destructor Documentation

33.162.2.1 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

33.163 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

33.164 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:
Public Member Functions

- **Fl_XPM_Image** (const char ∗filename)
  
  *The constructor loads the XPM image from the name filename.*

Additional Inherited Members

33.164.1 Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including trans-
parency.

33.164.2 Constructor & Destructor Documentation

33.164.2.1 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

33.165 Fl_Zenity_Native_File_Chooser_Driver Class Reference

Inheritance diagram for Fl_Zenity_Native_File_Chooser_Driver:

Friends

- class Fl_Native_File_Chooser

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser_Zenity.H
- Fl_Native_File_Chooser_Zenity.cxx
33.166   Fl_GIF_Image::GIF_FRAME Struct Reference

Classes

• struct CPAL

Public Member Functions

• void colors (int nclrs, int bg, int tp)
• void disposal (int mode, int delay)
• GIF_FRAME (int frame, int W, int H, int fx, int fy, int fw, int fh, uchar ∗data)
• GIF_FRAME (int frame, uchar ∗data)

Public Attributes

• int bkgd
• const uchar ∗ bptr
• int clrs
• const struct Fl_GIF_Image::GIF_FRAME::CPAL ∗ cpal
• int delay
• int dispose
• int h
• int height
• int ifrm
• int trans
• int w
• int width
• int x
• int y

The documentation for this struct was generated from the following file:

• Fl_GIF_Image.H

33.167   Fl_ICO_Image::IconDirEntry Struct Reference

Windows ICONDIRENTRY structure

#include <Fl_ICO_Image.H>

Public Attributes

• int bColorCount
  Number of colors (0 if 8bpp)
• int bHeight
  Image height.
• int bReserved
  Reserved.
• int bWidth
  Image width.
• int dwBytesInRes
  Resource size in bytes.
• int dwImageOffset
  Offset to the image.
• int wBitCount
  Bits per pixel.
• int wPlanes
  Color Planes.
33.167.1 Detailed Description

Windows ICONDIRENTRY structure

The documentation for this struct was generated from the following file:

- FL_ICO_Image.H

33.168 Fl_Text_Editor::Key_Binding Struct Reference

Simple linked list item associating a key/state to a function.

```cpp
#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

33.168.1 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

33.169 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.

```cpp
#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.

33.169.1 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.
33.169.2 Constructor & Destructor Documentation

33.169.2.1 Name() [1/2]

```cpp
Fl_Preferences::Name::Name (  
    unsigned int n)
```

Creates a group name or entry name on the fly.  
This version creates a simple unsigned integer as an entry name.
```cpp
int n, i;
FL_Preferences prev( appPrefs, "PreviousFiles" );
prev.get( "n", 0 );
for ( i=0; i<n; i++ )
    prev.get( FL_Preferences::Name(i), prevFile[i], "" );
```

33.169.2.2 Name() [2/2]

```cpp
Fl_Preferences::Name::Name (  
    const char * format,
    ... )
```

Creates a group name or entry name on the fly.  
This version creates entry names as in 'printf'.
```cpp
int n, i;
FL_Preferences prefs( USER, "matthiasm.com", "test" );
prefs.get( "nFiles", 0 );
for ( i=0; i<n; i++ )
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

33.170 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

33.171 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

33.171.1 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

33.172 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

33.173 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

### 33.173.1 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

### 33.174 Fl_Text_Display::Style_Table_Entry Struct Reference

This structure associates the color, font, and font size of a string to draw with an attribute mask matching `attr`. #include `<Fl_Text_Display.H>`

Public Attributes

- **unsigned attr**
  further attributes for the text style (see `ATTR_BGCOLOR`, etc.)

- **Fl_Color bgcolor**
  text background color if `ATTR_BGCOLOR` or `ATTR_BGCOLOR_EXT` is set

- **Fl_Color color**
  text color

- **Fl_Font font**
  text font

- **Fl_Fontsize size**
  text font size
33.174.1 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 34

File Documentation

34.1 Enumerations.H File Reference

This file contains type definitions and general enumerations.
#include <FL/fl_config.h>
#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h>

Macros

- \#define FL_IMAGE_WITH_ALPHA 0x40000000

Version Numbers

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.
The following constants are defined:

- \#define FL_ABI_VERSION FL_ABI_VERSION
  The FLTK ABI (Application Binary Interface) version number as an int.
- \#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)
  The FLTK API version number as an int.
- \#define FL_MAJOR_VERSION 1
  The major release version of this FLTK library.
- \#define FL_MINOR_VERSION 4
  The minor release version for this library.
- \#define FL_PATCH_VERSION 0
  The patch version for this library.
- \#define FL_VERSION
  The FLTK version number as a double.

Names of Non-ASCII keys and mouse buttons

The following constants define the names of non-ASCII keys on the keyboard and of mouse buttons for FL_TERMINAL
KEYBOARD and FL_SHORTCUT events.

See also

- Fl::event_key() and Fl::get_key(int) (use ASCII letters for all other keys):

  - \#define FL_Alt_L 0xffe9
    The left alt key.
  - \#define FL_Alt_R 0xffea
    The right alt key.
  - \#define FL_Back 0xEF26
    Like back on a browser.
• #define FL_BackSpace 0xff08
  The backspace key.
• #define FL_Button 0xfee8
  A mouse button; use FL_Button + n for mouse button n.
• #define FL_Caps_Lock 0xffe5
  The caps lock key.
• #define FL_Control_L 0xffe3
  The lefthand control key.
• #define FL_Control_R 0xffe4
  The righthand control key.
• #define FL_Delete 0xffff
  The delete key.
• #define FL_Down 0xff54
  The down arrow key.
• #define FL_Eisu 0xff2f
  The Eisu key of JIS keyboards.
• #define FL_End 0xff57
  The end key.
• #define FL_Escape 0xff1b
  The escape key.
• #define FL_F 0xffbd
  One of the function keys; use FL_F + n for function key n.
• #define FL_F_Last 0xffe0
  The last function key; use to range-check function keys.
• #define FL_Favorites 0xEF30
  Show favorite locations.
• #define FL_Forward 0xEF27
  Like forward on a browser.
• #define FL_Help 0xff68
  The 'help' key on Mac keyboards.
• #define FL_Home 0xff50
  The home key.
• #define FL_Home_Page 0xEF18
  Display user's home page.
• #define FL_Insert 0xff63
  The insert key.
• #define FL_Iso_Key 0xff0c
  The additional key of ISO keyboards.
• #define FL_JIS_Underscore 0xff31
  The underscore key of JIS keyboards.
• #define FL_Kana 0xff2e
  The Kana key of JIS keyboards.
• #define FL_KP 0xff80
  One of the keypad numbers; use FL_KP + n' for digit n.
• #define FL_KP_Enter 0xff8d
  The enter key on the keypad, same as FL_KP+'r'.
• #define FL_KP_Last 0xffbd
  The last keypad key; use to range-check keypad.
• #define FL_Left 0xff51
  The left arrow key.
• #define FL_Mail 0xEF19
  Invoke user's mail program.
• #define FL_Media_Next 0xEF17
  Next track.
• #define FL_Media_Play 0xEF14
  Start playing of audio.
• #define FL_Media_Prev 0xEF16
• #define FL_Media_Stop 0xEF15
  Stop playing audio.
• #define FL_Menu 0xff67
  The menu key.
• #define FL_Meta_L 0xffe7
  The left meta/Windows key.
• #define FL_Meta_R 0xffe8
  The right meta/Windows key.
• #define FL_Num_Lock 0xff7f
  The num lock key.
• #define FL_Page_Down 0xff56
  The page-down key.
• #define FL_Page_Up 0xff55
  The page-up key.
• #define FL_Pause 0xff13
  The pause key.
• #define FL_Print 0xff61
  The print (or print-screen) key.
• #define FL_Refresh 0xEF29
  Refresh the page.
• #define FL_Right 0xff53
  The right arrow key.
• #define FL_Scroll_Lock 0xff14
  The scroll lock key.
• #define FL_Search 0xEF1B
  Search.
• #define FL_Shift_L 0xffe1
  The lefthand shift key.
• #define FL_Shift_R 0xffe2
  The righthand shift key.
• #define FL_Sleep 0xEF2F
  Put system to sleep.
• #define FL_Stop 0xEF28
  Stop current operation.
• #define FL_Tab 0xff09
  The tab key.
• #define FL_Up 0xff52
  The up arrow key.
• #define FL_Volume_Down 0xEF11
  Volume control down.
• #define FL_Volume_Mute 0xEF12
  Mute sound from the system.
• #define FL_Volume_Up 0xEF13
  Volume control up.
• #define FL_Yen 0xff30
  The Yen key of JIS keyboards.

Mouse Buttons
These constants define the button numbers for FL_PUSH and FL_RELEASE events.

See also

  Fl::event_button()

• #define FL_LEFT_MOUSE 1
  The left mouse button.
• #define FL_MIDDLE_MOUSE 2
  The middle mouse button.
• #define FL_RIGHT_MOUSE 3
The right mouse button.

**Event States**

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

- `#define FL_ALT 0x00080000` One of the alt keys is down.
- `#define FL_BUTTON(n) (0x00800000<<(n))` Mouse button n (n > 0) is pushed.
- `#define FL_BUTTON1 0x01000000` Mouse button 1 is pushed.
- `#define FL_BUTTON2 0x02000000` Mouse button 2 is pushed.
- `#define FL_BUTTON3 0x04000000` Mouse button 3 is pushed.
- `#define FL_BUTTONS 0x7f000000` Any mouse button is pushed.
- `#define FL_CAPS_LOCK 0x00020000` The caps lock is on.
- `#define FL_CTRL 0x00040000` One of the ctrl keys is down.
- `#define FL_KEY_MASK 0x0000ffff` All keys are 16 bit for now.
- `#define FL_META 0x00400000` One of the meta/Windows keys is down.
- `#define FL_NUM_LOCK 0x00100000` The num lock is on.
- `#define FL_SCROLL_LOCK 0x00800000` The scroll lock is on.
- `#define FL_SHIFT 0x00010000` One of the shift keys is down.

**Typedefs**

- typedef int Fl_Fontsize
  
  Size of a font in pixels.

**Enumerations**

- enum { FL_READ = 1 , FL_WRITE = 4 , FL_EXCEPT = 8 }
  
  FD "when" conditions.
- enum Fl_Arrow_Type { FL_ARROW_SINGLE = 0x01 , FL_ARROW_DOUBLE = 0x02 , FL_ARROW_CHOICE = 0x03 , FL_ARROW_RETURN = 0x04 }
  
  Arrow types define the type of arrow drawing function.
- enum Fl_Damage {
  FL_DAMAGE_CHILD = 0x01 , FL_DAMAGEExpose = 0x02 , FL_DAMAGE_SCROLL = 0x04 ,
  FL_DAMAGE_OVERLAY = 0x08 ,
  FL_DAMAGE_USER1 = 0x10 , FL_DAMAGE_USER2 = 0x20 , FL_DAMAGE_ALL = 0x80 }
  
  Damage masks.
- enum Fl_Event {
  FL_NO_EVENT = 0 , FL_PUSH = 1 , FL_RELEASE = 2 , FL_ENTER = 3 ,
  FL_LEAVE = 4 , FL_DRAG = 5 , FL_FOCUS = 6 , FL_UNFOCUS = 7 ,
  FL_KEYDOWN = 8 , FL_KEYBOARD = 8 , FL_KEYUP = 9 , FL_CLOSE = 10 ,
  FL_MOVE = 11 , FL_SHORTCUT = 12 , FL_DEACTIVATE = 13 , FL_ACTIVATE = 14 ,
  FL_HIDE = 15 , FL_SHOW = 16 , FL_PASTE = 17 , FL_SELECTIONCLEAR = 18 ,
  FL_MOUSEWHEEL = 19 , FL_DND_ENTER = 20 , FL_DND_DRAG = 21 , FL_DND_LEAVE = 22 ,
  FL_DND_RELEASE = 23 , FL_SCREEN_CONFIGURATION_CHANGED = 24 , FL_FULLSCREEN = 25 ,
  FL_ZOOM_GESTURE = 26 ,
  FL_ZOOM_EVENT = 27 }
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

- **enum Fl_Labeltype**

  ```c
  FL_NORMAL_LABEL = 0 , FL_NO_LABEL , FL_SHADOW_LABEL , FL_ENGRAVED_LABEL ,
  FL_EMBOSSS_LABEL , FL_MULTI_LABEL , FL_ICON_LABEL , FL_IMAGE_LABEL ,
  FL_FREE_LABELTYPE
  ```

  The labeltype() method sets the type of the label.

- **enum Fl_Mode**

  ```c
  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)

- **enum Fl_Orientation**

  ```c
  FL_ORIENT_NONE = 0x00 , FL_ORIENT_RIGHT = 0x00 , FL_ORIENT_NE = 0x01 , FL_ORIENT_UP =
  0x02 ,
  FL_ORIENT_NW = 0x03 , FL_ORIENT_LEFT = 0x04 , FL_ORIENT_SW = 0x05 , FL_ORIENT_DOWN =
  0x06 ,
  FL_ORIENT_SE = 0x07
  ```

  Fl_Orientation describes the orientation of a GUI element.

When Conditions

- **enum Fl_When**

  ```c
  FL_WHEN_NEVER = 0 , FL_WHEN_CHANGED = 1 , FL_WHEN_NOT_CHANGED = 2 , FL_WHEN_RELEASE =
  4 ,
  FL_WHEN_RELEASE_ALWAYS = 6 , FL_WHEN_ENTER_KEY = 8 , FL_WHEN_ENTER_KEY_ALWAYS =
  10 , FL_WHEN_ENTER_KEY_CHANGED = 11 ,
  FL_WHEN_CLOSED = 16
  ```

  These constants determine when a callback is performed.

Callback Reasons

- **enum Fl_Callback_Reason**

  ```c
  FL_REASON_UNKNOWN =0 , FL_REASON_SELECTED , FL_REASON_DESELECTED , FL_REASON_RESELECTED ,
  FL_REASON_OPENED , FL_REASON_CLOSED , FL_REASON_DRAGGED , FL_REASON_CANCELLED ,
  FL_REASON_CHANGED , FL_REASON_GOT_FOCUS , FL_REASON_LOST_FOCUS , FL_REASON_RELEASED
  ```

  These constants describe why a callback is performed.

Cursors

- **enum Fl_Cursor**

  ```c
  FL_CURSOR_DEFAULT = 0 , FL_CURSOR_ARROW = 35 , FL_CURSOR_CROSS = 66 ,
  FL_CURSOR_WAIT = 76 ,
  FL_CURSOR_INSERT = 77 , FL_CURSOR_HAND = 31 , FL_CURSOR_HELP = 47 , FL_CURSOR_MOVE =
  27 ,
  FL_CURSOR_NS = 78 , FL_CURSOR_WE = 79 , FL_CURSOR_NWSE = 80 , FL_CURSOR_NESW = 81 ,
  FL_CURSOR_N = 70 , FL_CURSOR_NE = 69 , FL_CURSOR_E = 49 , FL_CURSOR_SE = 8 ,
  FL_CURSOR_S = 9 , FL_CURSOR_SW = 7 , FL_CURSOR_W = 36 , FL_CURSOR_NW = 68 ,
  FL_CURSOR_NONE =255
  ```

  The following constants define the mouse cursors that are available in FLTK.

Variables

- **Fl_Fontsize FL_NORMAL_SIZE**

  normal font size
Box Types

FLTK standard box types
This enum defines the standard box types included with FLTK.

Note

The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. _FL_SHADOW_BOX. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

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

Figure 34.1 FLTK Standard Box Types

Note

Not all box types are depicted in the figure above. See enum Fl_Boxtype below for the complete list of box types.

See also

Fl::get_system_colors()

• Fl_Boxtype fl_box (Fl_Boxtype b)
  Get the filled version of a frame.

• enum Fl_Boxtype {
  FL_NO_BOX = 0 , FL_FLAT_BOX , FL_UP_BOX , FL_DOWN_BOX ,
  FL_UP_FRAME , FL_DOWN_FRAME , FL_THIN_UP_BOX , FL_THIN_DOWN_BOX ,
  FL_THIN_UP_FRAME , FL_THIN_DOWN_FRAME , FL_ENGRAVED_BOX , FL_EMBOSSED_BOX ,
  FL_ENGRAVED_FRAME , FL_EMBOSSED_FRAME , FL_BORDER_BOX , _FL_SHADOW_BOX ,
  FL_PLASTIC_UP_BOX , FL_PLASTIC_DOWN_BOX , FL_PLASTIC_THIN_UP_BOX , FL_PLASTIC_THIN_DOWN_BOX ,
  FL_PLASTIC_ROUND_UP_BOX , FL_PLASTIC_ROUND_DOWN_BOX , FL_PLASTIC_THIN_ROUND_UP_BOX , FL_PLASTIC_THIN_ROUND_DOWN_BOX ,
  FL_GTK_UP_BOX , FL_GTK_DOWN_BOX , FL_GTK_THIN_UP_BOX , FL_GTK_THIN_DOWN_BOX ,
  FL_GTK_THIN_UP_FRAME , FL_GTK_THIN_DOWN_FRAME , FL_GTK_UP_FRAME , FL_GTK_DOWN_FRAME ,
  FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GTK_ROUND_UP_FRAME , FL_GTK_ROUND_DOWN_FRAME ,
  FL_SHADOW_BOX , FL_SHADOW_FRAME , FL_BORDER_BOX , FL_BORDER_FRAME ,
  _FL_SHADOW_BOX , _FL_SHADOW_FRAME }
FLTK standard box types.

- #define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
- #define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX
- Fl_Boxtype fl_down (Fl_Boxtype b)
  Get the "pressed" or "down" version of a box.
- Fl_Boxtype fl_frame (Fl_Boxtype b)
  Get the unfilled, frame only version of a box.
- #define FL_FRAME FL_ENGRAVED_FRAME
- #define FL_FRAME_BOX FL_ENGRAVED_BOX
- Fl_Labeltype fl_define_FL_EMBOSSED_LABEL ()
  Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_ENGRAVED_LABEL ()
  Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_ICON_LABEL ()
  Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_IMAGE_LABEL ()
  Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_MULTI_LABEL ()
  Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_SHADOW_LABEL ()
  Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
- #define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL() 
  Draws a label with embossed text.
- #define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL() 
  Draws a label with engraved text.
- #define FL_ICON_LABEL fl_define_FL_ICON_LABEL() 
  Draws an icon as the label.
- #define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL() 
  Draws an image (Fl_Image) as the label.
- #define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL() 
  Draws a label that can comprise several parts like text and images.
• #define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()
  Draws a label with shadows behind the text.
• #define FL_SYMBOL_LABEL FL_NORMAL_LABEL
  Sets the current label type and returns its corresponding Fl_Labeltype value.

Colors

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

```
Fl_Color => 0xrrrggbbi
  | | |     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
  Default background color.
• const Fl_Color FL_BLACK = 56
• const Fl_Color FL_BLUE = 216
• typedef unsigned int Fl_Color
  An FLTK color value; see also Colors

• Fl_Color fl_color_average (Fl_Color c1, Fl_Color c2, float weight)
  Returns the weighted average color between the two given colors.
• #define FL_COLOR_CUBE (Fl_Color)56
• Fl_Color fl_color_cube (int r, int g, int b)
  Returns a color out of the color cube.
• Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs=0, int context=0)
  Returns a color that contrasts with the background color.
• typedef Fl_Color() Fl_Contrast_Function(Fl_Color, Fl_Color, Fl_Fontsize, int)
  Type of a custom fl_contrast() function.
• void fl_contrast_function (Fl_Contrast_Function *f)
  Register a custom contrast function.
• int fl_contrast_level ()
  Get the contrast level (sensitivity) of the fl_contrast() method.
• void fl_contrast_level (int level)
  Set the contrast level (sensitivity) of the fl_contrast() method.
• enum Fl_Contrast_Mode {
  FL_CONTRAST_NONE = 0 , FL_CONTRAST_LEGACY , FL_CONTRAST_CIELAB , FL_CONTRAST_CUSTOM
  ,
  FL_CONTRAST_LAST }
  Define the possible modes to calculate fl_contrast().
• int fl_contrast_mode ()
  Return the current contrast algorithm (mode).
• void fl_contrast_mode (int mode)
Set the contrast algorithm (mode).

- const Fl_Color FL_CYAN = 223
- const Fl_Color FL_DARK1 = 47
- const Fl_Color FL_DARK2 = 45
- const Fl_Color FL_DARK3 = 39
- const Fl_Color FL_DARK_BLUE = 136
- const Fl_Color FL_DARK_CYAN = 140
- const Fl_Color FL_DARK_GREEN = 60
- const Fl_Color FL_DARK_MAGENTA = 152
- const Fl_Color FL_DARK_RED = 72
- const Fl_Color FL_DARK_YELLOW = 76

\textbf{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

\textbf{#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().

\textbf{#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\_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.

\textbf{double fl\_lightness (Fl\_Color color)}

Return the perceived lightness of a color.

\textbf{double fl\_luminance (Fl\_Color color)}

Return the raw / physical luminance of a color.

- const Fl_Color FL\_MAGENTA = 248
- #define FL\_NUM\_BLUE 5
- #define FL\_NUM\_FREE\_COLOR 16
- #define FL\_NUM\_GRAY 24
- #define FL\_NUM\_GREEN 8
- #define FL\_NUM\_RED 5
- const Fl_Color FL\_RED = 88
- Fl\_Color fl\_rgb\_color (uchar g)
  Returns the 24-bit color value closest to $g$ (grayscale).

- Fl\_Color fl\_rgb\_color (uchar r, uchar g, uchar b)
  Returns the 24-bit color value closest to $r$, $g$, $b$.

- const Fl_Color FL\_SELECTION\_COLOR = 15
  the default selection/highlight color

- const Fl_Color FL\_WHITE = 255
- const Fl\_Color FL\_YELLOW = 95
Alignment Flags

Flags to control the label alignment.
This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER (0)
for most widgets, which centers the label inside the widget.
All alignment flags use the common prefix “FL_ALIGN_”. In the following descriptions this prefix is sometimes
omitted for brevity.
Flags can be or’d to achieve a combination of alignments, but there are some “magic values” (e.g. combinations of
TOP and BOTTOM and of LEFT and RIGHT) that have special meanings (see below). For instance:

FL_ALIGN_TOP_LEFT == (FL_ALIGN_TOP | FL_ALIGN_LEFT) != FL_ALIGN_LEFT_TOP.

Outside alignments (FL_ALIGN_INSIDE is not set):

| LEFT_TOP | TOP_LEFT | TOP_TOP | TOP_RIGHT |
| LEFT | CENTER | RIGHT |
| LEFT_BOTTOM | TOP_BOTTOM | TOP_RIGHT |

Inside alignments (FL_ALIGN_INSIDE is set):

| TOP_LEFT | TOP | TOP_RIGHT |
| LEFT | CENTER | RIGHT |
| BOTTOM_LEFT | BOTTOM | BOTTOM_RIGHT |

See also

Fl_Align, FL_ALIGN_CENTER, etc.

Note

1. Bit positions not defined in the following constants of type Fl_Align are reserved for future extensions. Do not use.
2. The “magic values” (FL_ALIGN_)LEFT_TOP, RIGHT_TOP, LEFT_BOTTOM, and RIGHT_BOTTOM must not be used together with FL_ALIGN_INSIDE. Use TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, or BOTTOM_RIGHT instead.
3. Although bits can be or’d together there are some unused/illegal combinations, for instance:
   • setting both FL_ALIGN_TOP and FL_ALIGN_BOTTOM in combinations other than those given in the Fl_Align constants below (magic values)
   • setting both FL_ALIGN_LEFT and FL_ALIGN_RIGHT in combinations other than those given in the Fl_Align constants below (magic values)
   • using one of the “magic values” (2) together with FL_ALIGN_INSIDE

Using illegal bit combinations or undefined bits may yield unexpected behavior, and this behavior may be changed without notice in future FLTK versions.

• typedef unsigned Fl_Align
  
  FLTK type for alignment control.
• const Fl_Align FL_ALIGN_BOTTOM = 0x0002
  Align the label at the bottom of the widget.
• const Fl_Align FL_ALIGN_BOTTOM_LEFT = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT
• const Fl_Align FL_ALIGN_BOTTOM_RIGHT = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_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_ALIGN_POSITION_MASK = 0x000f
  Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.
• const Fl_Align FL_ALIGN_RIGHT = 0x0008
  Align the label to the right of the widget.
• const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e
  Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.
• const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b
  Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.
• const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120
  If the label contains an image, draw the text to the left of the image.
• const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
  If the label contains an image, draw the text on top of the image.
• const Fl_Align FL_ALIGN_TOP = 0x0001
  Align the label at the top of the widget.
• const Fl_Align FL_ALIGN_TOP_LEFT = FL_ALIGN_TOP | FL_ALIGN_LEFT
• const Fl_Align FL_ALIGN_TOP_RIGHT = FL_ALIGN_TOP | FL_ALIGN_RIGHT
• const Fl_Align FL_ALIGN_WRAP = 0x0080
  Wrap text that does not fit the width of the widget.

Font Numbers

The following constants define the standard FLTK fonts:

• const Fl_Font FL_BOLD = 1
  add this to helvetica, courier, or times
• const Fl_Font FL_BOLD_ITALIC = 3
  add this to helvetica, courier, or times
• const Fl_Font FL_COURIER = 4
  Courier normal.
• const Fl_Font FL_COURIER_BOLD = 5
  Courier bold.
• const Fl_Font FL_COURIER_BOLD_ITALIC = 7
  Courier bold-italic.
• const Fl_Font FL_COURIER_ITALIC = 6
  Courier italic.
• typedef int Fl_Font
  A font number is an index into the internal font table.
• const Fl_Font FL_FREE_FONT = 16

Generated by Doxygen
first one to allocate

- const Fl_Font FL_HELVETICA = 0
  Helvetica (or Arial) normal (0)
- const Fl_Font FL_HELVETICA_BOLD = 1
  Helvetica (or Arial) bold.
- const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3
  Helvetica (or Arial) bold-oblique.
- const Fl_Font FL_HELVETICA_ITALIC = 2
  Helvetica (or Arial) oblique.
- const Fl_Font FL_ITALIC = 2
  add this to helvetica, courier, or times
- const Fl_Font FL_SCREEN = 13
  Default monospaced screen font.
- const Fl_Font FL_SCREEN_BOLD = 14
  Default monospaced bold screen font.
- const Fl_Font FL_SYMBOL = 12
  Standard symbol font.
- const Fl_Font FL_TIMES = 8
  Times roman.
- const Fl_Font FL_TIMES_BOLD = 9
  Times roman bold.
- const Fl_Font FL_TIMES_BOLD_ITALIC = 11
  Times roman bold-italic.
- const Fl_Font FL_TIMES_ITALIC = 10
  Times roman italic.
- const Fl_Font FL_ZAPF_DINGBATS = 15
  Zapf-dingbats font.

34.1.1 Detailed Description

This file contains type definitions and general enumerations.

34.1.2 Macro Definition Documentation

34.1.2.1 FL_ABI_VERSION

#define FL_ABI_VERSION FL_API_VERSION
The FLTK ABI (Application Binary Interface) version number as an int.
FL_ABI_VERSION is an int that describes the major, minor, and patch ABI version numbers in the same format as
FL_API_VERSION.
The ABI version number FL_ABI_VERSION is usually the same as the API version FL_API_VERSION with the
last two digits set to '00'. FLTK retains the ABI (Application Binary Interface) during patch releases of the same major and minor versions.
Examples:

<table>
<thead>
<tr>
<th>FLTK Version</th>
<th>FL_API_VERSION</th>
<th>FL_ABI_VERSION</th>
<th>FL_VERSION (deprecated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.0</td>
<td>10300</td>
<td>10300</td>
<td>1.0300</td>
</tr>
<tr>
<td>1.3.4</td>
<td>10304</td>
<td>10300</td>
<td>1.0304</td>
</tr>
</tbody>
</table>

Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.
The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.
To enable new ABI-breaking features in patch releases you can configure FLTK to use a higher FL_ABI_VERSION.
34.1 Enumerations.H File Reference

See also

README.abi-version.txt

34.1.2.2 FL_API_VERSION

#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)

The FLTK API version number as an int.
FL_API_VERSION is an int that describes the major, minor, and patch version numbers.
Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.
The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values
for the major, minor, and patch releases respectively.

Note

FL_API_VERSION is intended to replace the deprecated double FL_VERSION.

See also

Fl::api_version()

34.1.2.3 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.

34.1.2.4 FL_MAJOR_VERSION

#define FL_MAJOR_VERSION 1

The major release version of this FLTK library.

See also

FL_VERSION

34.1.2.5 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.

34.1.2.6 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

34.1.2.7 FL_PATCH_VERSION

#define FL_PATCH_VERSION 0

The patch version for this library.
FLTK remains binary compatible between patches.
34.1.2.8  **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.

34.1.2.9  **FL_VERSION**

#define FL_VERSION

Value:

\[
\begin{align*}
\text{(double)} \text{FL\_MAJOR\_VERSION} + & \\
\text{(double)} \text{FL\_MINOR\_VERSION} \times 0.01 + & \\
\text{(double)} \text{FL\_PATCH\_VERSION} \times 0.0001 &
\end{align*}
\]

The FLTK version number as a double.

FL_VERSION is a double that describes the major, minor, and patch version numbers. Version 1.2.3 is actually stored as 1.0203 to allow for more than 9 minor and patch releases.

**Deprecated** This double version number is retained for compatibility with existing program code. New code should use int FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

FL_VERSION is equivalent to \((double)\text{FL\_API\_VERSION} / 10000\).

See also

- Fl::version() (deprecated as well)
- FL_API_VERSION
- Fl::api_version()

34.1.3  **Typedef Documentation**

34.1.3.1  **Fl\_Contrast\_Function**

typedef Fl\_Color() Fl\_Contrast\_Function(Fl\_Color, Fl\_Color, Fl\_Fontsize, int)

Type of a custom fl_contrast() function.

Use this signature to define your own custom fl_contrast() function together with fl_contrast_mode(FL_CONTRAST_CUSTOM). Example:

```c
Fl\_Color my\_contrast(Fl\_Color fg, Fl\_Color bg, Fl\_Fontsize fs, int context) {
  // calculate contrast and ...
  return color;
}

// call this early in your main() program:
fl\_contrast\_function(my\_contrast);
fl\_contrast\_mode(FL\_CONTRAST\_CUSTOM);
```

See also

- fl_contrast(Fl\_Color, Fl\_Color, Fl\_Fontsize, int)
- fl_contrast_mode(int)

34.1.3.2  **Fl\_Fontsize**

typedef int Fl\_Fontsize

Size of a font in pixels.

This is the approximate height of a font in pixels.
34.1.4 Enumeration Type Documentation

34.1.4.1 anonymous enum

Anonymous enum
FD "when" conditions.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_READ</td>
<td>Call the callback when there is data to be read.</td>
</tr>
<tr>
<td>FL_WRITE</td>
<td>Call the callback when data can be written without blocking.</td>
</tr>
<tr>
<td>FL_EXCEPT</td>
<td>Call the callback if an exception occurs on the file.</td>
</tr>
</tbody>
</table>

34.1.4.2 Fl_Arrow_Type

Enum Fl_Arrow_Type
Arrow types define the type of arrow drawing function.
FLTK schemes can draw several graphical elements in their particular way. One of these elements is an arrow type that can be in different GUI elements like scrollbars, choice buttons, and FLTK's Fl_Return_Button.

Note
This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

Since
1.4.0

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_ARROW_SINGLE</td>
<td>Single arrow, e.g. in Fl_Scrollbar.</td>
</tr>
<tr>
<td>FL_ARROW_DOUBLE</td>
<td>Double arrow, e.g. in Fl_Counter.</td>
</tr>
<tr>
<td>FL_ARROW_CHOICE</td>
<td>Dropdown box, e.g. in Fl_Choice.</td>
</tr>
<tr>
<td>FL_ARROW_RETURN</td>
<td>Return arrow, e.g. in Fl_Return_Button.</td>
</tr>
</tbody>
</table>

34.1.4.3 Fl_Boxtype

Enum Fl_Boxtype
FLTK standard box types.
This enum defines the standard box types included with FLTK.

Note
The documented enum Fl_Boxtype contains some values (names) with leading underscores, e.g. _FL_SHADOW_BOX. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

<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>Enumerator</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>FL_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_UP_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_DOWN_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_UP_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_DOWN_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_UP_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_THIN_DOWN_FRAME</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_BOX</td>
<td>see figure Standard Box Types</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</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_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_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_UP_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_UP_BOX</td>
</tr>
<tr>
<td>_FL_DIAMOND_DOWN_BOX</td>
<td>see figure Standard Box Types, use FL_DIAMOND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_OVAL_BOX</td>
<td>see figure Standard Box Types, use FL_OVAL_BOX</td>
</tr>
<tr>
<td>_FL_OSHADOW_BOX</td>
<td>see figure Standard Box Types, use FL_OSHADOW_BOX</td>
</tr>
<tr>
<td>_FL_OVAL_FRAME</td>
<td>see figure Standard Box Types, use FL_OVAL_FRAME</td>
</tr>
<tr>
<td>_FL_OFLAT_BOX</td>
<td>see figure Standard Box Types, use FL_OFLAT_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_BOX</td>
<td>plastic version of FL_UP_BOX, use FL_PLASTIC_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_BOX</td>
<td>plastic version of FL_DOWN_BOX, use FL_PLASTIC_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_FRAME</td>
<td>plastic version of FL_UP_FRAME, use FL_PLASTIC_UP_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_FRAME</td>
<td>plastic version of FL_DOWN_FRAME, use FL_PLASTIC_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_UP_BOX</td>
<td>plastic version of FL_THIN_UP_BOX, use FL_PLASTIC_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_DOWN_BOX</td>
<td>plastic version of FL_THIN_DOWN_BOX, use FL_PLASTIC_THIN_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_UP_BOX</td>
<td>plastic version of FL_ROUND_UP_BOX, use FL_PLASTIC_ROUND_UP_BOX</td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_DOWN_BOX</td>
<td>plastic version of FL_ROUND_DOWN_BOX, use FL_PLASTIC_ROUND_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_BOX</td>
<td>gtk+ version of FL_UP_BOX, use FL_GTK_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_BOX</td>
<td>gtk+ version of FL_DOWN_BOX, use FL_GTK_DOWN_BOX</td>
</tr>
<tr>
<td>_FL_GTK_UP_FRAME</td>
<td>gtk+ version of FL_UP_FRAME, use FL_GTK_UP_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_DOWN_FRAME</td>
<td>gtk+ version of FL_DOWN_FRAME, use FL_GTK_DOWN_FRAME</td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_BOX</td>
<td>gtk+ version of FL_THIN_UP_BOX, use FL_GTK_THIN_UP_BOX</td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_BOX</td>
<td>gtk+ version of FL_THIN_DOWN_BOX, use FL_GTK_THIN_DOWN_BOX</td>
</tr>
</tbody>
</table>
### 34.1 Enumerations.H File Reference

#### 34.1.4.4 Fl_Callback_Reason

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FL_GTK_THIN_UP_FRAME</td>
<td><code>gtk+</code> version of <code>FL_THIN_UP_FRAME</code>, use <code>FL_GTK_THIN_UP_FRAME</code></td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_FRAME</td>
<td><code>gtk+</code> version of <code>FL_THIN_DOWN_FRAME</code>, use <code>FL_GTK_THIN_DOWN_FRAME</code></td>
</tr>
<tr>
<td>_FL_GTK_ROUND_UP_BOX</td>
<td><code>gtk+</code> version of <code>FL_ROUND_UP_BOX</code>, use <code>FL_GTK_ROUND_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_GTK_ROUND_DOWN_BOX</td>
<td><code>gtk+</code> version of <code>FL_ROUND_DOWN_BOX</code>, use <code>FL_GTK_ROUND_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_GLEAM_UP_BOX</td>
<td><code>gleam</code> version of <code>FL_UP_BOX</code>, use <code>FL_GLEAM_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_BOX</td>
<td><code>gleam</code> version of <code>FL_DOWN_BOX</code>, use <code>FL_GLEAM_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_GLEAM_UP_FRAME</td>
<td><code>gleam</code> version of <code>FL_UP_FRAME</code>, use <code>FL_GLEAM_UP_FRAME</code></td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_FRAME</td>
<td><code>gleam</code> version of <code>FL_DOWN_FRAME</code>, use <code>FL_GLEAM_DOWN_FRAME</code></td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_UP_BOX</td>
<td><code>gleam</code> version of <code>FL_THIN_UP_BOX</code>, use <code>FL_GLEAM_THIN_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_DOWN_BOX</td>
<td><code>gleam</code> version of <code>FL_THIN_DOWN_BOX</code>, use <code>FL_GLEAM_THIN_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_UP_BOX</td>
<td><code>oxy</code> version of <code>FL_UP_BOX</code>, use <code>FL_OXY_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_DOWN_BOX</td>
<td><code>oxy</code> version of <code>FL_DOWN_BOX</code>, use <code>FL_OXY_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_UP_FRAME</td>
<td><code>oxy</code> version of <code>FL_UP_FRAME</code>, use <code>FL_OXY_UP_FRAME</code></td>
</tr>
<tr>
<td>_FL_OXY_DOWN_FRAME</td>
<td><code>oxy</code> version of <code>FL_DOWN_FRAME</code>, use <code>FL_OXY_DOWN_FRAME</code></td>
</tr>
<tr>
<td>_FL_OXY_THIN_UP_BOX</td>
<td><code>oxy</code> version of <code>FL_THIN_UP_BOX</code>, use <code>FL_OXY_THIN_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_THIN_DOWN_BOX</td>
<td><code>oxy</code> version of <code>FL_THIN_DOWN_BOX</code>, use <code>FL_OXY_THIN_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_THIN_UP_FRAME</td>
<td><code>oxy</code> version of <code>FL_THIN_UP_FRAME</code>, use <code>FL_OXY_THIN_UP_FRAME</code></td>
</tr>
<tr>
<td>_FL_OXY_THIN_DOWN_FRAME</td>
<td><code>oxy</code> version of <code>FL_THIN_DOWN_FRAME</code>, use <code>FL_OXY_THIN_DOWN_FRAME</code></td>
</tr>
<tr>
<td>_FL_OXY_ROUND_UP_BOX</td>
<td><code>oxy</code> version of <code>FL_ROUND_UP_BOX</code>, use <code>FL_OXY_ROUND_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_ROUND_DOWN_BOX</td>
<td><code>oxy</code> version of <code>FL_ROUND_DOWN_BOX</code>, use <code>FL_OXY_ROUND_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_UP_BOX</td>
<td><code>FL_OXY BUTTON_UP_BOX</code> (new boxtype ?), use <code>FL_OXY_BUTTON_UP_BOX</code></td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_DOWN_BOX</td>
<td><code>FL_OXY BUTTON_DOWN_BOX</code> (new boxtype ?), use <code>FL_OXY_BUTTON_DOWN_BOX</code></td>
</tr>
<tr>
<td>_FL_FREE_BOXTYPE</td>
<td>the first free box type for creation of new box types</td>
</tr>
</tbody>
</table>

**34.1.4.4 Fl_Callback_Reason**

```c
enum Fl_Callback_Reason
```

These constants describe why a callback is performed.

See also

- `Fl::callback_reason()`, `Fl_Widget::when()`, `Fl_When`

Generated by Doxygen
### FL_REASON

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_REASON_UNKNOWN</td>
<td>unknown or unset reason</td>
</tr>
<tr>
<td>FL_REASON_SELECTED</td>
<td>an item was selected</td>
</tr>
<tr>
<td>FL_REASON_DESELECTED</td>
<td>an item was de-selected</td>
</tr>
<tr>
<td>FL_REASON_RESELECTED</td>
<td>an item was re-selected (double-clicked).</td>
</tr>
<tr>
<td>FL_REASON_OPENED</td>
<td>an item was opened</td>
</tr>
<tr>
<td>FL_REASON_CLOSED</td>
<td>an item was closed</td>
</tr>
<tr>
<td>FL_REASON_DRAGGED</td>
<td>an item was dragged into a new place</td>
</tr>
<tr>
<td>FL_REASON_CANCELLLED</td>
<td>a dialog was cancelled</td>
</tr>
<tr>
<td>FL_REASON_CHANGED</td>
<td>the value of the widget was modified</td>
</tr>
<tr>
<td>FL_REASON_GOT_FOCUS</td>
<td>a widget received focus</td>
</tr>
<tr>
<td>FL_REASON_LOST_FOCUS</td>
<td>a widget lost focus</td>
</tr>
<tr>
<td>FL_REASON_RELEASED</td>
<td>the mouse button was released</td>
</tr>
</tbody>
</table>

### FL_Contrast_Mode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CONTRAST_NONE</td>
<td>always return foreground color</td>
</tr>
<tr>
<td>FL_CONTRAST_LEGACY</td>
<td>legacy (FLTK 1.3.x) contrast function</td>
</tr>
<tr>
<td>FL_CONTRAST_CIELAB</td>
<td>new (FLTK 1.4.0) default function</td>
</tr>
<tr>
<td>FL_CONTRAST_CUSTOM</td>
<td>optional custom contrast function</td>
</tr>
<tr>
<td>FL_CONTRAST_LAST</td>
<td>internal use only (invalid contrast mode)</td>
</tr>
</tbody>
</table>

### FL_Cursor

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_CURSOR_DEFAULT</td>
<td>the default cursor, usually an arrow:</td>
</tr>
<tr>
<td>FL_CURSOR_ARROW</td>
<td>an arrow pointer:</td>
</tr>
<tr>
<td>FL_CURSOR_CROSS</td>
<td>crosshair:</td>
</tr>
<tr>
<td>FL_CURSOR_WAIT</td>
<td>busy indicator (for instance hourglass): ,</td>
</tr>
<tr>
<td>FL_CURSOR_INSERT</td>
<td>I-beam:</td>
</tr>
<tr>
<td>FL_CURSOR_HAND</td>
<td>pointing hand:</td>
</tr>
<tr>
<td>FL_CURSOR_HELP</td>
<td>question mark pointer:</td>
</tr>
<tr>
<td>FL_CURSOR_MOVE</td>
<td>4-pointed arrow or hand:</td>
</tr>
<tr>
<td>FL_CURSOR_NS</td>
<td>up/down resize:</td>
</tr>
<tr>
<td>FL_CURSOR_WE</td>
<td>left/right resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NWSE</td>
<td>diagonal resize:</td>
</tr>
<tr>
<td>FL_CURSOR_NESW</td>
<td>diagonal resize:</td>
</tr>
<tr>
<td>FL_CURSOR_N</td>
<td>upwards resize:</td>
</tr>
</tbody>
</table>
### 34.1 Enumerations.H File Reference

#### 34.1.4.7 Fl_Damage

**enum Fl_Damage**  
Damage masks.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_DAMAGE_CHILD</td>
<td>A child needs to be redrawn.</td>
</tr>
<tr>
<td>FL_DAMAGE_EXPOSE</td>
<td>The window was exposed.</td>
</tr>
<tr>
<td>FL_DAMAGE_SCROLL</td>
<td>The Fl_Scroll widget was scrolled.</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>

#### 34.1.4.8 Fl_Event

**enum Fl_Event**  
Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

Events can also come from other programs like the window manager.

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

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

**See also**

Fl::event_text(), Fl::event_key(), class Fl::

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NO_EVENT</td>
<td>No event.</td>
</tr>
<tr>
<td>FL_PUSH</td>
<td>A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling Fl::event_button(). You find out the mouse position by calling Fl::event_x() and Fl::event_y(). A widget indicates that it &quot;wants&quot; the mouse click by returning non-zero from its Fl_Widget::handle() method. It will then become the Fl::pushed() widget and will get FL_DRAG and the matching FL_RELEASE events. If Fl_Widget::handle() returns zero then FLTK will try sending the FL_PUSH to another widget.</td>
</tr>
<tr>
<td>FL_RELEASE</td>
<td>A mouse button has been released. You can find out what button by calling Fl::event_button(). In order to receive the FL_RELEASE event, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_ENTER</td>
<td>The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.</td>
</tr>
<tr>
<td>FL_LEAVE</td>
<td>The mouse has moved out of the widget. In order to receive the FL_LEAVE event, the widget must return non-zero when handling FL_ENTER.</td>
</tr>
<tr>
<td>FL_DRAG</td>
<td>The mouse has moved with a button held down. The current button state is in Fl::event_state(). The mouse position is in Fl::event_x() and Fl::event_y(). In order to receive FL_DRAG events, the widget must return non-zero when handling FL_PUSH.</td>
</tr>
<tr>
<td>FL_FOCUS</td>
<td>This indicates an attempt to give a widget the keyboard focus. If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its handle() method. It then becomes the Fl::focus() widget and gets FL_KEYDOWN, FL_KEYUP, and FL_UNFOCUS events. The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check Fl::event_key() to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.</td>
</tr>
<tr>
<td>FL_UNFOCUS</td>
<td>This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.</td>
</tr>
<tr>
<td>Enumerator</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FL_KEYDOWN</td>
<td>A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN. The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length(). If you use the key handle() should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a FL_SHORTCUT event. To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events. If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into non-ASCII characters. FL_KEYUP events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding FL_KEYDOWN event because focus may have changed between events.</td>
</tr>
<tr>
<td>FL_KEYBOARD</td>
<td>Equivalent to FL_KEYDOWN. See also FL_KEYDOWN</td>
</tr>
<tr>
<td>FL_KEYUP</td>
<td>Key release event. See also FL_KEYDOWN</td>
</tr>
<tr>
<td>FL_CLOSE</td>
<td>The user clicked the close button of a window. This event is used internally only to trigger the callback of Fl_Window derived classed. The default callback closes the window calling Fl_Window::hide().</td>
</tr>
<tr>
<td>FL_MOVE</td>
<td>The mouse has moved without any mouse buttons held down. This event is sent to the Fl::belowmouse() widget. In order to receive FL_MOVE events, the widget must return non-zero when handling FL_ENTER.</td>
</tr>
<tr>
<td>FL_SHORTCUT</td>
<td>If the Fl::focus() widget is zero or ignores an FL_KEYBOARD event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. FL_SHORTCUT is first sent to the Fl::belowmouse() widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes! You can also make &quot;global&quot; shortcuts by using Fl::add_handler(). A global shortcut will work no matter what windows are displayed or which one has the focus.</td>
</tr>
<tr>
<td>FL_DEACTIVATE</td>
<td>This widget is no longer active, due to Fl_Widget::deactivate() being called on it or one of its parents. Fl_Widget::active() may still be true after this, the widget is only active if Fl_Widget::active() is true on it and all its parents (use Fl_Widget::active_r() to check this).</td>
</tr>
<tr>
<td>FL_ACTIVATE</td>
<td>This widget is now active, due to Fl_Widget::activate() being called on it or one of its parents.</td>
</tr>
</tbody>
</table>
### Enumerator

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FL_HIDE</strong></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><strong>FL_SHOW</strong></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 Fl_Windows respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass FL_SHOW to the base class <code>Fl_Widget::handle()</code> method!</td>
</tr>
<tr>
<td><strong>FL_PASTE</strong></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><strong>FL_SELECTIONCLEAR</strong></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><strong>FL_MOUSEWHEEL</strong></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><strong>FL_DND_ENTER</strong></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><strong>FL_DND_DRAG</strong></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><strong>FL_DND_LEAVE</strong></td>
<td>The mouse has moved out of the widget.</td>
</tr>
<tr>
<td><strong>FL_DND_RELEASE</strong></td>
<td>The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following FL_PASTE event.</td>
</tr>
<tr>
<td><strong>FL_SCREEN_CONFIGURATION_CHANGED</strong></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><strong>FL_FULLSCREEN</strong></td>
<td>The fullscreen state of the window has changed. This event is sent to the window's handle method.</td>
</tr>
<tr>
<td><strong>FL_ZOOM_GESTURE</strong></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><strong>FL_ZOOM_EVENT</strong></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>

### 34.1.4.9 Fl_Labeltype

```cpp
enum Fl_Labeltype
```

The `labeltype()` method sets the type of the label.
Note

The documented enum `Fl_Labeltype` contains some values (names) with leading underscores, e.g. `_FL_IMAGE_LABEL`. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

The following standard label types are included:

**Enumerator**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_NORMAL_LABEL</td>
<td>draws the text (0)</td>
</tr>
<tr>
<td>FL_NO_LABEL</td>
<td>does nothing</td>
</tr>
<tr>
<td>_FL_SHADOW_LABEL</td>
<td>draws a drop shadow under the text</td>
</tr>
<tr>
<td>_FL_ENGRAVED_LABEL</td>
<td>draws edges as though the text is engraved</td>
</tr>
<tr>
<td>_FL_EMBOSSED_LABEL</td>
<td>draws edges as though the text is raised</td>
</tr>
<tr>
<td>_FL_MULTI_LABEL</td>
<td>draws a composite label</td>
</tr>
<tr>
<td>_FL_ICON_LABEL</td>
<td>draws the icon associated with the text</td>
</tr>
<tr>
<td>_FL_IMAGE_LABEL</td>
<td>the label displays an &quot;icon&quot; based on a Fl_Image</td>
</tr>
<tr>
<td>FL_FREE_LABELTYPE</td>
<td>first free labeltype to use for creating own labeltypes</td>
</tr>
</tbody>
</table>

---

### 34.1.4.10 Fl_Orientation

**enum Fl_Orientation**

Fl_Orientation describes the orientation of a GUI element. FLTK schemes can draw several graphical elements, for instance arrows, pointing at different directions. This enum defines the direction to use for drawing a particular GUI element. The definition of this enum was chosen such that the enum value can be multiplied by 45 to get a rotation angle in degrees starting at the horizontal axis (0 = right, 1 = NE, 2 = up, ...) that can be used with `fl_rotate()`. Note: angle is counter-clockwise in degrees.

The 'unspecified' value `FL_ORIENT_NONE` shall be used for elements that would usually not be rotated, like the return arrow of the Fl_Return_Button. It can still be used as an angle though since it is the same value as `FL__ORIENT_RIGHT` (0 degrees).

**Note**

This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

Since

1.4.0

**Enumerator**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_ORIENT_NONE</td>
<td>GUI element direction is unspecified.</td>
</tr>
<tr>
<td>FL_ORIENT_RIGHT</td>
<td>GUI element pointing right (0°)</td>
</tr>
<tr>
<td>FL_ORIENT_NE</td>
<td>GUI element pointing NE (45°)</td>
</tr>
<tr>
<td>FL_ORIENT_UP</td>
<td>GUI element pointing up (90°)</td>
</tr>
<tr>
<td>FL_ORIENT_NW</td>
<td>GUI element pointing NW (135°)</td>
</tr>
<tr>
<td>FL_ORIENT_LEFT</td>
<td>GUI element pointing left (180°)</td>
</tr>
<tr>
<td>FL_ORIENT_SW</td>
<td>GUI element pointing SW (225°)</td>
</tr>
</tbody>
</table>
34.1.4.11  Fl_When

enum Fl_When
These constants determine when a callback is performed. Fl_When is a bit field. Some values are merely shortcuits for common bit combinations. New flags may be added in the future, so it's important to mask the required bit when reading via when().

Note
Some widgets may not fully support FL_WHEN_... flags.

See also
Fl_Widget::when(), Fl::callback_reason(), Fl_Callback_Reason, Fl_Widget::do_callback()

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_WHEN_NEVER</td>
<td>Never call the callback.</td>
</tr>
<tr>
<td>FL_WHEN_CHANGED</td>
<td>Do the callback only when the widget value changes.</td>
</tr>
<tr>
<td>FL_WHEN_NOT_CHANGED</td>
<td>Do the callback whenever the user interacts with the widget.</td>
</tr>
<tr>
<td>FL_WHEN_RELEASE</td>
<td>Do the callback when the button or key is released and the value changes.</td>
</tr>
<tr>
<td>FL_WHEN_RELEASE_ALWAYS</td>
<td>Do the callback when the button or key is released, even if the value doesn't change.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY</td>
<td>Do the callback when the user presses the ENTER key and the value changes.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY_ALWAYS</td>
<td>Do the callback when the user presses the ENTER key, even if the value doesn't change.</td>
</tr>
<tr>
<td>FL_WHEN_ENTER_KEY_CHANGED</td>
<td>Do callbacks whether the value changed or not, and when the ENTER key is pressed.</td>
</tr>
<tr>
<td>FL_WHEN_CLOSED</td>
<td>Do the callback when a child of Fl Tabs is closed.</td>
</tr>
</tbody>
</table>

34.1.5  Function Documentation

34.1.5.1  fl_box()

Fl_Boxtype fl_box (Fl_Boxtype b) [inline]
Get the filled version of a frame.
If no filled version of a given frame exists, the behavior of this function is undefined and some random box or frame is returned.

34.1.5.2  fl_color_cube()

Fl_Color fl_color_cube (int r,
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:
\[
\text{fl_color_cube}(R \ast (\text{FL_NUM_RED} - 1) / 255, \\
G \ast (\text{FL_NUM_GREEN} - 1) / 255, \\
B \ast (\text{FL_NUM_BLUE} - 1) / 255);
\]

34.1.5.3 fl_define_FL_EMBOSSED_LABEL()

\text{FL_Labeltype} fl_define_FL_EMBOSSED_LABEL ()

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
Internal use only.

34.1.5.4 fl_define_FL_ENGRAVED_LABEL()

\text{FL_Labeltype} fl_define_FL_ENGRAVED_LABEL ()

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
Internal use only.

34.1.5.5 fl_define_FL_ICON_LABEL()

\text{FL_Labeltype} fl_define_FL_ICON_LABEL ()

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
Internal use only.

34.1.5.6 fl_define_FL_IMAGE_LABEL()

\text{FL_Labeltype} fl_define_FL_IMAGE_LABEL ()

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
Internal use only.

34.1.5.7 fl_define_FL_MULTI_LABEL()

\text{FL_Labeltype} fl_define_FL_MULTI_LABEL ()

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
Internal use only.

34.1.5.8 fl_define_FL_SHADOW_LABEL()

\text{FL_Labeltype} fl_define_FL_SHADOW_LABEL ()

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
Internal use only.

34.1.5.9 fl_down()

\text{FL_Boxtype} fl_down ( 
\text{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.

34.1.5.10 fl_frame()

\text{FL_Boxtype} fl_frame ( 
\text{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.

### 34.1.5.11 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:

```c
fl_gray_ramp(i * (FL_NUM_GRAY - 1) / 255)
```

### 34.1.6 Variable Documentation

#### 34.1.6.1 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.

#### 34.1.6.2 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.

#### 34.1.6.3 FL_NORMAL_SIZE

```c
Fl_Fontsize FL_NORMAL_SIZE [extern]
```

normal font size

### 34.2 Enumerations.H

Go to the documentation of this file.

1 //
2 // Enumerations for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2023 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 #ifndef Fl_Enumerations_H
17 #define Fl_Enumerations_H
18 #include <FL/fl_config.h>
19 # include "Fl_Export.H"
20 /
21 +%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%22 + FL_ABI_VERSION is defined by configure or CMake since FLTK 1.3.4.
23 +%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%24 + For more informations on FL_ABI_VERSION see README.abi-version.txt.
25 +%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%26 +
27 +
28 +
29 +
30 +
31 +
32 #include <FL/fl_config.h>
33 # include "Fl_Export.H"
34
# include "fl_types.h"
# include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL

#define FL_MAJOR_VERSION 1
#define FL_MINOR_VERSION 4
#define FL_PATCH_VERSION 0
#define FL_VERSION ( (double)FL_MAJOR_VERSION + 
                 (double)FL_MINOR_VERSION * 0.01 + 
                 (double)FL_PATCH_VERSION * 0.0001 )
#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)

// group: Version Numbers

enum Fl_Event { // events
    FL_NO_EVENT = 0,
    FL_PUSH = 1,
    FL_RELEASE = 2,
    FL_ENTER = 3,
    FL_LEAVE = 4,
    FL_DRAG = 5,
    FL_FOCUS = 6,
    FL_KEYBOARD = 8,
    FL_KEYDOWN = 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,
}
enum Fl_SelectionClear = 18,
enum FL_MOUSEWHEEL = 19,
enum FL_DND_ENTER = 20,
enum FL_DND_DRAG = 21,
enum FL_DND_LEAVE = 22,
enum FL_DND_RELEASE = 23,
enum FL_SCREEN_CONFIGURATION_CHANGED = 24,
enum FL_FULLSCREEN = 25,
enum FL_ZOOM_GESTURE = 26,
enum FL_ZOOM_EVENT = 27
// DEV NOTE: Keep this list in sync with FL/names.h
};

enum Fl_When { // Fl_Widget::when():
enum FL_WHEN_NEVER = 0,
enum FL_WHEN_CHANGED = 1,
enum FL_WHEN_NOT_CHANGED = 2,
enum FL_WHEN_RELEASE = 4,
enum FL_WHEN_RELEASE_ALWAYS = 6,
enum FL_WHEN_ENTER_KEY = 8,
enum FL_WHEN_ENTER_KEY_ALWAYS = 10,
enum FL_WHEN_ENTER_KEY_CHANGED = 11,
enum FL_WHEN_CLOSED = 16
}; // group: When Conditions

enum Fl_Callback_Reason { // group: Callback Reasons
enum FL_REASON_UNKNOWN = 0,
enum FL_REASON_SELECTED,
enum FL_REASON_DESELECTED,
enum FL_REASON_RESELECTED,
enum FL_REASON_OPENED,
enum FL_REASON_CLOSED,
enum FL_REASON_DRAGGED,
enum FL_REASON_CANCELLED,
enum FL_REASON_CHANGED,
enum FL_REASON_GOT_FOCUS,
enum FL_REASON_LOST_FOCUS,
enum FL_REASON_RELEASED
};

// FIXME: These codes collide with valid Unicode keys
#define FL_Button 0xe8
#define FL_BackSpace 0x08
#define FL_Tab 0x09
#define FL_Isol Key 0x0c
#define FL_Enter 0x0d
#define FL_Pause 0x13
#define FL_Scroll_Lock 0x14
#define FL_Escape 0x1b
#define FL_Kana 0x2e
#define FL_Eisu 0x2f
#define FL_Yen 0x30
#define FL_JIS_Underscore 0x31
#define FL_Home 0x50
#define FL_Left 0x51
#define FL_Up 0x52
#define FL_Right 0x53
#define FL_Down 0x54
#define FL_Page_Up 0x55
#define FL_Page_Down 0x56
#define FL_End 0x57
#define FL_Print 0x61
#define FL_Insert 0x63
#define FL_Menu 0x67
#define FL_Help 0x68
#define FL_F1 0xf7
#define FL_KP 0x80
#define FL_KP_Enter 0x8d
#define FL_KP_Last 0xf4
#define FL_F 0xb
#define FL_F_Last 0xe
#define FL_F1 0xe1
#define FL_F2 0xe2
#define FL_F3 0xe3
#define FL_F4 0xe4
#define FL_F5 0xe5
#define FL_F6 0xe7
#define FL_F7 0xe8

Generated by Doxygen
#define FL_Alt_L 0xffe9
#define FL_Alt_R 0xffea
#define FL_Delete 0xffff

// These primarily map to the XFree86 keysym range
#define FL_Volume_Down 0xEF11
#define FL_Volume_Mute 0xEF12
#define FL_Volume_Up 0xEF13
#define FL_Media_Play 0xEF14
#define FL_Media_Stop 0xEF15
#define FL_Media_Prev 0xEF16
#define FL_Media_Next 0xEF17
#define FL_Home_Page 0xEF18
#define FL_Back 0xEF26
#define FL_Forward 0xEF27
#define FL_Stop 0xEF28
#define FL_Refresh 0xEF29
#define FL_Sleep 0xEF2F
#define FL_Favorites 0xEF30

#define FL_LEFT_MOUSE 1
#define FL_MIDDLE_MOUSE 2
#define FL_RIGHT_MOUSE 3

#define FL_SHIFT 0x00010000
#define FL_CAPS_LOCK 0x00020000
#define FL_CTRL 0x00040000
#define FL_ALT 0x00080000
#define FL_NUM_LOCK 0x00100000
#define FL_META 0x00400000
#define FL_SCROLL_LOCK 0x00800000
#define FL_BUTTON1 0x01000000
#define FL_BUTTON2 0x02000000
#define FL_BUTTON3 0x04000000
#define FL_BUTTON(n) (0x00800000 «(n))

#define FL_KEY_MASK 0x0000ffff

enum Fl_Boxtype { // boxtypes (if you change these you must also change fl_boxtype.cxx):
    FL_NO_BOX = 0,
    FL_FLAT_BOX,
    FL_UP_BOX,
    FL_DOWN_BOX,
    FL_UP_FRAME,
    FL_DOWN_FRAME,
    FL_THIN_UP_BOX,
    FL_THIN_DOWN_BOX,
    FL_THIN_UP_FRAME,
    FL_THIN_DOWN_FRAME,
    FL_ENGRAVED_BOX,
    FL_EMBOSSED_BOX,
    FL_ENGRAVED_FRAME,
    FL_EMBOSSED_FRAME,
    FL_BORDER_BOX,
    FL_BORDER_FRAME,
    FL_SHADOW_BOX,
    FL_SHADOW_FRAME,
    FL_ROUNDED_BOX,
    FL_ROUNDED_FRAME,
    FL_OVAL_BOX
};

Generated by Doxygen
#ifndef FL_DOXYGEN

extern Fl_Boxtype fl_define_FL_ROUND_UP_BOX();
#define FL_ROUND_UP_BOX fl_define_FL_ROUND_UP_BOX()
#define FL_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_ROUND_UP_BOX()+1)
extern Fl_Boxtype fl_define_FL_SHADOW_BOX();
#define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()
#define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)
extern Fl_Boxtype fl_define_FL_ROUNDED_BOX();
#define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()
#define FL_ROUNDED_FRAME (Fl_Boxtype)(fl_define_FL_ROUNDED_BOX()+2)
extern Fl_Boxtype fl_define_FL_RFLAT_BOX();
#define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()
extern Fl_Boxtype fl_define_FL_RSHADOW_BOX();
#define FL_RSHADOW_BOX fl_define_FL_RSHADOW_BOX()
#define FL_RSHADOW_FRAME (Fl_Boxtype)(fl_define_FL_RSHADOW_BOX()+2)
extern Fl_Boxtype fl_define_FL_DIAMOND_BOX();
#define FL_DIAMOND_UP_BOX fl_define_FL_DIAMOND_BOX()
#define FL_DIAMOND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)
extern Fl_Boxtype fl_define_FL_OVAL_BOX();
#define FL_OVAL_BOX fl_define_FL_OVAL_BOX()
#define FL_OSHADOW_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+1)
#define FL_OSHADOW_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)
#define FL_OFLAT_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+3)
extern Fl_Boxtype fl_define_FL_PLASTIC_UP_BOX();
#define FL_PLASTIC_UP_BOX fl_define_FL_PLASTIC_UP_BOX()
#define FL_PLASTIC_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+1)
#define FL_PLASTIC_UP_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+2)
#define FL_PLASTIC_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+3)
#define FL_PLASTIC_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+4)
#define FL_PLASTIC_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+5)
#define FL_PLASTIC_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+6)
#define FL_PLASTIC_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_PLASTIC_UP_BOX()+7)
#define FL_FREE_BOXTYPE

#endif FL_DOXYGEN
#define FL_GTK_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GTK_UP_BOX()+9)

extern FL_EXPORT Fl_Boxtype fl_define_FL_GLEAM_UP_BOX();
#define FL_GLEAM_UP_BOX fl_define_FL_GLEAM_UP_BOX()
#define FL_GLEAM_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+1)
#define FL_GLEAM_UP_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+2)
#define FL_GLEAM_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+3)
#define FL_GLEAM_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+4)
#define FL_GLEAM_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+5)
#define FL_GLEAM_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+6)
#define FL_GLEAM_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_GLEAM_UP_BOX()+7)

extern FL_EXPORT Fl_Boxtype fl_define_FL_OXY_UP_BOX();
#define FL_OXY_UP_BOX fl_define_FL_OXY_UP_BOX()
#define FL_OXY_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+1)
#define FL_OXY_UP_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+2)
#define FL_OXY_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+3)
#define FL_OXY_THIN_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+4)
#define FL_OXY_THIN_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+5)
#define FL_OXY_ROUND_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+6)
#define FL_OXY_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+7)
#define FL_OXY_THIN_UP_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+8)
#define FL_OXY_THIN_DOWN_FRAME (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+9)
#define FL_OXY_BUTTON_UP_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+10)
#define FL_OXY_BUTTON_DOWN_BOX (Fl_Boxtype)(fl_define_FL_OXY_UP_BOX()+11)

#endif // ! FL_DOXYGEN

// conversions of box types to other box types:
inline Fl_Boxtype fl_box(Fl_Boxtype b) {
    return (Fl_Boxtype)((b<FL_UP_BOX||b%4>1)?b:(b-2));
}
inline Fl_Boxtype fl_down(Fl_Boxtype b) {
    return (Fl_Boxtype)((b<FL_UP_BOX)?b:(b|1));
}
inline Fl_Boxtype fl_frame(Fl_Boxtype b) {
    return (Fl_Boxtype)((b%4<2)?b:(b+2));
}

// back-compatibility box types:
#define FL_FRAME FL_ENGRAVED_FRAME
#define FL_FRAME_BOX FL_ENGRAVED_BOX
#define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
#define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX

// group: Box Types
enum Fl_Labeltype { // labeltypes:
    FL_NORMAL_LABEL = 0,
    FL_NO_LABEL,
    _FL_SHADOW_LABEL,
    _FL_ENGRAVED_LABEL,
    _FL_EMBOSSED_LABEL,
    _FL_MULTI_LABEL,
    _FL_ICON_LABEL,
    _FL_IMAGE_LABEL,
    FL_FREE_LABELTYPE
};

#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
extern Fl_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL();
#define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()
extern Fl_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL();
#define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()
extern Fl_Labeltype FL_EXPORT fl_define_FL_EMBOSSED_LABEL();
#define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()
extern Fl_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL();
#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()
extern Fl_Labeltype FL EXPORT fl_define_FL_ICON_LABEL();
#define FL_ICON_LABEL fl_define_FL_ICON_LABEL()
extern Fl_Labeltype FL EXPORT fl_define_FL_IMAGE_LABEL();
#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

typedef unsigned Fl_Align;
const Fl_Align FL_ALIGN_CENTER = 0x0000;
const Fl_Align FL_ALIGN_TOP = 0x0001;
const Fl_Align FL_ALIGN_BOTTOM = 0x0002;
const Fl_Align FL_ALIGN_LEFT = 0x0004;

typedef unsigned Fl_Align;
981 const Fl_Align FL_ALIGN_RIGHT = 0x0008;
982 const Fl_Align FL_ALIGN_INSIDE = 0x0010;
983 const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020;
984 const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000;
985 const Fl_Align FL_ALIGN_CLIP = 0x0040;
986 const Fl_Align FL_ALIGN_WRAP = 0x0080;
987 const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100;
988 const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120;
989 const Fl_Align FL_ALIGN_IMAGE_BACKDROP = 0x0200;
990 const Fl_Align FL_ALIGN_TOP_LEFT = FL_ALIGN_TOP | FL_ALIGN_LEFT;
991 const Fl_Align FL_ALIGN_TOP_RIGHT = FL_ALIGN_TOP | FL_ALIGN_RIGHT;
992 const Fl_Align FL_ALIGN_BOTTOM_LEFT = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT;
993 const Fl_Align FL_ALIGN_BOTTOM_RIGHT = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT;
994 const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007;
995 const Fl_Align FL_ALIGN_RIGHT_TOP = 0x000b;
996 const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d;
997 const Fl_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e;
998 const Fl_Align FL_ALIGN_NOWRAP = 0x0000;
999 const Fl_Align FL_ALIGN_POSITION_MASK = 0x000f;
1000 const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320;
1001 typedef int Fl_Font;
1002 typedef int Fl_Fontsize;
1003 extern FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE;
1004 const Fl_Font FL_HELVETICA = 0;
1005 const Fl_Font FL_HELVETICA_BOLD = 1;
1006 const Fl_Font FL_HELVETICA_ITALIC = 2;
1007 const Fl_Font FL_HELVETICA_BOLD_ITALIC = 3;
1008 const Fl_Font FL_COURIER = 4;
1009 const Fl_Font FL_COURIER_BOLD = 5;
1010 const Fl_Font FL_COURIER_ITALIC = 6;
1011 const Fl_Font FL_COURIER_BOLD_ITALIC = 7;
1012 const Fl_Font FL_TIMES = 8;
1013 const Fl_Font FL_TIMES_BOLD = 9;
1014 const Fl_Font FL_TIMES_ITALIC = 10;
1015 const Fl_Font FL_TIMES_BOLD_ITALIC = 11;
1016 const Fl_Font FL_SYMBOL = 12;
1017 const Fl_Font FL_SCREEN = 13;
1018 const Fl_Font FL_SCREEN_BOLD = 14;
1019 const Fl_Font FL_ZAPF_DINGBATS = 15;
1020 const Fl_Font FL_FREE_FONT = 16;
1021 const Fl_Font FL_BOLD = 1;
1022 const Fl_Font FL_ITALIC = 2;
1023 const Fl_Font FL_BOLD_ITALIC = 3;
1024 typedef unsigned int Fl_Color;
1025 // Standard colors. These are used as default colors in widgets and altered as necessary
1026 const Fl_Color FL_FOREGROUND_COLOR = 0;
1027 const Fl_Color FL_BACKGROUND2_COLOR = 7;
1028 const Fl_Color FL_INACTIVE_COLOR = 8;
1029 const Fl_Color FL_SELECTION_COLOR = 15;
1030 const Fl_Color FL_GRAY0 = 32; // 'A'
1031 const Fl_Color FL_DARK3 = 39; // 'H'
1032 const Fl_Color FL_DARK2 = 45; // 'N'
1033 const Fl_Color FL_DARK1 = 47; // 'P'
1034 const Fl_Color FL_BACKGROUND_COLOR = 49; // 'R' default background color
1035 const Fl_Color FL_LIGHT1 = 50; // 'S'
1036 const Fl_Color FL_LIGHT2 = 52; // 'U'
1037 const Fl_Color FL_LIGHT3 = 54; // 'W'
1038 // FLTK provides a 5x8x5 color cube that is used with colormap visuals
1039 Generated by Doxygen
const Fl_Color FL_BLACK = 56;
const Fl_Color FL_RED = 88;
const Fl_Color FL_GREEN = 63;
const Fl_Color FL_YELLOW = 95;
const Fl_Color FL_BLUE = 216;
const Fl_Color FL_MAGENTA = 248;
const Fl_Color FL_CYAN = 223;
const Fl_Color FL_DARK_RED = 72;
const Fl_Color FL_DARK_GREEN = 60;
const Fl_Color FL_DARK_YELLOW = 76;
const Fl_Color FL_DARK_BLUE = 136;
const Fl_Color FL_DARK_MAGENTA = 152;
const Fl_Color FL_DARK_CYAN = 140;
const Fl_Color FL_WHITE = 255;
define FL_FREE_COLOR (Fl_Color)16
#define FL_NUM_FREE_COLOR 16
#define FL_GRAY_RAMP (Fl_Color)32
#define FL_GRAY 24
#define FL_BACKGROUND_COLOR
#define FL_COLOR_CUBE (Fl_Color)56
#define FL_NUM_RED 5
#define FL_NUM_GREEN 8
#define FL_NUM_BLUE 5

FL_EXPORT Fl_Color fl_inactive(Fl_Color c);

typedef Fl_Color (Fl_Contrast_Function)(Fl_Color, Fl_Color, Fl_Fontsize, int);

FL_EXPORT void fl_contrast_function(Fl_Contrast_Function *f);

enum Fl_Contrast_Mode {
   FL_CONTRAST_NONE = 0,
   FL_CONTRAST_LEGACY,
   FL_CONTRAST_CIELAB,
   FL_CONTRAST_CUSTOM,
   FL_CONTRAST_LAST
};

// The following functions are defined and documented in src/fl_contrast.cxx
FL_EXPORT void fl_contrast_level(int level);
FL_EXPORT int fl_contrast_level();
FL_EXPORT void fl_contrast_mode(int mode);
FL_EXPORT int fl_contrast_mode();

FL_EXPORT Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg, Fl_Fontsize fs = 0, int context = 0);
FL_EXPORT double fl_lightness(Fl_Color color);
FL_EXPORT double fl_luminance(Fl_Color color);

// Other color functions are defined and documented in src/fl_color.cxx
FL_EXPORT Fl_Color fl_color_average(Fl_Color c1, Fl_Color c2, float weight);
inline Fl_Color fl_lighter(Fl_Color c) { return fl_color_average(c, FL_WHITE, .67f); }
inline Fl_Color fl_darker(Fl_Color c) { return fl_color_average(c, FL_BLACK, .67f); }
inline Fl_Color fl_rgb_color(uchar r, uchar g, uchar b) {
   if (!r && !g && !b) return FL_BLACK;
   else return (Fl_Color)(((((r « 8) | g) « 8) | b) « 8);
}
inline Fl_Color fl_rgb_color(uchar g) {
   if (!g) return FL_BLACK;
   else return (Fl_Color)(((((g « 8) | g) « 8) | g) « 8);
}
inline Fl_Color fl_gray_ramp(int i) {return (Fl_Color)(i+FL_GRAY_RAMP);}
inline Fl_Color fl_color_cube(int r, int g, int b) {
   return (Fl_Color)((b *FL_NUM_RED + r) * FL_NUM_GREEN + g + FL_COLOR_CUBE);}

// group: Colors

// FIXME: We should renumber these, but that will break the ABI +/
enum Fl_Cursor {
   FL_CURSOR_DEFAULT = 0, // U+2196
   FL_CURSOR_ARROW = 35, // U+2196
   FL_CURSOR_CROSS = 66, // U+FF0B
   FL_CURSOR_WAIT = 76, // U+231A, U+231B
   FL_CURSOR_INSERT = 77, // U+2316
   FL_CURSOR_HAND = 31, // U+261C
   FL_CURSOR_HELP = 47,
   FL_CURSOR_MOVE = 27, // U+2725, U+270B

Generated by Doxygen
/* Resize indicators */
FL_CURSOR_NS = 78, // U+21D5
FL_CURSOR_WE = 79, // U+21D4
FL_CURSOR_NWSE = 80, // U+2921
FL_CURSOR_NESW = 81, // U+2922
FL_CURSOR_N = 70, // U+2912
FL_CURSOR_NE = 69, // U+2197
FL_CURSOR_E = 49, // U+21E5
FL_CURSOR_SE = 8, // U+21F2
FL_CURSOR_S = 9, // U+2913
FL_CURSOR_SW = 7, // U+2199
FL_CURSOR_W = 36, // U+21E4
FL_CURSOR_NW = 68, // U+21F1
FL_CURSOR_NONE = 255;

enum { // values for "when" passed to Fl::add_fd()
   FL_READ = 1,
   FL_WRITE = 4,
   FL_EXCEPT = 8
};

enum Fl_Mode {
   FL_RGB = 0,
   FL_INDEX = 1,
   FL_SINGLE = 0,
   FL_DOUBLE = 2,
   FL_ACCUM = 4,
   FL_RGBA = 8,
   FL_DEPTH = 16,
   FL_STENCIL = 32,
   FL_RGB8 = 64,
   FL_MULTISAMPLE = 128,
   FL_STEREO = 256,
   FL_FAKE_SINGLE = 512, // Fake single buffered windows using double-buffer
   FL_OPENGL3 = 1024
};

(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
};

// FLTK 1.0.x compatibility definitions (FLTK_1_0_COMPAT) dropped in 1.4.0

#define FL_IMAGE_WITH_ALPHA 0x40000000

enum Fl_Arrow_Type {
   FL_ARROW_SINGLE = 0x01,
   FL_ARROW_DOUBLE = 0x02,
   FL_ARROW_CHOICE = 0x03,
   FL_ARROW_RETURN = 0x04
};

enum Fl_Orientation {
   FL_ORIENTATION_NONE = 0x00,
   FL_ORIENTATION_RIGHT = 0x00,
   FL_ORIENTATION_NE = 0x01,
   FL_ORIENTATION_UP = 0x02,
   FL_ORIENTATION_NW = 0x03,
   FL_ORIENTATION_LEFT = 0x04,
   FL_ORIENTATION_SW = 0x05,
   FL_ORIENTATION_DOWN = 0x06,
   FL_ORIENTATION_SE = 0x07
};

#endif

34.3  filename.H File Reference

File names and URI utility functions.
#include "Fl_Export.H"
#include <FL/platform_types.h>
Macros

- `#define FL_PATH_MAX 2048`
  
  *all path buffers should use this length*

TypeDefs

- `typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)`
  
  *File sorting function.*

Functions

- `void fl_decode_uri (char *uri)`
  
  *Decodes a URL-encoded string.*

- `int fl_filename_absolute (char *, int tolen, const char *from)`
  
  *Makes a filename absolute from a relative filename.*

- `int fl_filename_expand (char *, int tolen, const char *from)`
  
  *Expands a filename containing shell variables and tilde (~).*

- `const char * fl_filename_ext (const char *)`
  
  *Gets the extension of a filename.*

- `void fl_filename_free_list (struct dirent ***, int n)`
  
  *Free the list of filenames that is generated by fl_filename_list().*

- `int fl_filename_isdir (const char *)`
  
  *Determines if a file exists and is a directory from its filename.*

- `int fl_filename_isdir (const char *name)`
  
  *Determines if a file exists and is a directory from its filename.*

- `int fl_filename_list (const char *, struct dirent ***, Fl_File_Sort_F *s=fl_numericsort)`
  
  *Portable and const-correct wrapper for the scandir() function.*

- `int fl_filename_match (const char *, const char *)`
  
  *Checks if a string s matches a pattern p.*

- `const char * fl_filename_name (const char *)`
  
  *Gets the file name from a path.*

- `int fl_filename_relative (char *, int tolen, const char *from)`
  
  *Makes a filename relative to the current working directory.*

- `char * fl_filename_setext (char *, int tolen, const char *)`
  
  *Replaces the extension in buf of max.*

- `int fl_open_uri (const char *, char *, int)`
  
  *Opens the specified Uniform Resource Identifier (URI).*

### 34.3.1 Detailed Description

File names and URI utility functions.

### 34.4 filename.H

Go to the documentation of this file.
```c
#ifndef FL_FILENAME_H
#define FL_FILENAME_H

#include "Fl_Export.H"
#include <FL/platform_types.h>

#define FL_PATH_MAX 2048

FL_EXPORT const char *fl_filename_name(const char *filename);
FL_EXPORT const char *fl_filename_ext(const char *buf);
FL_EXPORT char *fl_filename_setext(char *to, int tolen, const char *ext);
FL_EXPORT int fl_filename_expand(char *to, int tolen, const char *from);
FL_EXPORT int fl_filename_absolute(char *to, int tolen, const char *from);
FL_EXPORT int fl_filename_relative(char *to, int tolen, const char *from, const char *cwd);
FL_EXPORT int fl_filename_match(const char *name, const char *pattern);
FL_EXPORT int fl_filename_isdir(const char *name);

#if defined(__cplusplus) && !defined(FL_DOXYGEN)
/*
Under Windows, we include filename.H from numericsort.c; this should probably change...
*/
#endif /* __cplusplus */

#if defined (__cplusplus)
extern "C" {
#endif /* __cplusplus */

#if !defined(FL_DOXYGEN)
FL_EXPORT int fl_alphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casealphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casenumericsort(struct dirent **, struct dirent **);
FL_EXPORT int fl_numericsort(struct dirent **, struct dirent **);
#endif

typedef int (Fl_File_Sort_F)(struct dirent **, struct dirent **);

#if defined(__cplusplus)
}
#endif /* __cplusplus */

#if defined (__cplusplus)
extern "C" {
#endif /* __cplusplus */

FL_EXPORT int fl_filename_list(const char *d, struct dirent ***l, Fl_File_Sort_F *s = fl_numericsort);
FL_EXPORT void fl_filename_free_list(struct dirent ***l, int n);

/* Portable "scandir" function. Ugly but necessary... */

#if defined(__cplusplus)
}
#endif /* __cplusplus */

/* FLTK 1.0.x compatibility definitions (FLTK_1_0_COMPAT) dropped in 1.4.0 */
#endif /* FL_FILENAME_H */
```

34.5 FL.H File Reference

Fl static class.
#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_Acloc_close_Handler) (Fl_Window *window, void *data)**
  Signature of set_atclose functions passed as parameters.

- **typedef void( Fl_Awake_Handler) (void *data)**
  Signature of some wakeup callback functions passed as parameters.

- **typedef void( Fl_Box_Draw_F) (int x, int y, int w, int h, Fl_Color color)**
  Signature of some box drawing functions passed as parameters.

- **typedef void( Fl_Clipboard_Notify_Handler) (int source, void *data)**
  Signature of add_clipboard_notify functions passed as parameters.

- **typedef int( Fl_Event_Dispatch) (int event, Fl_Window *w)**
  Signature of event_dispatch functions passed as parameters.

- **typedef int( Fl_Event_Handler) (int event)**
  Signature of add_handler functions passed as parameters.

- **typedef void( Fl_FD_Handler) (FL_SOCKET fd, void *data)**
  Signature of add_fd functions passed as parameters.

- **typedef void( Fl_Idle_Handler) (void *data)**
  Signature of add_idle callback functions passed as parameters.

- **typedef void( Fl_Label_Draw_F) (const Fl_Label *label, int x, int y, int w, int h, Fl_Align align)**
  Signature of some label drawing functions passed as parameters.

- **typedef void( Fl_Label_Measure_F) (const Fl_Label *label, int &width, int &height)**
  Signature of some label measurement functions passed as parameters.

- **typedef void( Fl_Old_Idle_Handler) ()**
  Signature of set_idle callback functions passed as parameters.

- **typedef int( Fl_System_Handler) (void *event, void *data)**
  Signature of add_system_handler functions passed as parameters.

- **typedef void( Fl_Timeout_Handler) (void *data)**
  Signature of timeout callback functions passed as parameters.
Variables

- `const char * fl_local_alt`
  
  String pointer used in shortcuts, you can change it to another language

- `const char * fl_local_ctrl`
  
  String pointer used in shortcuts, you can change it to another language

- `const char * fl_local_meta`
  
  String pointer used in shortcuts, you can change it to another language

- `const char * fl_local_shift`
  
  String pointer used in shortcuts, you can change it to another language

34.5.1 Detailed Description

**Fl** static class.

34.6 **Fl.H**

Go to the documentation of this file.

```cpp
1 #ifndef Fl_H
2 # define Fl_H
3 #include <FL/fl_config.h> // build configuration
4 #include <FL/Fl_Export.H>
5 #include <FL/platform_types.h> // for FL_SOCKET
6 #include <FL/fl_casts.H> // experimental
7 #ifdef FLTK_HAVE_CAIRO
8 # include <FL/Fl_Cairo.H>
9 #endif
10 # include "fl_utf8.h"
11 # include "Enumerations.H"
12 #ifdef Fl_Object
13 # define Fl_Object Fl_Widget
14 #endif
15
16 #ifdef check
17 # undef check
18 #endif
19 #ifdef BSD
20 # undef BSD
21 #endif
22
23 #include <string.h> // FIXME: Fl::is_scheme(): strcmp needs string.h
24
25 class Fl_Widget;
26 class Fl_Window;
27 class Fl_Image;
28 struct Fl_Label;
29 class Fl_Screen_Driver;
30 class Fl_System_Driver;
31
32 // Pointers you can use to change FLTK to another language.
33 // Note: Similar pointers are defined in Fl/fl_ask.H and src/fl_ask.cxx
34 extern FL_EXPORT const char* fl_local_alt;
35 extern FL_EXPORT const char* fl_local_ctrl;
36 extern FL_EXPORT const char* fl_local_meta;
37 extern FL_EXPORT const char* fl_local_shift;
```
typedef void (Fl_Label_Draw_F)(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align);

typedef void (Fl_Label_Measure_F)(const Fl_Label *label, int &width, int &height);

typedef void (Fl_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color);

typedef void (*Fl_Timeout_Handler)(void *data);

typedef void (*Fl_Awake_Handler)(void *data);

typedef void (*Fl_Idle_Handler)(void *data);

typedef void (*Fl_Old_Idle_Handler)();

typedef void (*Fl_FD_Handler)(FL_SOCKET fd, void *data);

typedef int (*Fl_Event_Handler)(int event);

typedef int (*Fl_System_Handler)(void *event, void *data);

typedef void (*Fl_Abort_Handler)(const char *format,...);

typedef void (*Fl_Atclose_Handler)(Fl_Window *window, void *data);

typedef int (*Fl_Args_Handler)(int argc, char **argv, int &i);

typedef int (*Fl_Event_Dispatch)(int event, Fl_Window *w);

typedef void (*Fl_Clipboard_Notify_Handler)(int source, void *data);
/* group callback_functions */

class FL_EXPORT Fl {  // should be private!
  friend class Fl_System_Driver;
  Fl() {} // no constructor!
  private:
  static int use_high_res.GL_;  // only for compatibility with 1.3
  static int draw.GL_text_with_textures_;  // should be private!
  static int box_shadow_width_;  // only for compatibility with 1.3
  static int box_border_radius_max_;  // should be private!
  public:
  static Fl_Screen_Driver *screen_driver();
  static Fl_System_Driver *system_driver();
  static int box_shadow_width() { return box_shadow_width_; }
  static void box_shadow_width(int W) { box_shadow_width_ = W < 1 ? 1 : W; }
  static int box_border_radius_max() { return box_border_radius_max_; }
  static void box_border_radius_max(int R) { box_border_radius_max_ = R < 5 ? 5 : R; }
  #endif
  #ifdef __APPLE__ // deprecated in 1.4 - only for compatibility with 1.3
  static void reset_marked_text();
  static void insertion_point_location(int x, int y, int height);
  #endif
  #ifndef FL_DOXYGEN
  static int e_number;
  static int e_x;
  static int e_y;
  static int e_x_root;
  static int e_y_root;
  static int e_dx;
  static int e_dy;
  static int e_state;
  static int e_clicks;
  static int e_is_click;
  static int e_keysym;
  static char *e_text;
  static int e_length;
  static void *e_clipboard_data;
  static const char *e_clipboard_type;
  static Fl_Event_Dispatch e_dispatch;
  static Fl_Callback_Reason callback_reason_;  // only for compatibility with 1.3
  static Fl_Widget *belowmouse_;  // should be private!
  static Fl_Widget *pushed_;  // should be private!
  static Fl_Widget *focus_;  // should be private!
  static int damage_;  // should be private!
  static Fl_Widget *selection_owner_;  // should be private!
  static Fl_Window *modal_;  // should be private!
  #ifdef __APPLE__
  static void reset_marked_text();
  static void insertion_point_location(int x, int y, int height);
  #endif
  #endif
};
213  static Fl_Window* grab_;
214  static int compose_state; // used for dead keys (Windows) or marked text (MacOS)
215  static void call_screen_init(); // recompute screen number and dimensions
216  #endif // FL_DOXYGEN
217
218
222  static void damage(int d) {damage_ = d;}
223
224  public:
231  typedef enum {
241    OPTION_ARROW_FOCUS = 0,
242    // When switched on, FLTK will use the file chooser dialog that comes
243    // with your operating system whenever possible. When switched off, FLTK
244    // will present its own file chooser.
245    // "todo implement me"
246    OPTION_NATIVE_FILECHOOSER,
247    // When Filechooser Preview is enabled, the FLTK or native file chooser
248    // will show a preview of a selected file (if possible) before the user
249    // decides to choose the file.
250    // "todo implement me"
251    OPTION_FILECHOOSER_PREVIEW,
256    OPTION_VISIBLE_FOCUS,
260    OPTION_DND_TEXT,
264    OPTION_SHOW_TOOLTIPS,
268    OPTION_FNFC_USES_GTK,
272    OPTION_PRINTER_USES_GTK,
276    OPTION_SHOW_SCALING,
279    OPTION_FNFC_USES_ZENITY,
280    // don’t change this, leave it always as the last element
282    OPTION_LAST
283  } Fl_Option;
284
285  private:
286  static unsigned char options_[OPTION_LAST];
287  static unsigned char options_read_;
288  static int program_should_quit_; // non-zero means the program was asked to cleanly terminate
289
290  public:
291  /*
292   Return a global setting for all FLTK applications, possibly overridden
293   by a setting specifically for this application.
294   */
295  static bool option(Fl_Option opt);
296
297  /*
298   Override an option while the application is running.
299   */
300  static void option(Fl_Option opt, bool val);
301
302  #ifndef FL_DOXYGEN
303  private:
304  static Fl_Awake_Handler *awake_ring_;
305  static void **awake_data_;
306  static int awake_ring_size_; 
307  static int awake_ring_head_; 
308  public:
309  static const char * scheme_; 
310  static Fl_Image * scheme_bg_; 
311  static int e_original_keysym; // late addition
312  static int scrollbar_size_; 
313  static int menu_linespacing_; // STR #2927
314  #endif
315
316  static int add_awake_handler_(Fl_Awake_Handler, void *);
317  static int get_awake_handler_(Fl_Awake_Handler&, void *&);
318
319  public:
320  // API version number
321  static double version();
322  static int api_version();
323  static int abi_version();
324  static int abi_check(const int val = FL_ABI_VERSION) {
325    return val == abi_version();
326  }
327
328  // argument parsers:
329  static int arg(int argc, char **argv, int& i);
330  static int args(int argc, char **argv, int& i, Fl_Args_Handler cb = 0);
static void argv(int argc, char **argv);
static const char * const help;

// things called by initialization:
static void display(const char *);
static int visual(int);
static int gl_visual(int, int *alist=0); // platform dependent
static void own_colormap();
static void foreground(uchar, uchar, uchar);
static void background(uchar, uchar, uchar);
static void background2(uchar, uchar, uchar);

// schemes:
static int scheme(const char *name);
static const char * scheme() {return scheme_;
}
static int is_scheme(const char *name) {
    return (scheme_ && name && !strcmp(name, scheme_));
}
static int reload_scheme(); // defined in 'src/Fl_get_system_colors.cxx'

static int scrollbar_size();
static void scrollbar_size(int W);
static int menu_linespacing();
static void menu_linespacing(int H);

// execution:
static int wait();
static double wait(double time);
static int check();
static int ready();
static int run();
static int program_should_quit() {return program_should_quit_;
}
static void program_should_quit(int should_i) { program_should_quit_ = should_i;
}
static void hide_all_windows();

static Fl_Widget * readqueue();

// cross-platform timer support
static void add_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
static void repeat_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
static int has_timeout(Fl_Timeout_Handler cb, void *data = 0);
static void remove_timeout(Fl_Timeout_Handler cb, void *data = 0);
static void add_check(Fl_Timeout_Handler, void * = 0);
static int has_check(Fl_Timeout_Handler, void * = 0);
static void remove_check(Fl_Timeout_Handler, void * = 0);

static Fl_Timestamp now();
static double seconds_since(Fl_Timestamp& then);
static double seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back);
static long ticks_since(Fl_Timestamp& then);
static long ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back);

// private
static void run_idle();
static void run_checks();
static void add_fd(int fd, int when, Fl_FD_Handler cb, void * = 0); // platform dependent
static void add_fd(int fd, Fl_FD_Handler cb, void * = 0); // platform dependent
static void remove_fd(int, int when); // platform dependent
static void remove_fd(int); // platform dependent
static void add_idle(Fl_Idle_Handler cb, void * data = 0);
static int has_idle(Fl_Idle_Handler cb, void * data = 0);
static void remove_idle(Fl_Idle_Handler cb, void * data = 0);
static void *warning(const char*, ...);
static void *error(const char*, ...);
static void *fatal(const char*, ...);
static Fl_Window * first_window();
static void first_window(Fl_Window *);
static Fl_Window * next_window(const Fl_Window *);
static Fl_Window * modal() {return modal_;
}
static Fl_Window * grab() {return grab_;
}
static void grab(Fl_Window *); // platform dependent
static int event() {return e_number;
}
static int event_x() {return e_x;
}
static int event_y() {return e_y;
}
static int event_x_root() {return e_x_root;}
static int event_y_root() {return e_y_root;}
static int event_dx() {return e_dx;}
static int event_dy() {return e_dy;}
static void get_mouse(int &, int &);
static int event_clicks() {return e_clicks;}
static void event_clicks(int i) {e_clicks = i;}
static int event_is_click(int i) {e_is_click = i;}
static int event_button() {return e_keysym_FL_Button;}
static int event_state() {return e_state;}
static int event_state(int mask) {return e_state & mask;}
static int event_key() {return e_keysym;}
static int event_original_key(){return e_original_keysym;}
static void *event_clipboard() { return e_clipboard_data; }
static const char *event_clipboard_type() {return e_clipboard_type; }
static int compose(int &del);
static void compose_reset();
static int event_inside(int, int, int, int);
static int event_inside(const Fl_Widget *);
static int test_shortcut(Fl_Shortcut);
static void enable_im();
static void disable_im();
static Fl_Widget * belowmouse() {return belowmouse_;
static void belowmouse(Fl_Widget *);
static Fl_Widget * pushed() {return pushed_;
static void pushed(Fl_Widget *);
static Fl_Widget * focus() {return focus_;
static void add_handler(Fl_Event_Handler h);
static void remove_handler(Fl_Event_Handler h);
static void add_system_handler(Fl_System_Handler h, void *data);
static void remove_system_handler(Fl_System_Handler h);
static void event_dispatch(Fl_Event_Dispatch d);
static Fl_Callback_Reason callback_reason();
static void paste(Fl_Widget &receiver, int source, const char *type = Fl::clipboard_plain_text);
static void add_clipboard_notify(Fl_Clipboard_Notify_Handler h, void *data = 0);
static void remove_clipboard_notify(Fl_Clipboard_Notify_Handler h);
static int clipboard_contains(const char *type);
static char const * const clipboard_plain_text;
static char const * const clipboard_image;
static int dnd(); // platform dependent
static Fl_Widget * selection_owner() {return selection_owner_;}
static void selection(Fl_Widget &owner, const char *, int len);
static void paste(Fl_Widget &receiver);
static int x(); // via screen driver
static int y(); // via screen driver
static int w(); // via screen driver
static int h(); // via screen driver

// multi-head support:
static int screen_count(); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int n); // via screen driver
static void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my, int nx, int ny, int mh); // via screen driver
static int screen_num(int x, int y); // via screen driver

// event destinations:
static int handle(int, Fl_Window *);
static int handle(int, Fl_Window *);
static FL.Widget* belomouse() { return belomouse_;}
static FL.Widget* belomouse(FL_Widget *);
static FL.Widget* pushed() { return pushed_;}
static FL.Widget* focus() { return focus_;
static void add_handler(FL_Event_Handler h);
static void remove_handler(FL_Event_Handler h);
static void add_system_handler(FL_System_Handler h, void *data);
static void remove_system_handler(FL_System_Handler h);
static void event_dispatch(FL_Event_Dispatch d);
static FL_Callback_Reason callback_reason();

// cut/paste:
static void copy(const char *stuff, int len, int destination = 0, const char *type = Fl::clipboard_plain_text);
static void selection_to_clipboard(int mode) {
selection_to_clipboard_ = mode ? 1 : 0;
}

// These are for back-compatibility only:
static FL.Widget* selection_owner() {return selection_owner_;}
static void selection_owner(FL_Widget*);
static void selection(FL_Widget &owner, const char *, int len);
static void paste(FL_Widget &receiver);
static int x(); // via screen driver
static int y(); // via screen driver
static int w(); // via screen driver
static int h(); // via screen driver

// multi-head support:
static int screen_count(); // via screen driver
static void screen_xywh(int X, int Y, int W, int H); // via screen driver
static void screen_xywh(int X, int Y, int W, int H, int mx, int my); // via screen driver
static void screen_xywh(int X, int Y, int W, int H, int n); // via screen driver
static void screen_xywh(int X, int Y, int W, int H, int mx, int my, int nx, int ny, int mh); // via screen driver
static int screen_num(int x, int y); // via screen driver

Generated by Doxygen
static int screen_num(int x, int y, int w, int h); // via screen driver
static void screen_dpi(float &h, float &v, int n=0); // via screen driver
static void screen_work_area(int &X, int &Y, int &W, int &H, int mx, int my); // via screen driver
static void screen_work_area(int &X, int &Y, int &W, int &H, int n); // via screen driver
static void screen_work_area(int &X, int &Y, int &W, int &H); // via screen driver
static float screen_scale(int n); // via screen driver
static void screen_scale(int n, float factor); // via screen driver
static int screen_scaling_supported();
static void keyboard_screenScaling(int value);

// color map:
static void set_color(Fl_Color, uchar, uchar, uchar);
static void set_color(Fl_Color, uchar, uchar, uchar, uchar);
static void set_color(Fl_Color i, unsigned c); // platform dependent
static unsigned get_color(Fl_Color i);
static void get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue);
static void free_color(Fl_Color i, int overlay = 0); // platform dependent

// fonts:
static const char * get_font(Fl_Font);
static const char * get_font_name(Fl_Font, int* attributes = 0);
static int get_font_sizes(Fl_Font, int* & sizep);
static void set_font(Fl_Font, const char *);
static void set_font(Fl_Font, Fl_Font);
static Fl_Font set_fonts(const char * = 0); // platform dependent

// <Hack to re-order the 'Drawing functions' group>

// labetypes:
static void set_labeltype(Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *);
static void set_labeltype(Fl_Labeltype, Fl_Labeltype from); // is it defined ?

// boxtypes:
static Fl_Box_Draw_F *get_boxtype(Fl_Boxtype);
static void set_boxtype(Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar);
static void set_boxtype(Fl_Boxtype, Fl_Boxtype from);
static int box_dx(Fl_Boxtype);
static int box_dy(Fl_Boxtype);
static int box_dw(Fl_Boxtype);
static int box_dh(Fl_Boxtype);
static void draw_box_active();
static Fl_Color box_color(Fl_Color);
static void set_box_color(Fl_Color);

// back compatibility:
static void set_abort(Fl_Abort_Handler f) {fatal = f;}
static void default_atclose(Fl_Window*, void*);
static void set_atclose(Fl_Atclose_Handler f) {atclose = f;}
static int event_shift() {return e_state&FL_SHIFT;}
static int event_ctrl() {return e_state&FL_CTRL;}
static int event_command() {return e_state&FL_COMMAND;}
static int event_alt() {return e_state&FL_ALT;}
static int event_button1() {return e_state&FL_BUTTON1;}
static int event_button2() {return e_state&FL_BUTTON2;}
static int event_button3() {return e_state&FL_BUTTON3;}
static int event_buttons() {return e_state&0x7f000000;}
static void set_idle(Fl_Old_Idle_Handler cb) {idle = cb;}
static void grab(Fl_Window& win) {grab(&win);}
static void release() {grab(0);}

// Visible focus methods...
static void visible_focus(int v) { option(OPTION_VISIBLE_FOCUS, (v!=0)); }
static int visible_focus() { return option(OPTION_VISIBLE_FOCUS); }

// Drag-n-drop text operation methods...
static void dnd_text_ops(int v) { option(OPTION_DND_TEXT, (v!=0)); }
static int dnd_text_ops() { return option(OPTION_DND_TEXT); }

// Multithreading support:
static int lock();
static void unlock();
static void awake(void * message = 0);
static int awake(Fl_Awake_Handler cb, void * message = 0);
static void * thread_message(); // platform dependent

// Widget deletion:
static void delete_widget(Fl_Widget *w);
static void do_widget_deletion();
static void watch_widget_pointer(Fl_Widget *w);
static void release_widget_pointer(Fl_Widget *w);
static void use_high_res_GL(int val) { use_high_res_GL_ = val; }
static int use_high_res_GL() { return use_high_res_GL_; }
static void use_high_res_GL(GLint val) { use_high_res_GL_ = val; }
static int draw_GL_text_with_textures() { return draw_GL_text_with_textures_; }

Generated by Doxygen
static int system(const char *command);

#ifdef FLTK_HAVE_CAIRO
public:
    // Cairo support API

    static cairo_t *cairo_make_current(Fl_Window *w);
    static void cairo_autolink_context(bool alink) {
        cairo_state_.autolink(alink);
    }
    static bool cairo_autolink_context() {
        return cairo_state_.autolink();
    }
    static cairo_t *cairo_make_current(cairo_t *c, bool own=false) {
        cairo_state_.cc(c, own);
    }

    static void cairo_flush(cairo_t *c) {
        // flush Cairo drawings: necessary at least for Windows
        cairo_surface_t *s = cairo_get_target(Fl::cairo_cc());
        cairo_surface_flush(s);
    }

    private:
    static cairo_t *cairo_make_current(void *gc);
    static cairo_t *cairo_make_current(void *gc, int W, int H);
    static Fl_Cairo_State cairo_state_;

#endif // FLTK_HAVE_CAIRO

public:
    Fl_Widget_Tracker(Fl_Widget *wi);
    ~Fl_Widget_Tracker();

    Fl_Widget *widget() {return wp_;
    int deleted() {return wp_ == 0;
    int exists() {return wp_ != 0;

34.7 Fl_Adjuster.H

1 //
2 // Adjuster widget header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16
17 /* file
18 Fl_Adjuster widget. */
19
20 // 3-button "slider", made for Nuke
21
22 #ifndef Fl_Adjuster_H
23 #define Fl_Adjuster_H

Generated by Doxygen
```cpp
#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Adjuster : public Fl_Valuator {
  int drag;
  int ix;
  int soft_;  
protected:
  void draw() FL_OVERRIDE;
  int handle(int) FL_OVERRIDE;
  void value_damage() FL_OVERRIDE;
public:
  Fl_Adjuster(int X,int Y,int W,int H, const char *l=0);
  void soft(int s) {soft_ = s;}
  int soft() const {return soft_ ;}
};
#endif

#ifndef Fl_Anim_Gif_Image_H
#define Fl_Anim_Gif_Image_H

// forward declarations
class Fl_Image;
class Fl_Widget;

#include <FL/Fl_GIF_Image.H>

// Load and display animator GIF images
class FL_EXPORT Fl_Anim_GIF_Image : public Fl_GIF_Image {
private:
  enum Flags {
    DONT_START = 1,
    DONT_RESIZE_CANVAS = 2,
    DONT_SET_AS_IMAGE = 4,
    OPTIMIZE_MEMORY = 8,
    LOG_FLAG = 64,
    DEBUG_FLAG = 128
  };

  // -- constructors and destructor
  Fl_Anim_GIF_Image(const char *filename, Fl_Widget *canvas = 0, unsigned short flags = 0);
  Fl_Anim_GIF_Image(const char * imagename, const unsigned char *data,
                    const size_t length, Fl_Widget *canvas = 0,
                    unsigned short flags = 0);
  Fl_Anim_GIF_Image();
  ~Fl_Anim_GIF_Image() FL_OVERRIDE;

  // -- file handling
  bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0);
  bool valid() const;

  // -- getters and setters
  bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0);
  bool valid() const;

  // -- file handling
  bool load(const char *filename, Fl_Widget *canvas = 0, unsigned short flags = 0);
  Fl_Anim_GIF_Image();
  ~Fl_Anim_GIF_Image() FL_OVERRIDE;

  // -- file handling
  bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0);
  bool valid() const;

  // -- file handling
  void frame_uncache(bool uncache);
  bool frame_uncache() const;
  double delay(int frame) const;
  void delay(int frame, double delay);
  void canvas(Fl_Widget *canvas, unsigned short flags = 0);
}
#endif
```
bool is_animated() const;
const char *name() const;
void speed(double speed);
double speed() const;

// -- animation
int frames() const;
void frame(int frame);
int frame() const;
Fl_Image *image() const;
Fl_Image *image(int frame) const;
bool start();
bool stop();
bool next();

bool playing() const { return valid() && Fl::has_timeout(cb_animate, (void *)this); }

// -- image data
Fl_Anim_GIF_Image & resize(int w, int h);
Fl_Anim_GIF_Image & resize(double scale);
int frame_x(int frame) const;
int frame_y(int frame) const;
int frame_w(int frame) const;
int frame_h(int frame) const;

// -- overridden methods
void color_average(Fl_Color c, float i) FL_OVERRIDE;
Fl_Image *copy(int W, int H) const FL_OVERRIDE;
Fl_Image *copy() const { return Fl_Pixmap::copy(); }
desaturate() FL_OVERRIDE;
draw(int x, int y, int w, int h, int cx = 0, int cy = 0) FL_OVERRIDE;
uncache() FL_OVERRIDE;

// -- debugging and logging
toルドbug() const;

// -- static methods
static int frame_count(const char *name, const unsigned char *imgdata = NULL, size_t imglength = 0);
static bool loop;
static double min_delay;

protected:
bool next_frame();
void clear_frames();
void set_frame(int frame);
static void cb_animate(void *d);
scale_frame();
static cb_animate(void *d);
void on_frame_data(Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE;
void on_extension_data(Fl_GIF_Image::GIF_FRAME &f) FL_OVERRIDE;

private:
char *name _;
unsigned short flags _;
Fl_Widget *canvas _;
bool uncache _;
int frame _; // current frame
double speed _;
FrameInfo *fi _;

};

#endif // Fl_Anim_Gif_Image_H

34.9 fl_ask.H File Reference

API for common dialogs.
#include <FL/Enumerations.H>
#include <FL/Fl_String.H>
#include <FL/fl_attr.h>

Generated by Doxygen
Enumerations

- enum Fl_Beep {
  FL_BEEP_DEFAULT = 0, FL_BEEP_MESSAGE, FL_BEEP_ERROR, FL_BEEP_QUESTION,
  FL_BEEP_PASSWORD, FL_BEEP_NOTIFICATION
}

Defines the different system beeps available.

Functions

- void void fl_alert (const char *,...) __fl_attr((__format__::__printf__)
- void void fl_ask (const char *,...) __fl_attr((__format__::__printf__)
- void fl_beep (int type=FL_BEEP_DEFAULT)
  Emits a system beep.
- int const char const char int fl_choice (const char *, const char *, const char *, const char *,...) __fl_attr((__format__( __printf__)
- int const char const char int fl_choice_n (const char *, const char *, const char *, const char *,...) __fl_attr((__format__(__printf__
- int const char + fl_input (const char *label, const char *deflt=0,...) __fl_attr(__(format__(__printf__
- int const char + Fl_String Fl_String fl_input_str (int &ret, int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- int const char + Fl_String Fl_String fl_input_str (int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- void fl_message (const char *,...) __fl_attr((__format__(__printf__
- void fl_message_font (Fl_Font f, Fl_Fontsize s)
- void fl_message_hotspot (int enable)
  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- void fl_message_hotspot (void)
  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- int const char + Fl_String Fl_String Fl_String Fl_String Fl_Widget * fl_message_icon ()
  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char *str)
  Sets the icon label of the dialog window used in many common dialogs.
- void fl_message_position (const int x, const int y, const int center=0)
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_position (Fl_Widget &widget)
- void fl_message_position (Fl_Widget *widget)
  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- int fl_message_position (int *x=0, int *y=0)
  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_title (const char *title)
  Sets the title of the dialog window used in many common dialogs.
- void fl_message_title_default (const char *title)
  Sets the default title of the dialog window used in many common dialogs.
- int const char const char + fl_password (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- int const char const char int Fl_String Fl_String Fl_String Fl_String Fl_String fl_password_str (int &ret, int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- int const char const char int Fl_String Fl_String Fl_String Fl_String fl_password_str (int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__

Generated by Doxygen
Variables

- void __deprecated__
- const char * fl_cancel
  - string pointer used in common dialogs, you can change it to another language
- const char * fl_close
  - string pointer used in common dialogs, you can change it to another language
- Fl_Font fl_message_font_
- Fl_Fontsize fl_message_size_
- const char * fl_no
  - string pointer used in common dialogs, you can change it to another language
- const char * fl_ok
  - string pointer used in common dialogs, you can change it to another language
- const char * fl_yes
  - string pointer used in common dialogs, you can change it to another language

34.9.1 Detailed Description

API for common dialogs.

34.9.2 Enumeration Type Documentation

34.9.2.1 Fl_Beep

enum Fl_Beep

Defines the different system beeps available.

Some systems may play different sounds or use different sound volume depending on the Fl_Beep value. The implementation is platform dependent.

See also

fl_beep(int)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_BEEP_DEFAULT</td>
<td>Default beep.</td>
</tr>
<tr>
<td>FL_BEEP_MESSAGE</td>
<td>Message beep.</td>
</tr>
<tr>
<td>FL_BEEP_ERROR</td>
<td>Error beep.</td>
</tr>
<tr>
<td>FL_BEEP_QUESTION</td>
<td>Question beep.</td>
</tr>
<tr>
<td>FL_BEEP_PASSWORD</td>
<td>Password beep.</td>
</tr>
<tr>
<td>FL_BEEP_NOTIFICATION</td>
<td>Notification beep.</td>
</tr>
</tbody>
</table>

34.9.3 Function Documentation

34.9.3.1 fl_message_position()

void fl_message_position ( Fl_Widget & widget ) [inline]
See also

```
fl_message_position(Fl_Widget *widget).
```

## 34.10  fl_ask.H

Go to the documentation of this file.

```
// Standard dialog header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php
```

```c
#ifndef _FL_fl_ask_H_
#define _FL_fl_ask_H_

#include <FL/Enumerations.H>
#include <FL/Fl_String.H>
#include <FL/fl_attr.h>

class Fl_Widget;

enum Fl_Beep {
    FL_BEEP_DEFAULT = 0,
    FL_BEEP_MESSAGE,
    FL_BEEP_ERROR,
    FL_BEEP_QUESTION,
    FL_BEEP_PASSWORD,
    FL_BEEP_NOTIFICATION
};

FL_EXPORT void fl_beep(int type = FL_BEEP_DEFAULT);
FL_EXPORT void fl_message(const char *, ...) __fl_attr((__format__(__printf__, 1, 2)));
FL_EXPORT void fl_alert(const char *, ...) __fl_attr((__format__(__printf__, 1, 2)));

// fl_ask() is deprecated since it uses "Yes" and "No" for the buttons,
// which does not conform to the current FLTK Human Interface Guidelines.
// Use fl_choice() or fl_choice_n() with the appropriate verbs instead.

FL_EXPORT int fl_ask(const char *, ...) __fl_attr((__format__(__printf__, 1, 2), __deprecated__));

FL_EXPORT int fl_choice(const char *q, const char *b0, const char *b1, const char *b2, ...)
    __fl_attr((__format__(__printf__, 1, 5)));
FL_EXPORT const char *fl_input(const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 1, 3)));
FL_EXPORT const char *fl_password(const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 1, 3)));

// since FLTK 1.3.8:
// - fl_choice_n() with extended return value (-2, -1, 0, 1, 2)
FL EXPORT int fl_choice_n(const char *q, const char *b0, const char *b1, const char *b2, ...)
    __fl_attr((__format__(__printf__, 1, 5)));

FL EXPORT int fl_choice_n(const char *q, const char *b0, const char *b1, const char *b2, ...)
    __fl_attr((__format__(__printf__, 1, 5)));

// since FLTK 1.4.0:
FL_EXPORT Fl_Widget *fl_message_icon();
extern FL_EXPORT Fl_Font fl_message_font_;
extern FL_EXPORT Fl_Fontsize fl_message_size_;
inline void fl_message_font(Fl_Font f, Fl_Fontsize s) {
```
# fl attr.h File Reference

This file defines compiler-specific macros.

## Macros

- `#define __fl_attr(x)`
  This section lists macros for Doxygen documentation only.

- `#define FL_DEPRECATED(msg, func)`
  Enclosing a function or method in FL_DEPRECATED marks it as no longer recommended.

- `#define FL_OVERRIDE override`
  This macro makes it safe to use the C++11 keyword `override` with older compilers.

## Detailed Description

This file defines compiler-specific macros.

## Macro Definition Documentation

### 34.11.2.1 __fl_attr

`#define __fl_attr(x)`

This section lists macros for Doxygen documentation only.

The next section will define the actual macros based on the compile used and based on the capabilities of the version of that compiler. To be used in prototypes with a variable list of arguments. This macro helps detection of mismatches between format string and argument list at compilation time.

Usage example: `FL/fl_ask.H`

### 34.11.2.2 FL_DEPRECATED

`#define FL_DEPRECATED(msg, func)`

Generated by Doxygen
Value:

```c
/**\#\#\#\# deprecated msg \#\#\# */
func
```

Enclosing a function or method in FL_DEPRECATED marks it as no longer recommended. This macro syntax cannot be used if the return type contains a comma, which is not the case in FLTK.

```c
FL_DEPRECATED("Outdated, don’t use", int position()) { return position_; }
```

### 34.12 fl_attr.h

Go to the documentation of this file.

```c
1 /*
2 * Function attribute declarations for the Fast Light Tool Kit (FLTK).
3 * 
4 * Copyright 1998-2023 by Bill Spitzak and others.
5 * 
6 * This library is free software. Distribution and use rights are outlined in
7 * the file “COPYING” which should have been included with this file. If this
8 * file is missing or damaged, see the license at:
9 * 
10 * https://www.fltk.org/COPYING.php
11 * 
12 * Please see the following page on how to report bugs and issues:
13 * 
14 * https://www.fltk.org/bugs.php
15 */
16
17 ifndef _FL_fl_attr_h_
18 #define _FL_fl_attr_h_
19
20 #ifdef __cplusplus
21
22 #if defined (_MSC_VER)
23 
24 #if (_MSC_VER >= 1400) // Visual Studio .NET 2003 (7.1)
25 
26 #define FL_DEPRECATED(msg, func)
27 /##*##* \
deprecated msg *##/ \
28 func
29
30 #elif _MSC_VER >= 1310 // Visual Studio .NET 2003 (7.1)
31 
32 #ifndef __cplusplus
33 
34 #ifndef __cplusplus
35 
36 #define __fl_attr(x)
37 
38 #endif /* __cplusplus */
39 
40 #define __fl_attr(x)
41 
42 #define __fl_attr(x)
43 
44 #define FL_OVERRIDE override
45 
46 #define FL_OVERRIDE override
47 
48 #endif /* __fl_attr(x) */
49 
50 #endif /* FL_OVERRIDE */
51 
52 #endif /* FL_DEPRECATED */
53 
54 #endif /* __cplusplus */
55 
56 #endif /* _FL_fl_attr_h_ */
57 
58 #endif /* _FL_fl_attr_h_ */
59 
60 #endif /* _FL_fl_attr_h_ */
61 
62#endif /* _FL_fl_attr_h_ */
63 
64#endif /* __cplusplus */
65 
66#endif /* _FL_fl_attr_h_ */
67 
68# endif /* _FL_fl_attr_h_ */
69 
70# endif /* _FL_fl_attr_h_ */
71 
72# endif /* _FL_fl_attr_h_ */
73 
74# endif /* _FL_fl_attr_h_ */
75 
76# endif /* _FL_fl_attr_h_ */
77 
78# endif /* _FL_fl_attr_h_ */
79 
80# endif /* _FL_fl_attr_h_ */
81 
82# endif /* _FL_fl_attr_h_ */
83 
84# endif /* _FL_fl_attr_h_ */
85 
86# endif /* _FL_fl_attr_h_ */
87 
88# endif /* _FL_fl_attr_h_ */
89 
90# endif /* _FL_fl_attr_h_ */
91 
92# endif /* _FL_fl_attr_h_ */
93 
94# endif /* _FL_fl_attr_h_ */
95 
96# endif /* _FL_fl_attr_h_ */
97 
98# endif /* _FL_fl_attr_h_ */
99 
100#endif /* _FL_fl_attr_h_ */
101 
102#endif /* _FL_fl_attr_h_ */
103
```

Generated by Doxygen
Declare macros specific to the C++ standard used.

Macros may have been declared already in previous sections.

Declare macros specific to clang

Macros may have been declared already in previous sections.

Declare macros specific to gcc.
34.13 Fl_Bitmap.H

1 //
2 // Bitmap header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2017 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /*
18 file
19 */
20
21 #ifndef Fl_Bitmap_H
22 #define Fl_Bitmap_H
23 #include "Fl_Image.H"
24 #include "Fl_Widget.H" // for fl_uintptr_t
25
26 class Fl_Widget;
27 struct Fl_Menu_Item;
28
29 class FL_EXPORT Fl_Bitmap : public Fl_Image {
30 friend class Fl_Graphics_Driver;
31 public:
32 const uchar *array;
33 int alloc_array;
34
35 private:
36 fl_uintptr_t id_; //栲 critical array cache size
37 int cache_w_, cache_h_; // size of bitmap when cached
38
39 public:
40 Fl_Bitmap(const uchar *bits, int W, int H) :
41 Fl_Image(W,H,0), array(bits), alloc_array(0), id_(0), cache_w_(0),cache_h_(0) {data((const char **)&array, 1);
42 }
43 Fl_Bitmap(const char *bits, int W, int H) :
44 Fl_Image(W,H,0), array((const uchar *)bits), alloc_array(0), id_(0), cache_w_(0),cache_h_(0)
45 {data((const char **)&array, 1);
46 }
47 Fl_Bitmap(const uchar *bits, int bits_length, int W, int H);
48 Fl_Bitmap(const char *bits, int bits_length, int W, int H);
49 virtual ~Fl_Bitmap();
50 Fl_Image *copy(const uchar *bits, int W, int H) :
51 Fl_Image(W,H,0), array(bits), alloc_array(0), id_(0), cache_w_(0),cache_h_(0) [data((const char **)&array, 1);
52 }
53 Fl_Bitmap(const char *bits, int W, int H) :
54 Fl_Bitmap(const uchar *bits, int bits_length, int W, int H);
55 Fl_Bitmap(const char *bits, int bits_length, int W, int H);
56 void draw(const Fl_Widget *w) const;
57 void label(Fl_Widget *w) const;
58 void draw(const char *bits) const;
59 void draw(int X, int Y, int W, int H) const;
60 void draw(const Fl_Menu_Item *m) const;
61 void draw(const Fl_Widget *w) const;
62 virtual ~Fl_Bitmap();
63 void uncache();
64
65 #endif

34.14 Fl_BMP_Image.H

1 //
2 // BMP image header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
```cpp
/*
file

Fl_BMP_Image widget . */

#ifndef Fl_BMP_Image_H
#define Fl_BMP_Image_H
#include "Fl_Image.H"

class FL_EXPORT Fl_BMP_Image : public Fl_RGB_Image {
public:
  Fl_BMP_Image(const char * filename);
  Fl_BMP_Image(const char * imagename, const unsigned char *data, const long length = -1);
protected:
  void load_bmp_(class Fl_Image_Reader &rdr, int ico_height = 0, int ico_width = 0);
};

#endif
```

### 34.15 Fl_Box.H File Reference

**Fl_Box** widget.

```cpp
#include "Fl_Widget.H"
```

**Classes**

- class **Fl_Box**

  *This widget simply draws its box, and possibly its label.*

### 34.15.1 Detailed Description

**Fl_Box** widget.

### 34.16 Fl_Box.H

Go to the documentation of this file.

```cpp
#ifndef Fl_Box_H
#define Fl_Box_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

class FL_EXPORT Fl_Box : public Fl_Widget {
protected:
  void draw() FL_OVERRIDE;
public:
```
34.17 Fl_Browser.H

 /* file
 Fl_Browser widget. */

 // Browser header file for the Fast Light Tool Kit (FLTK).
 // Copyright 1998-2016 by Bill Spitzak and others.
 // This library is free software. Distribution and use rights are outlined in
 // the file "COPYING" which should have been included with this file. If this
 // file is missing or damaged, see the license at:
 // https://www.fltk.org/COPYING.php
 // Please see the following page on how to report bugs and issues:
 // https://www.fltk.org/bugs.php

 #ifndef Fl_Browser_H
 #define Fl_Browser_H

 #include "Fl_Browser_.H"
 #include "Fl_Image.H"

 struct FL_BLINE;

 class FL_EXPORT Fl_Browser : public Fl_Browser_ {
  
  // the array of lines
  FL_BLINE *first;
  FL_BLINE *last;
  FL_BLINE *cache;
  int cacheline; // line number of cache
  int lines; // Number of lines
  int full_height_; // full_height()
  const int * column_widths_; // Integer array of column widths
  char format_char_; // alternative to @-sign
  char column_char_; // alternative to tab

 protected:
  
  // required routines for Fl_Browser_ subclass:
  void* item_first() const FL_OVERRIDE;
  void* item_next(void* item) const FL_OVERRIDE;
  void* item_prev(void* item) const FL_OVERRIDE;
  void* item_last() const FL_OVERRIDE;
  int item_selected(void* item) const FL_OVERRIDE;
  void item_select(void* item, int val) FL_OVERRIDE;
  int item_height(void* item) const FL_OVERRIDE;
  int item_width(void* item) const FL_OVERRIDE;
  void item_draw(void* item, int X, int Y, int W, int H) const FL_OVERRIDE;
  int full_height() const FL_OVERRIDE;
  int incr_height() const FL_OVERRIDE;
  const char* item_text(void* item) const FL_OVERRIDE;
  int item_swap(void* a, void* b) FL_OVERRIDE { swap((FL_BLINE*)a, (FL_BLINE*)b); }
  void* item_at(int line) const FL_OVERRIDE { return (void*)find_line(line); }
  FL_BLINE* find_line(int line) const;
  FL_BLINE* _remove(int line);
  void insert(int line, FL_BLINE* item);
  int lineno(void* item) const;
  void swap(FL_BLINE* a, FL_BLINE* b);

 public:
  
  void remove(int line);
  void add(const char* newtext, void* d = 0);
  void insert(int line, const char* newtext, void* d = 0);
  void move(int to, int from);
  int load(const char* filename);

  #endif
void swap(int a, int b);
void clear();
int size() const { return lines; }
void size(int W, int H) { Fl_Widget::size(W, H); }
Fl_Fontsize textsize() const { return Fl_Browser::textsize(); }
/
Sets the default text size for the lines in the browser to newSize.
Defined and documented in Fl_Browser.cxx
/*
Sets the default text size for the lines in the browser to newSize.
Defined and documented in Fl_Browser.cxx
*/
void textsize(Fl_Fontsize newSize);
int topline() const ;
enum Fl_Line_Position { TOP, BOTTOM, MIDDLE };
void lineposition(int line, Fl_Line_Position pos);
void topoline(int line) { lineposition(line, TOP); }
void bottomline(int line) { lineposition(line, BOTTOM); }
void middeline(int line) { lineposition(line, MIDDLE); }
int select(int line, int val=1);
int selected(int line) const ;
void show(int line);
void show() FL_OVERRIDE { Fl_Widget::show(); }
void hide(int line);
void hide() FL_OVERRIDE { Fl_Widget::hide(); }
int visible(int line) const ;
int value() const ;
int displayed(int line) const { return Fl_Browser::displayed(find_line(line)); }
void make_visible(int line) {
  if (line < 1) Fl_Browser::display(find_line(1));
  else if (line > lines) Fl_Browser::display(find_line(lines));
  else Fl_Browser::display(find_line(line));
}

// icon support

34.18 Fl_Browser_.H

1 //
2 // Common browser header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2016 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //*/
17
18 Fl_Browser_. H

Generated by Doxygen
#ifndef Fl_Browser__H
#define Fl_Browser__H

#ifndef Fl_Group_H
#include "Fl_Group.H"
#endif
#include "Fl_Scrollbar.H"
#include <FL/Fl.H> // Fl::scrollbar_size()

#define FL_NORMAL_BROWSER 0
#define FL_SELECT_BROWSER 1
#define FL_HOLD_BROWSER 2
#define FL_MULTI_BROWSER 3
#define FL_SORT_ASCENDING 0
#define FL_SORT_DESCENDING 1
#define FL_SORT_CASEINSENSITIVE 0x2

class FL_EXPORT Fl_Browser_ : public Fl_Group {
  int position_; // where user wants it scrolled to
  int real_position_; // the current vertical scrolling position
  int hposition_; // where user wants it panned to
  int real_hposition_; // the current horizontal scrolling position
  int offset_; // how far down top_ item the real_position is
  int max_width; // widest object seen so far
  uchar has_scrollbar_; // which scrollbars are enabled
  Fl_Font textfont_; // font for text
  Fl_Fontsize textsize_; // size for text
  Fl_Color textcolor_; // color for text
  void * top_; // which item scrolling position is in
  void * selection_; // which is selected (except for FL_MULTI_BROWSER)
  Fl_Scrollbar scrollbar; // vertical scrollbar
  Fl_Scrollbar hscrollbar; // horizontal scrollbar

  protected:
  // All of the following must be supplied by the subclass:
  virtual void *item_first() const = 0;
  virtual void *item_next(void *item) const = 0;
  virtual void *item_prev(void *item) const = 0;
  virtual int item_height(void *item) const = 0;
  virtual int item_width(void *item) const = 0;
  virtual int item_quick_height(void *item) const ;
  virtual void item_draw(void *item,int X,int Y,int W,int H) const = 0;
  virtual const char *item_text(void *item)const { (void)item; return 0L; }
  virtual void item_swap(void *a,void *b) { (void)a; (void)b; }
  virtual void *item_at(int index)const { (void)index; return 0L; }
  // you don't have to provide these but it may help speed it up:
  virtual int full_width() const ; // current width of all items
  virtual int full_height() const ; // current height of all items
  // These only need to be done by subclass if you want a multi-browser:
  virtual void item_select(void *item,int val=1);
  virtual int item_selected(void *item) const ;

  // things the subclass may want to call:
  void *top()const { return top_; }
  void *selection()const { return selection_; }
  void new_list(); // completely clobber all data, as though list replaced

  protected:
  // things the subclass may want to call:
  virtual void *item_first() const = 0;
  virtual void *item_next(void *item) const = 0;
  virtual void *item_prev(void *item) const = 0;
  virtual int item_height(void *item) const = 0;
  virtual int item_width(void *item) const = 0;
  virtual int item_quick_height(void *item) const ;
  virtual void item_draw(void *item,int X,int Y,int W,int H) const = 0;
  virtual const char *item_text(void *item)const { (void)item; return 0L; }
  virtual void item_swap(void *a,void *b) { (void)a; (void)b; }
  virtual void *item_at(int index)const { (void)index; return 0L; }
  // you don't have to provide these but it may help speed it up:
  virtual int full_width() const ; // current width of all items
  virtual int full_height() const ; // current height of all items
  // These only need to be done by subclass if you want a multi-browser:
  virtual void item_select(void *item,int val=1);
  virtual int item_selected(void *item) const ;

  // things the subclass may want to call:
  void *top()const { return top_; }
  void *selection()const { return selection_; }
  void new_list(); // completely clobber all data, as though list replaced
  void deleteItem(void *item); // get rid of any pointers to item
  void deleting(void *item); // get rid of any pointers to item
  void replacing(void *a,void *b); // change a pointer to b
  void swapping(void *a,void *b); // exchange pointers a and b
  void inserting(void *a,void *b); // insert b near a
  int displayed(void *item) const ; // true if this item is visible
  void redraw_line(void *item); // minimal update, no change in size
  void redraw_lines() { damage(FL_DAMAGE_SCROLL); } // redraw all of them
  void bbox(int &X,int &Y,int &W,int &H) const;
  int leftedge() const; // x position after scrollbar & border
  void *find_item(int ypos); // item under mouse
  void draw() FL_OVERRIDE;
  Fl_Browser_(int X,int Y,int W,int H, const char *L=0);
public:
  Fl_Scrollbar scrollbar;
  Fl_Scrollbar hscrollbar;
  int handle(int event) FL_OVERRIDE;
  void resize(int X,int Y,int W,int H) FL_OVERRIDE;
  int select(void *item,int val=1,int docallbacks=0);
int select_only(void *item, int docallbacks=0);
int deselect(int docallbacks=0);

# FL_DEPRECATED("in 1.4.0 - use vposition() instead",
int position() const { return vposition(); }
]

void vposition(int pos); // scroll to here
FL_DEPRECATED("in 1.4.0 - use vposition(pos) instead",
void position(int x, int y) { Fl_Group::position(x, y); }

void position(int pos) { return vposition(pos); }

enum { // values for has_scrollbar()
HORIZONTAL = 1,
VERTICAL = 2,
BOTH = 3,
ALWAYS_ON = 4,
HORIZONTAL_ALWAYS = 5,
VERTICAL_ALWAYS = 6,
BOTH_ALWAYS = 7
};
uchar has_scrollbar() const { return has_scrollbar_; }

void has_scrollbar(uchar mode) { has_scrollbar_ = mode; }

Fl_Font textfont() const { return textfont_; }

void textfont(Fl_Font font) { textfont_ = font; }

Fl_Fontsize textsize() const { return textsize_; }

void textsize(Fl_Fontsize newSize) { textsize_ = newSize; }

Fl_Color textcolor() const { return textcolor_; }

void textcolor(Fl_Color col) { textcolor_ = col; }

int scrollbar_size() const { return scrollbar_size_; }

void scrollbar_size(int newSize) { scrollbar_size_ = newSize; }

int scrollbar_width() const { return Fl::scrollbar_size(); }

void scrollbar_width(int width) { Fl::scrollbar_size(width); }

void scrollbar_right() { scrollbar.align(FL_ALIGN_RIGHT); }

void scrollbar_left() { scrollbar.align(FL_ALIGN_LEFT); }

void sort(int flags=0);

34.19 Fl_Button.H

// Button header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2014 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#define FL_NORMAL_BUTTON 0
#define FL_TOGGLE_BUTTON 1
#define FL_RADIO_BUTTON (FL_RESERVED_TYPE+2)
#define FL_HIDDEN_BUTTON 3

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);

class Fl_Widget_Tracker;

class FL_EXPORT Fl_Button : public Fl_Widget {
    int shortcut_;  
    char value_;    
    char oldval;   
    uchar down_box_; 
    protected: 
        static Fl_Widget_Tracker *key_release_tracker;
        static void key_release_timeout(void *);
        void simulate_key_action();
        void draw() FL_OVERRIDE;

    public:
        int handle(int) FL_OVERRIDE;

    Fl_Button(int X, int Y, int W, int H, const char *L = 0);
    int value(int v);
    char value()const {return value_;}
    int set() {return value(1);}
    int clear() {return value(0);}
    void setonly(); // this should only be called on FL_RADIO_BUTTONs
    int shortcut()const {return shortcut_;}
    void shortcut(const char *s) {shortcut(fl_old_shortcut(s));}
    Fl_Boxtype down_box()const {return (Fl_Boxtype)down_box_;}
    void down_box(Fl_Boxtype b) {down_box_ = b;}
    void shortcut(const char *, Fl_Shortcut s) {shortcut(s);}
    Fl_Color down_color()const {return selection_color();}
    void down_color(unsigned c) {selection_color(c);}
}

#define FL_HIDDEN_BUTTON 3

34.20 Fl_Cairo.H File Reference

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).

#include <FL/Fl.H>
#include <cairo.h>

Classes

• class Fl_Cairo_State

    Contains all the necessary info on the current cairo context.

34.20.1 Detailed Description

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).
In FLTK 1.3.x this header file (`Fl_Cairo.H`) included the platform specific Cairo headers. This is no longer true since 1.4.0.

This header file is platform agnostic. If you need platform specific Cairo headers you need to `#include <FL/Fl.H>` before you include any other FLTK header which is officially required anyway. Since FLTK 1.4.0 the preprocessor constants `FLTK_HAVE_CAIRO` and/or `FLTK_HAVE_CAIROEXT` are defined in `<FL/Fl.H>` by including `<FL/fl_config.h>`.

34.21  **Fl_Cairo.H**

Go to the documentation of this file.

```c
// Main Cairo support header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//    https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//    https://www.fltk.org/bugs.php

#ifndef FL_CAIRO_H
#define FL_CAIRO_H

#include <FL/Fl.H>

#ifdef FLTK_HAVE_CAIRO

#include <cairo.h>

class FL_EXPORT Fl_Cairo_State {
public:
    Fl_Cairo_State()
        : cc_(0), own_cc_(false), autolink_(false), window_(0), gc_(0) {}

    cairo_t *cc() const { return cc_; }
    bool autolink() const { return autolink_; }

    void cc(cairo_t *c, bool own = true) {
        if (cc_ && own_cc_)
            cairo_destroy(cc_);
        cc_ = c;
        if (!cc_)
            window_ = 0;
        own_cc_ = own;
    }

    void autolink(bool b);
    void window(void *w) { window_ = w; }
    void *window() const { return window_; }

    void gc(void *c) { gc_ = c; }
    void *gc() const { return gc_; }

private:
    cairo_t *cc_;    // contains the unique autoupdated cairo context
    bool own_cc_;    // indicates whether we must delete the cc, useful for internal cleanup
    bool autolink_;  // false by default, prevents the automatic cairo mapping on fltk windows
    // for custom cairo implementations.
    void *window_;   // for keeping track internally of last win+gc treated
};
#endif // FLTK_HAVE_CAIRO
#endif // FL_CAIRO_H
```

34.22  **Fl_Cairo_Window.H** File Reference

`Fl_Cairo_Window`, an FLTK window incorporating a Cairo draw callback.
#include <FL/fl_config.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>

Classes

- class Fl_Cairo_Window

  This defines an FLTK window with Cairo support.

34.22.1 Detailed Description

Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback.

34.23 Fl_Cairo_Window.H

Go to the documentation of this file.

1 //
2 // Fl_Cairo_Window header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2023 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 #ifndef FL_CAIRO_WINDOW_H
17 # define FL_CAIRO_WINDOW_H
18
19 #if FLTK_HAVE_CAIRO
20
21 # include <FL/Fl.H>
22 # include <FL/Fl_Double_Window.H>
23
24 class FL_EXPORT Fl_Cairo_Window : public Fl_Double_Window {
25    public:
26     Fl_Cairo_Window(int W, int H, const char *L = 0)
27          : Fl_Double_Window(W, H, L), draw_cb_(0) {}
28     Fl_Cairo_Window(int X, int Y, int W, int H, const char *L = 0)
29          : Fl_Double_Window(X, Y, W, H, L), draw_cb_(0) {}
30
31     virtual void draw() FL_OVERRIDE {
32         Fl_Double_Window::draw();
33         if (draw_cb_) { // call the Cairo draw callback
34             cairo_t* def = Fl::cairo_autolink_context(); // get the Cairo context
35             if (def) {
36                 Fl::cairo_make_current(this);
37                 cairo_t* def = Fl::cairo_autolink_context();
38                 Fl::cairo_make_current(this);
39                 draw_cb_(this, def);
40                 Fl::cairo_flush(def);
41                 Fl::cairo_flush(def);
42             }
43         }
44     }
45
46     protected:
47     void draw() FL_OVERRIDE {
48         Fl_Double_Window::draw();
49         if (draw_cb_) { // call the Cairo draw callback
50             cairo_t* def = Fl::cairo_autolink_context(); // get the Cairo context
51             if (def) {
52                 Fl::cairo_make_current(this);
53                 draw_cb_(this, def);
54                 Fl::cairo_flush(def);
55                 Fl::cairo_flush(def);
56             }
57         }
58     }
59
60     public:
61     typedef void ( *cairo_draw_cb) (Fl_Cairo_Window* self, cairo_t* def);
62     void set_draw_cb(cairo_draw_cb cb) { draw_cb_ = cb; }
63
64     private:
65     cairo_draw_cb draw_cb_;
#fl_casts.H

```c
// Experimental inline "cast functions" for the Fast Light Toolkit (FLTK).
// See also issue #109: "VS2017 warnings when building fltk 1.4.x"
// Copyright 1998-2021 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef _FL_fl_casts_H_
#define _FL_fl_casts_H_

#include <FL/platform_types.h>

inline char fl_char(void *v) { return (char)(fl_intptr_t)v; }
inline int fl_int(void *v) { return (int)(fl_intptr_t)v; }
inline long fl_long(void *v) { return (long)(fl_intptr_t)v; }

inline unsigned char fl_uchar(void *v) { return (unsigned char)(fl_uintptr_t)v; }
inline unsigned int fl_uint(void *v) { return (unsigned int)(fl_uintptr_t)v; }
inline unsigned long fl_ulong(void *v) { return (unsigned long)(fl_uintptr_t)v; }

// the following conversions can be used to silence MSVC warning C4312:
// 'type cast': conversion from '<type>' to 'void *' of greater size
inline void *fl_voidptr(int v) { return (void *)(fl_intptr_t)v; }
inline void *fl_voidptr(unsigned int v) { return (void *)(fl_uintptr_t)v; }
inline void *fl_voidptr(long v) { return (void *)(fl_intptr_t)v; }
inline void *fl_voidptr(unsigned long v) { return (void *)(fl_uintptr_t)v; }

#endif /* _FL_fl_casts_H_ */
```

# Fl_Chart.H File Reference

**Fl_Chart** widget.

```
#include "Fl_Widget.H"
```

## Classes

- **class Fl_Chart**
  
  `Fl_Chart` displays simple charts.

- **struct FL_CHART_ENTRY**
  
  `For internal use only.`

## Macros

- **#define FL_BAR_CHART 0**
  
  `type()` for Bar Chart variant

- **#define FL_CHART_LABEL_MAX 18**
  
  `max label length for entry`

- **#define FL_CHART_MAX 128**
  
  `max entries per chart`

- **#define FL_FILL_CHART 3**
  
  `type()` for Fill Chart variant

- **#define FL_FILLED_CHART FL_FILL_CHART**
  
  `for compatibility`

- **#define FL_HORBAR_CHART 1**

```
34.25.1 Detailed Description

**Fl.Chart** widget.

### 34.26 Fl.Chart.H

Go to the documentation of this file.

```c
#ifndef Fl_Chart_H
#define Fl_Chart_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type()
#define FL_BAR_CHART 0
#define FL_HORBAR_CHART 1
#define FL_LINE_CHART 2
#define FL_FILL_CHART 3
#define FL_SPIKE_CHART 4
#define FL_PIE_CHART 5
#define FL_SPECIALPIE_CHART 6
#define FL_FILLED_CHART FL_FILL_CHART

#define FL_CHART_MAX 128
#define FL_CHART_LABEL_MAX 18

struct FL_CHART_ENTRY {
    float val;
    unsigned col;
    char str[FL_CHART_LABEL_MAX + 1];
};

class FL_EXPORT Fl_Chart : public Fl_Widget {
    int numb;
    int maxnumb;
    int sizenumb;
    FL_CHART_ENTRY *entries;
    double min, max;
    uchar autosize_;
    FL_Font textfont_;
    FL_Fsize textsize_;
    FL_Color textcolor_;

protected:
    void draw() FL_OVERRIDE;
    // (static) protected draw methods (STR 2022)
    // these methods are documented in src/Fl_Chart.cxx
```

Generated by Doxygen
static void draw_barchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

static void draw_horbarchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

static void draw_linechart(int type, int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

static void draw_piechart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[], int special, Fl_Color textcolor);

public:
Fl_Chart(int X, int Y, int W, int H, const char *L = 0);

~Fl_Chart();

void clear();

void add(double val, const char *str = 0, unsigned col = 0);

void insert(int ind, double val, const char *str = 0, unsigned col = 0);

void replace(int ind, double val, const char *str = 0, unsigned col = 0);

void bounds(double *a, double *b)const {
    *a = min;
    *b = max;
}

void bounds(double a, double b);

int size()const { return numb; }

int maxsize()const { return maxnumb; }

Fl_Font textfont()const { return textfont_; }

void textfont(Fl_Font s) { textfont_ = s; }

Fl_Fontsize textsize()const { return textsize_; }

void textsize(Fl_Fontsize s) { textsize_ = s; }

Fl_Color textcolor()const { return textcolor_; }

void textcolor(Fl_Color n) { textcolor_ = n; }

uchar autosize()const { return autosize_; }

void autosize(uchar n) { autosize_ = n; }

};

#endif

34.27  Fl_Check_Browser.H

1 //
2 // Fl_Check_Browser header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2020 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //
17 // Fl_Check_Browser widget . */
18 #ifndef Fl_Check_Browser_H
19 #define Fl_Check_Browser_H
20
21 Generated by Doxygen
```cpp
#include "Fl.H"
#include "Fl_Browser_.H"

class FL_EXPORT Fl_Check_Browser : public Fl_Browser_ {
  protected:
    /* required routines for Fl_Browser_subclass: */
    void *item_first() const FL_OVERRIDE;
    void *item_next(void *) const FL_OVERRIDE;
    void *item_prev(void *) const FL_OVERRIDE;
    int item_height(void *) const FL_OVERRIDE;
    int item_width(void *) const FL_OVERRIDE;
    void item_draw(void *, int, int, int, int) const FL_OVERRIDE;
    int item_selected(void *) const FL_OVERRIDE;
    void item_select(void *, int) FL_OVERRIDE;
    int item_selected(void *) const FL_OVERRIDE;
    const char *item_text(void *, void *) const FL_OVERRIDE;

    public:
    void *item_at(int index) const FL_OVERRIDE;
    void item_swap(int ia, int ib);
    void item_swap(void *, void *) FL_OVERRIDE;

    /* private data */
    
    private:
    struct cb_item {
      cb_item *next;
      cb_item *prev;
      char checked;
      char selected;
      char *text;
    };

    public: // IRIX 5.3 C++ compiler doesn't support private structures...
      #ifndef FL_DOXYGEN
    struct cb_item {
      cb_item *next;
      cb_item *prev;
      char checked;
      char selected;
      char *text;
    };
      #endif // !FL_DOXYGEN
    
    private:
    cb_item *first;
    cb_item *last;
    cb_item *cache;
    int cached_item;
    int nitems;
    int nchecked;
    int lineno(cb_item *) const;
    cb_item *find_item(int) const;

    public:
    Fl_Check_Browser(int x, int y, int w, int h, const char *l = 0);
    ~Fl_Check_Browser() { clear(); }
    int add(char *s); // add an (unchecked) item
    int add(char *s, int b); // add a item and set checked
    // both return the new nitems()
    int remove(int item); // delete an item. Returns nitems()

    // inline const char * methods to avoid breaking binary compatibility...
    int add(const char *s) { return add((char *)s); }
    int add(const char *s, int b) { return add((char *)s, b); }

    void clear(); // delete all items
    int nitems() const { return nitems_; }
    int nchecked() const { return nchecked_; }
    int checked(int item) const;
    void checked(int item, int b);
    void set_checked(int item) { checked(item, 1); }
    void check_all();
    void check_none();
    int value() const; // currently selected item
    char *text(int item) const; // returns pointer to internal buffer

    protected:
    int handle(int) FL_OVERRIDE;
    }
  
  #endif // Fl_Check_Browser_H

};
```

### 34.28 Fl_Check_Browser.H

// Check button header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2014 by Bill Spitzak and others.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this

-- Generated by Doxygen --

```cpp
#ifndef Fl_Check_Button_H
#define Fl_Check_Button_H

#include "Fl_Light_Button.H"

class FL_EXPORT Fl_Check_Button : public Fl_Light_Button {
public:
  Fl_Check_Button(int X, int Y, int W, int H, const char *L = 0);
};
#endif
```

---

### 34.29 Fl_Choice.H

```cpp
#ifndef Fl_Choice_H
#define Fl_Choice_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Choice : public Fl_Menu_ {
protected:
  void draw() FL_OVERRIDE;
public:
  int handle(int) FL_OVERRIDE;
  Fl_Choice(int X, int Y, int W, int H, const char *L = 0);
  int value() const {return Fl_Menu_::value();}
  int value(int v);
  int value(const Fl_Menu_Item * v);
};
#endif
```

---

### 34.30 Fl_Clock.H

```cpp
#ifndef Fl_Clock_H
#define Fl_Clock_H

// Clock header file for the Fast Light Tool Kit (FLTK).

// Copyright 1998-2017 by Bill Spitzak and others.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
```

---

Generated by Doxygen
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 //
17 /*
18  * Fl_Clock, Fl_Clock_Output widgets . */
19 */
20 ifndef Fl_Clock_H
21 define Fl_Clock_H
22
23 ifndef Fl_Widget_H
24 include "Fl_Widget.H"
25 endif
26
27 // Values for type():
28 // Please change doxygen documentation below (class Fl_Clock_Output)
29 // accordingly as well when changing the following type values:
30
31 #define FL_SQUARE_CLOCK 0
32 #define FL_ROUND_CLOCK 1
33 #define FL_ANALOG_CLOCK FL_SQUARE_CLOCK
34 #define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK
35
36 // fabien: Please keep the horizontal formatting of both images in class desc,
37 // don't lose vertical space for nothing!
38
39 65 class FL_EXPORT Fl_Clock_Output : public Fl_Widget {
40     int hour_, minute_, second_; 
41     ulong value_;    // draw shadows of hands
42     void drawhands(Fl_Color,Fl_Color); // part of draw
43     void draw() FL_OVERRIDE;
44     void draw(int X, int Y, int W, int H);
45     public:
46     Fl_Clock_Output(int X, int Y, int W, int H, const char *L = 0);
47     void value(ulong v); // set to this Unix time
48     void value(int H, int m, int s);
49     ulong value()const {return value_;
50     int hour()const {return hour_;
51     int minute()const {return minute_;
52     int second()const {return second_;
53     int shadow()const {return shadow_;
54     void shadow(int mode) { shadow_ = mode ? 1 : 0;
55 
56 // a Fl_Clock displays the current time always by using a timeout:
57 152 class FL_EXPORT Fl_Clock : public Fl_Clock_Output {
58 public:
59     int handle(int) FL_OVERRIDE;
60     Fl_Clock(int X, int Y, int W, int H, const char *L = 0);
61     Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L);
62     Fl_Clock();
63     ~Fl_Clock();
64     };
65 };
66 endif

34.31 Fl_Color_Chooser.H File Reference

Fi_Color_Chooser widget.
#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>
Classes

- class Fl_Color_Chooser

  The Fl_Color_Chooser widget provides a standard RGB color chooser.

### 34.31.1 Detailed Description

**Fl_Color_Chooser** widget.

### 34.32 Fl_Color_Chooser.H

Go to the documentation of this file.

```cpp
// Color chooser header file for the Fast Light Tool Kit (FLTK).

// Copyright 1998-2019 by Bill Spitzak and others.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

// The color chooser object and the color chooser popup. The popup
// is just a window containing a single color chooser and some boxes
// to indicate the current and cancelled color.

#ifndef Fl_Color_Chooser_H
#define Fl_Color_Chooser_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>

#ifndef FL_DOXYGEN

class FL_EXPORT Flcc_HueBox : public Fl_Widget {
  int px, py;
protected:
  void draw() FL_OVERRIDE;
  int handle_key(int);
public:
  int handle(int) FL_OVERRIDE;
  Flcc_HueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
    px = py = 0;}
};

class FL_EXPORT Flcc_ValueBox : public Fl_Widget {
  int py;
protected:
  void draw() FL_OVERRIDE;
  int handle_key(int);
public:
  int handle(int) FL_OVERRIDE;
  Flcc_ValueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
    py = 0;}
};

class FL_EXPORT Flcc_Value_Input : public Fl_Value_Input {
public:
  int format(char *) FL_OVERRIDE;
  Flcc_Value_Input(int X, int Y, int W, int H) : Fl_Value_Input(X,Y,W,H) {}
};
#endif // !FL_DOXYGEN

class FL_EXPORT Fl_Color_Chooser : public Fl_Group {
  Flcc_HueBox huebox;
  Flcc_ValueBox valuebox;
  Fl_Choice choice;
  Flcc_Value_Input rvalue;
  Flcc_Value_Input gvalue;
  Flcc_Value_Input bvalue;
  Fl_Box resize_box;

Generated by Doxygen
```
double hue_, saturation_, value_;  
double r_, g_, b_;  
void set_valuators();  
static void rgb_cb(Fl_Widget*, void*);  
static void mode_cb(Fl_Widget*, void*);  
public:  
int handle(int e) FL_OVERRIDE;  
int mode() {return choice.value();}  
void mode(int newMode);  
double hue()const {return hue_;}  
double saturation()const {return saturation_;}  
double value()const {return value_;}  
double r()const {return r_;}  
double g()const {return g_;}  
double b()const {return b_;}  
int hsv(double H, double S, double V);  
int rgb(double R, double G, double B);  
static void hsv2rgb(double H, double S, double V, double& R, double& G, double& B);  
static void rgb2hsv(double R, double G, double B, double& H, double& S, double& V);  
Fl_Color_Chooser(int X, int Y, int W, int H, const char *L = 0);  
};  

FL_EXPORT int fl_color_chooser(const char* name, double& r, double& g, double& b, int m=-1);  
FL_EXPORT int fl_color_chooser(const char* name, uchar& r, uchar& g, uchar& b, int m=-1);  
#endif
This implies to link cairo.lib in all FLTK based apps.

#undef FLTK_HAVE_CAIROEXT

Do we use X11 for the current platform?

#define FLTK_USE_X11 1

Do we use Cairo to draw to the display?

#undef FLTK_USE_CAIRO

Do we use Wayland for the current platform?

#undef FLTK_USE_WAYLAND

ifndef _FL_fl_config_h_
endif /*_FL_fl_config_h_*/

34.34  Fl_Copy_Surface.H

ifndef Fl_Copy_Surface_H
#define Fl_Copy_Surface_H

#include <FL/Fl_Widget_Surface.H>

class FL_EXPORT Fl_Copy_Surface : public Fl_Widget_Surface {
private:
  class Fl_Copy_Surface_Driver *platform_surface;
protected:
  void translate(int x, int y) FL_OVERRIDE;
  void untranslate() FL_OVERRIDE;
public:
  Fl_Copy_Surface(int w, int h);
  ~Fl_Copy_Surface();
  void set_current() FL_OVERRIDE;
  bool is_current() FL_OVERRIDE;
  int w();
  int h();
  void origin(int x, int y) FL_OVERRIDE;
  void origin(int x, int y) FL_OVERRIDE;
  int printable_rect(int *w, int *h) FL_OVERRIDE;
};

class Fl_Copy_Surface_Driver : public Fl_Widget_Surface {
friend class Fl_Copy_Surface;
protected:
  int width;
  int height;
  Fl_Copy_Surface_Driver(int w, int h) : Fl_Widget_Surface(NULL), width(w), height(h) {}
virtual ~Fl_Copy_Surface_Driver() {}  
void set_current() FL_OVERRIDE = 0;  
void translate(int x, int y) FL_OVERRIDE = 0;  
void untranslate() FL_OVERRIDE = 0;  
int printable_rect(int *w, int *h) FL_OVERRIDE;  
static Fl_Copy_Surface_Driver *newCopySurfaceDriver(int w, int h);  
};  
};  
#endif // Fl_Copy_Surface_H

34.35 Fl_Counter.H

/*
  Fl_Counter widget.
*/

// A numerical value with up/down step buttons. From Forms.

#ifndef Fl_Counter_H
#define Fl_Counter_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

// values for type():
#define FL_NORMAL_COUNTER 0
#define FL_SIMPLE_COUNTER 1

class FL_EXPORT Fl_Counter : public Fl_Valuator {
  Fl_Font textfont_;  
  Fl_Fontsize textsize_;  
  Fl_Color textcolor_;  
  double lstep_;  
  uchar mouseobj_;  
  static void repeat_callback(void *);
  int calc_mouseobj();
  void increment_cb();

protected:
  void draw() FL_OVERRIDE;
  // compute widths of arrow boxes
  void arrow_widths(int &w1, int &w2);

public:
  int handle(int) FL_OVERRIDE;

  Fl_Counter(int X, int Y, int W, int H, const char * L = 0);
  ~Fl_Counter();

  void lstep(double a) {lstep_ = a;}
  void step(double a, double b) {Fl_Valuator::step(a); lstep_ = b;}
  void step(double a) {Fl_Valuator::step(a);}
  double step()const {return Fl_Valuator::step();}

  Fl_Font textfont()const {return textfont_;
  void textfont(Fl_Font s) {textfont_ = s;}

  Fl_Fontsize textsize()const {return textsize_;
  void textsize(Fl_Fontsize s) {textsize_ = s;}

  Fl_Color textcolor()const {return textcolor_;
  void textcolor(Fl_Color s) {textcolor_ = s;}

Generated by Doxygen
34.36 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.

34.36.1 Detailed Description

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

34.37 Fl_Device.H

Go to the documentation of this file.
# Fl_Dial.H

## 34.38 Fl_Dial.H

1 // 2 // Dial header file for the Fast Light Tool Kit (FLTK). 3 // 4 // Copyright 1998-2022 by Bill Spitzak and others. 5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php 15 // 16 /* 17 \file Fl_Dial widget . */ 18 #ifndef Fl_Dial_H 19 #define Fl_Dial_H 20 21 #ifndef Fl_Valuator_H 22 #include "Fl_Valuator.H" 23 #endif 24 25 // values for type(): 26 #define FL_NORMAL_DIAL 0 27 #define FL_LINE_DIAL 1 28 #define FL_FILL_DIAL 2 29 30 #endif 31 #endif // Fl_Dial_H

## 34.39 Fl_Double_Window.H

1 // 2 // Double-buffered window header file for the Fast Light Tool Kit (FLTK).
/*
 file
 Fl_Double_Window widget. */

#ifndef Fl_Double_Window_H
#define Fl_Double_Window_H

#include "Fl_Window.H"

class FL_EXPORT Fl_Double_Window : public Fl_Window
{
 public:
  Fl_Double_Window *as_double_window() FL_OVERRIDE {return this; }
  void show() FL_OVERRIDE;
  void show(int a, char **b) {Fl_Window::show(a,b);}
  void resize(int,int,int,int) FL_OVERRIDE;
  void hide() FL_OVERRIDE;
  void flush() FL_OVERRIDE;
  ~Fl_Double_Window();

 Fl_Double_Window(int W, int H, const char *l = 0);

 Fl_Double_Window(int X, int Y, int W, int H, const char *l = 0);

};

#endif

34.40 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

• int fl_add_symbol (const char *name, void(*drawit)(Fl_Color), int scalable)
  Adds a symbol to the system.

• int fl_antialias ()
  Return whether line drawings are currently antialiased.

• void fl_antialias (int state)
  Turn antialiased line drawings ON or OFF, if supported by platform.

• void fl_arc (double x, double y, double r, double start, double end)
  Add a series of points to the current path on the arc of a circle.

• void fl_arc (int x, int y, int w, int h, double a1, double a2)
  Draw ellipse sections using integer coordinates.

• void fl_begin_complex_polygon ()
Start drawing a complex filled polygon.

- void fl_begin_line()

  Start drawing a list of lines.

- void fl_begin_loop()

  Start drawing a closed sequence of lines.

- void fl_begin_offscreen(Fl_Offscreen ctx)

  Send all subsequent drawing commands to this offscreen buffer.

- void fl_begin_points()

  Start drawing a list of points.

- void fl_begin_polygon()

  Start drawing a convex filled polygon.

- char fl_can_do_alpha_blending()

  Check whether platform supports true alpha blending for RGBA images.

- Fl_RGB_Image * fl_capture_window(Fl_Window *win, int x, int y, int w, int h)

  Captures the content of a rectangular zone of a mapped window.

- void fl_chord(int x, int y, int w, int h, double a1, double a2)

  fl_chord declaration is a place holder - the function does not yet exist.

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

  fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.

- void fl_clip(int x, int y, int w, int h)

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

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

  Intersect a rectangle with the current clip region and return the bounding box of the result.

- Fl_Region fl_clip_region()

  Return the current clipping region.

- void fl_clip_region(Fl_Region r)

  Replace the top of the clipping stack with a clipping region of any shape.

- Fl_Color fl_color()

  Return the last fl_color() that was set.

- void fl_color(Fl_Color c)

  Set the color for all subsequent drawing operations.

- void fl_color(int c)

  for back compatibility - use fl_color(Fl_Color c) instead

- void fl_color(uchar r, uchar g, uchar b)

  Set the color for all subsequent drawing operations.

- void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)

  Copy a rectangular area of the given offscreen buffer into the current drawing destination.

- Fl_Offscreen fl_create_offscreen(int w, int h)

  Creation of an offscreen graphics buffer.

- void fl_cursor(Fl_Cursor)

  Sets the cursor for the current window to the specified shape and colors.

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

  Add a series of points on a Bezier curve to the path.

- void fl_delete_offscreen(Fl_Offscreen ctx)

  Deletion of an offscreen graphics buffer.

- int fl_descent()

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

- void fl_draw(const char *str, int n, int x, int y)

  Draws starting at the given x, y location a UTF-8 string of length n bytes.
• void fl_draw (const char *str, int x, int y)
  Draw a nul-terminated UTF-8 string starting at the given x, y location.

• void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1)
  Fancy string drawing function which is used to draw all the labels.

• void fl_draw (const char *str, int x, int y, int 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)
  Draw at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.

• void fl_draw (int angle, const char *str, int x, int y)
  Draw a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.

• void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)
  Draw an "arrow like" GUI element for the selected scheme.

• void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)
  Draws a box using given type, position, size and color.

• void fl_draw_check (Fl_Rect bb, Fl_Color col)
  Draw a check mark inside the given bounding box.

• void fl_draw_circle (int x0, int y0, int d, Fl_Color color)
  Draw a potentially small, filled circle as a GUI element.

• 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_pixels (char *const *data, int x, int y, Fl_Color bg=FL_GRAY)
  Draw XPM image data, with the top-left corner at the given position.

• int fl_draw_pixels (const char *const *data, int x, int y, Fl_Color bg=FL_GRAY)
  Draw XPM image data, with the top-left corner at the given position.

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

• void fl_end_complex_polygon ()
  End complex filled polygon, and draw.

• void fl_end_line ()
  End list of lines, and draw.

• void fl_end_loop ()
  End closed sequence of lines, and draw.

• void fl_end_offscreen ()
  Quit sending drawing commands to the current offscreen buffer.

• void fl_end_points ()
  End list of points, and draw.

• void fl_end_polygon ()
  End convex filled polygon, and draw.

• const char * fl_expand_text (const char *from, char *buf, int maxbuf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)
  Copy from to buf, replacing control characters with ^X.

• void fl_focus_rect (int x, int y, int w, int h)
  Draw a dotted rectangle, used to indicate keyboard focus on a widget.
• **Fl_Font fl_font ()**
  
  *Return the face set by the most recent call to fl_font().*

• **void fl_font (Fl_Font face, Fl_Fontsize fsize)**
  
  *Sets the current font, which is then used in various drawing routines.*

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

• **void fl_frame2 (const char *s, int x, int y, int w, int h)**
  
  *Draws a series of line segments around the given box.*

• **void fl_gap ()**
  
  *Separate loops of the path.*

• **int fl_height ()**
  
  *Return the recommended minimum line spacing for the current font.*

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

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

• **void fl_line (int x, int y, int x1, int y1)**
  
  *Draw a line from (x,y) to (x1,y1)*

• **void fl_line (int x, int y, int x1, int y1, int x2, int y2)**
  
  *Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)*

• **void fl_line_style (int style, int width=0, char *dashes=0)**
  
  *Set how to draw lines (the "pen").*

• **void fl_load_identity ()**
  
  *Set the transformation matrix to identity.*

• **void fl_not_clipped (int x, int y, int w, int h)**
  
  *Does the rectangle intersect the current clip region?*

• **unsigned int fl_old_shortcut (const char *s)**
  
  *Emulation of XForms named shortcuts.*

• **void fl_overlay_clear ()**
  
  *Erase a selection rectangle without drawing a new one.*

• **void fl_overlay_rect (int x, int y, int w, int h)**
  
  *Generated by Doxygen*
Draw a transient dotted selection rectangle.

• float **fl_override_scale** ()
  
  Removes any GUI scaling factor in subsequent drawing operations.

• void **fl_point** (int x, int y)
  
  Draw a single pixel at the given coordinates.

• void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2)
  
  Fill a 3-sided polygon.

• void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
  
  Fill a 4-sided polygon.

• void **fl_pop_clip** ()
  
  Restore the previous clip region.

• void **fl_pop_matrix** ()
  
  Restore the current transformation matrix from the stack.

• void **fl_push_clip** (int x, int y, int w, int h)
  
  Intersect the current clip region with a rectangle and push this new region onto the stack.

• void **fl_push_matrix** ()
  
  Save the current transformation matrix on the stack.

• void **fl_push_no_clip** ()
  
  Push an empty clip region onto the stack so nothing will be clipped.

• uchar ∗ **fl_read_image** (uchar ∗p, int X, int Y, int W, int H, int alpha=0)
  
  Reads an RGB(A) image from the current window or off-screen buffer.

• void **fl_rect** (Fl_Rect r)
  
  Draw a 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.

• void **fl_rescale_offscreen** (Fl_Offscreen &ctx)
  
  Adapts an offscreen buffer to a changed value of the scale factor.

• void **fl_reset_spot** (void)
  
  Resets marked text.

• void **fl_rotate** (double d)
  
  Concatenate rotation transformation onto the current one.
• void fl_rounded_rect (int x, int y, int w, int h, int r)
  *Draw a 1-pixel rounded border inside the given bounding box.*

• void fl_rounded_rectf (int x, int y, int w, int h, int r)
  *Color with current color a rounded rectangle that exactly fills the given bounding box.*

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

• void fl_scale (double x)
  *Concatenate scaling transformation onto the current one.*

• void fl_scale (double x, double y)
  *Concatenate scaling transformation onto the current one.*

• void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void (draw_area)(void *, int, int, int, int), void *data)
  *Scroll a rectangle and draw the newly exposed portions.*

• void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)
  *Inform text input methods about the current text insertion cursor.*

• void fl_set_status (int X, int Y, int W, int H)
  *Related to text input methods under X11.*

• const char * fl_shortcut_label (unsigned int shortcut)
  *Get a human-readable string from a shortcut value.*

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

• Fl_Fontsize fl_size ()
  *Return the size set by the most recent call to fl_font().*

• void fl_text_extents (const char *str, 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().*

• double fl_transform_dx (double x, double y)
  *Transform distance using current transformation matrix.*

• double fl_transform_dy (double x, double y)
  *Transform distance using current transformation matrix.*

• double fl_transform_x (double x, double y)
  *Transform coordinate using the current transformation matrix.*

• double fl_transform_y (double x, double y)
  *Transform coordinate using the current transformation matrix.*

• void fl_transformed_vertex (double xf, double yf)
  *Add coordinate pair to the vertex list without further transformations.*

• void fl_translate (double x, double y)
  *Concatenate translation transformation onto the current one.*

• void fl_vertex (double x, double y)
  *Add a single vertex to the current path.*

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

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

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

• void fl_xline (int x, int y, int x1)
  *Draw a horizontal line from (x,y) to (x1,y).*

• void fl_xline (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_xline (int x, int y, int x1, int y2, int x3)
Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).

- **void fl_yxline** (int x, int y, int y1)
  
  Draw a vertical line from (x,y) to (x,y1)

- **void fl_yxline** (int x, int y, int y1, int x2)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).

- **void fl_yxline** (int x, int y, int y1, int x2, int y3)
  
  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

**Variables**

- char fl_draw_shortcut

**34.40.1 Detailed Description**

utility header to pull drawing functions together

**34.41 fl_draw.H**

Go to the documentation of this file.

```c
#ifndef fl_draw_H
#define fl_draw_H

#include <FL/Enumerations.H> // color names
#include <FL/Fl_Graphics_Driver.H> // fl_graphics_driver + FlREGION
#include <FL/Fl_Rect.H>

// Image class...
class Fl_Image;
class Fl_Window;

// Label flags...
FL_EXPORT extern char fl_draw_shortcut;

// Colors:
inline void fl_color(Fl_Color c) {
  fl_graphics_driver->color(c);
} // select indexed color
inline void fl_color(int c) {
  fl_color((Fl_Color)c);
}
inline void fl_color(uchar r, uchar g, uchar b) {
  fl_graphics_driver->color(r, g, b);
}
inline Fl_Color fl_color() {
  return fl_graphics_driver->color();
}

// clip:
inline void fl_push_clip(int x, int y, int w, int h) {
  fl_graphics_driver->push_clip(x, y, w, h);
}
inline void fl_clip(int x, int y, int w, int h) {
  fl_graphics_driver->push_clip(x, y, w, h);
}
inline void fl_push_no_clip() {
  fl_graphics_driver->push_no_clip();
}
inline void fl_pop_clip() {
```
```c
inline int fl_not_clipped(int x, int y, int w, int h) {
    return fl_graphics_driver->not_clipped(x, y, w, h);
}

inline int fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H) {
    return fl_graphics_driver->clip_box(x, y, w, h, X, Y, W, H);
}

inline void fl_restore_clip() {
    fl_graphics_driver->restore_clip();
}

inline void fl_clip_region(Fl_Region r) {
    fl_graphics_driver->clip_region(r);
}

inline Fl_Region fl_clip_region() {
    return fl_graphics_driver->clip_region();
}

// points:
inline void fl_point(int x, int y) {
    fl_graphics_driver->point(x, y);
}

// line type:
inline void fl_line_style(int style, int width = 0, char *dashes = 0) {
    fl_graphics_driver->line_style(style, width, dashes);
}

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
};

inline void fl_antialias(int state) {
    fl_graphics_driver->antialias(state);
}

inline int fl_antialias() {
    return fl_graphics_driver->antialias();
}

// rectangles tweaked to exactly fill the pixel rectangle:
inline void fl_rect(int x, int y, int w, int h) {
    fl_graphics_driver->rect(x, y, w, h);
}

inline void fl_rounded_rect(int x, int y, int w, int h, int r) {
    fl_graphics_driver->rounded_rect(x, y, w, h, r);
}

inline void fl_rect(Fl_Rect r) {
    fl_rect(r.x(), r.y(), r.w(), r.h());
}

inline void fl_focus_rect(int x, int y, int w, int h) {
    fl_graphics_driver->focus_rect(x, y, w, h);
}

inline void fl_rect(int x, int y, int w, int h, Fl_Color c) {
    fl_color(c);
    fl_rect(x, y, w, h);
}

inline void fl_rectf(int x, int y, int w, int h) {
    fl_graphics_driver->rectf(x, y, w, h);
}

inline void fl_rounded_rectf(int x, int y, int w, int h, int r) {
    fl_graphics_driver->rounded_rectf(x, y, w, h, r);
}
```
inline void fl_rectf(int x, int y, int w, int h, Fl_Color c) {
    fl_color(c);
    fl_rectf(x, y, w, h);
}

inline void fl_rectf(Fl_Rect r) {
    fl_graphics_driver->rectf(r.x(), r.y(), r.w(), r.h());
}

inline void fl_rectf(Fl_Rect r, Fl_Color c) {
    fl_color(c);
    fl_rectf(r);
}

inline void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b) {
    fl_graphics_driver->colored_rectf(x, y, w, h, r, g, b);
}

inline void fl_rectf(Fl_Rect bb, uchar r, uchar g, uchar b) {
    fl_graphics_driver->colored_rectf(bb.x(), bb.y(), bb.w(), bb.h(), r, g, b);
}

// line segments:
inline void fl_line(int x, int y, int x1, int y1) {
    fl_graphics_driver->line(x, y, x1, y1);
}

inline void fl_line(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->line(x, y, x1, y1, x2, y2);
}

// closed line segments:
inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->loop(x, y, x1, y1, x2, y2);
}

inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
    fl_graphics_driver->loop(x, y, x1, y1, x2, y2, x3, y3);
}

// filled polygons
inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2) {
    fl_graphics_driver->polygon(x, y, x1, y1, x2, y2);
}

inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
    fl_graphics_driver->polygon(x, y, x1, y1, x2, y2, x3, y3);
}

// draw rectilinear lines, horizontal segment first:
inline void fl_xyline(int x, int y, int x1) {
    fl_graphics_driver->xyline(x, y, x1);
}

inline void fl_xyline(int x, int y, int x1, int y2) {
    fl_graphics_driver->xyline(x, y, x1, y2);
}

inline void fl_xyline(int x, int y, int x1, int y2, int x3) {
    fl_graphics_driver->xyline(x, y, x1, y2, x3);
}

// draw rectilinear lines, vertical segment first:
inline void fl_yxline(int x, int y, int y1) {
    fl_graphics_driver->yxline(x, y, y1);
}

inline void fl_yxline(int x, int y, int y1, int x2) {
    fl_graphics_driver->yxline(x, y, y1, x2);
}

inline void fl_yxline(int x, int y, int y1, int x2, int y3) {
    fl_graphics_driver->yxline(x, y, y1, x2, y3);
}

// circular lines and pie slices (code in fl_arci.C):
inline void fl_arc(int x, int y, int w, int h, double a1, double a2) {
    fl_graphics_driver->arc(x, y, w, h, a1, a2);
}

inline void fl_pie(int x, int y, int w, int h, double a1, double a2) {
    fl_graphics_driver->pie(x, y, w, h, a1, a2);
}

inline void fl_chord(int x, int y, int w, int h, double a1, double a2); // nyi

// scalable drawing code (code in fl_vertex.cxx and fl_arc.cxx):
inline void fl_push_matrix() {
    fl_graphics_driver->push_matrix();
}

inline void fl_pop_matrix() {
    fl_graphics_driver->pop_matrix();
}

inline void fl_scale(double x, double y) {
    fl_graphics_driver->mult_matrix(x, 0, 0, y, 0, 0);
}
inline void fl_scale(double x) {
    fl_graphics_driver->mult_matrix(x, 0, 0, x, 0, 0);
}

inline void fl_translate(double x, double y) {
    fl_graphics_driver->translate(x, y);
}

inline void fl_rotate(double d) {
    fl_graphics_driver->rotate(d);
}

inline void fl_load_identity() {
    fl_graphics_driver->load_identity();
}

inline void fl_load_matrix(double a, double b, double c, double d, double x, double y) {
    fl_graphics_driver->load_matrix(a, b, c, d, x, y);
}

inline void fl_mult_matrix(double a, double b, double c, double d, double x, double y) {
    fl_graphics_driver->mult_matrix(a, b, c, d, x, y);
}

inline void fl_begin_points() {
    fl_graphics_driver->begin_points();
}

inline void fl_begin_line() {
    fl_graphics_driver->begin_line();
}

inline void fl_begin_loop() {
    fl_graphics_driver->begin_loop();
}

inline void fl_begin_polygon() {
    fl_graphics_driver->begin_polygon();
}

inline void fl_vertex(double x, double y) {
    fl_graphics_driver->vertex(x, y);
}

inline void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3) {
    fl_graphics_driver->curve(X0, Y0, X1, Y1, X2, Y2, X3, Y3);
}

inline void fl_arc(double x, double y, double r, double start, double end) {
    fl_graphics_driver->arc(x, y, r, start, end);
}

inline void fl_circle(double x, double y, double r) {
    fl_graphics_driver->circle(x, y, r);
}

inline void fl_end_points() {
    fl_graphics_driver->end_points();
}

inline void fl_end_line() {
    fl_graphics_driver->end_line();
}

inline void fl_end_loop() {
    fl_graphics_driver->end_loop();
}

inline void fl_end_polygon() {
    fl_graphics_driver->end_polygon();
}

inline void fl_begin_complex_polygon() {
    fl_graphics_driver->begin_complex_polygon();
}

inline void fl_gap() {
    fl_graphics_driver->gap();
}

inline void fl_end_complex_polygon() {
    fl_graphics_driver->end_complex_polygon();
}

// get and use transformed positions:
inline double fl_transform_x(double x, double y) {
    return fl_graphics_driver->transform_x(x, y);
}

inline double fl_transform_y(double x, double y) {
    return fl_graphics_driver->transform_y(x, y);
}

inline double fl_transform_dx(double x, double y) {
    return fl_graphics_driver->transform_dx(x, y);
}

inline double fl_transform_dy(double x, double y) {
    return fl_graphics_driver->transform_dy(x, y);
}

inline void fl_transformed_vertex(double xf, double yf) {
    fl_graphics_driver->transformed_vertex(xf, yf);
}

inline void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy) {
    fl_graphics_driver->copy_offscreen(x, y, w, h, pixmap, srcx, srcy);
}

FL_EXPORT Fl_Offscreen fl_create_offscreen(int w, int h);
FL_EXPORT void fl_begin_offscreen(Fl_Offscreen b);
FL_EXPORT void fl_end_offscreen(void);
FL_EXPORT void fl_delete_offscreen(Fl_Offscreen bitmap);
FL_EXPORT void fl_rescale_offscreen(Fl_Offscreen &ctx);

/* NOTE: doxygen comments here to avoid triplication in os-specific sources */

// Fonts:
Set the current font, which is then used in various drawing routines.
Implemented and documented in src/fl_draw.cxx

FL_EXPORT void fl_font(Fl_Font face, Fl_Fontsize fsize);

inline Fl_Font fl_font() {
    return fl_graphics_driver->font();
}

inline Fl_Fontsize fl_size() {
    return fl_graphics_driver->size();
}

// Information you can get about the current font:

inline int fl_height() {
    return fl_graphics_driver->height();
}

inline int fl_height(int font, int size);

inline int fl_descent();

FL_EXPORT int fl_height(int font, int size);
FL_EXPORT int fl_descent(int font);

FL_EXPORT double fl_width(const char *txt);

inline double fl_width(const char *txt, int n) {
    return fl_graphics_driver->width(txt, n);
}

inline double fl_width(unsigned int c) {
    return fl_graphics_driver->width(c);
}

FL_EXPORT void fl_text_extents(const char *, int &dx, int &dy, int &w, int &h);

inline void fl_text_extents(const char *t, int n, int &dx, int &dy, int &w, int &h) {
    fl_graphics_driver->text_extents(t, n, dx, dy, w, h);
}

// font encoding:
Note: doxygen comments here to avoid duplication for os-specific cases

FL_EXPORT const char *fl_latin1_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_latin1(const char *t, int n = -1);
FL_EXPORT const char *fl_mac_roman_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_mac_roman(const char *t, int n = -1);
FLEXPORT float fl_override_scale();
FL_EXPORT void fl_restore_scale(float s);

FL_EXPORT void fl_draw(const char *str, int x, int y);
FL_EXPORT void fl_draw(int angle, const char *str, int x, int y);

inline void fl_draw(const char *str, int n, int x, int y) {
    fl_graphics_driver->draw(str, n, x, y);
}

inline void fl_draw(int angle, const char *str, int n, int x, int y) {
    fl_graphics_driver->draw(angle, str, n, x, y);
}

inline void fl_rtl_draw(const char *str, int n, int x, int y) {
    fl_graphics_driver->rtl_draw(str, n, x, y);
}

FL_EXPORT void fl_measure(const char *str, int &x, int &y, int draw_symbols = 1);
FL_EXPORT void fl_draw(const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img = 0,
int draw_symbols = 1);

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);

FL_EXPORT void fl_frame(const char *s, int x, int y, int w, int h);
FL_EXPORT void fl_frame2(const char *s, int x, int y, int w, int h);
FL_EXPORT void fl_draw_box(Fl_Boxtype, int x, int y, int w, int h, Fl_Color);

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Color col);

void fl_draw_check(Fl_Rect bb, Fl_Color col);

// arrows:

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Color col);

// Draw one or more "arrows" (triangles)

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color);

// Draw a potentially small, filled circle

FL_EXPORT void fl_draw_circle(int x0, int y0, int d, Fl_Color color);

// boxtypes:

FL_EXPORT void fl_draw_box(Fl_Boxtype, int x, int y, int w, int h, Fl_Color);

// basic GUI objects (check marks, arrows, more to come ...):

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Color col);

// Draw a check mark in the given color inside the bounding box bb.

FL_EXPORT void fl_draw_check(Fl_Rect bb, Fl_Color col);
inline void fl_draw_image(const uchar *buf, int X, int Y, int W, int H, int D = 3, int L = 0) {
    fl_graphics_driver->draw_image(buf, X, Y, W, H, D, L);
}

inline void fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D = 1, int L = 0) {
    fl_graphics_driver->draw_image_mono(buf, X, Y, W, H, D);
}

inline void fl_draw_image(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 3) {
    fl_graphics_driver->draw_image(cb, data, X, Y, W, H, D);
}

inline void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 1) {
    fl_graphics_driver->draw_image_mono(cb, data, X, Y, W, H, D);
}

inline char fl_can_do_alpha_blending() {
    return Fl_Graphics_Driver::default_driver().can_do_alpha_blending();
}

FL_EXPORT uchar *fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha = 0);
FL_EXPORT Fl_RGB_Image *fl_capture_window(Fl_Window *win, int x, int y, int w, int h);

FL_EXPORT int fl_draw_pixmap(const char *const *data, int x, int y, Fl_Color bg = FL_GRAY);
inline int fl_draw_pixmap(/*const*/ char *const *data, int x, int y, Fl_Color bg = FL_GRAY) {
    return fl_draw_pixmap((const char *const *)data, x, y, bg);
}

FL_EXPORT int fl_measure_pixmap(/*const*/ char *const *data, int &w, int &h);
FL_EXPORT int fl_measure_pixmap(const char *const *cdata, int &w, int &h);

// other:
FL_EXPORT void fl_scroll(int X, int Y, int W, int H, int dx, int dy,
    void (*draw_area)(void *, int, int, int, int), void *data);

// XIM:
FL_EXPORT void fl_set_status(int X, int Y, int W, int H);

// XForms symbols:
FL_EXPORT int fl_draw_symbol(const char *label, int x, int y, int w, int h, Fl_Color);
FL_EXPORT int fl_add_symbol(const char *name, void (*drawit)(Fl_Color), int scalable);

flendid;
# if defined(FL_DLL)
#define FL_LIBRARY
#define FL_EXPORT __declspec(dllexport)
#else
#define FL_EXPORT __declspec(dllimport)
#endif /* FL_LIBRARY */
#elif __GNUC__ >= 4
#define FL_EXPORT __attribute__ ((visibility("default")))
#else
#define FL_EXPORT
#endif /* FL_DLL */
#endif /* !Fl_Export_H */

4.43 Fl_File_Browser.H

1 /*
2 FileBrowser widget.
3 */
4 // Include necessary header files...
5
6 ifndef _Fl_File_Browser_H_
7 # define _Fl_File_Browser_H_
8
9 # include "Fl_Browser.H"
10 # include "Fl_File_Icon.H"
11 # include "filename.H"
12
13 // Fl_File_Browser class...
14
15 class FL_EXPORT Fl_File_Browser : public Fl_Browser {
16     
17     int filetype_,
18     uchar iconsize_,
19     const char *directory_,
20     const char *pattern_,
21     const char *errmsg_;
22
23     int full_height() const FL_OVERRIDE;
24     int item_height(void *) const FL_OVERRIDE;
25     int item_width(void *) const FL_OVERRIDE;
26     void item_draw(void *, int, int, int, int) const FL_OVERRIDE;
27
28     int incr_height() const FL_OVERRIDE { return (item_height(0)); }
29
30     public:
31         enum { FILES, DIRECTORIES };
32
33         Fl_File_Browser(int, int, int, int, const char * = 0);
34         ~Fl_File_Browser();
35
36         uchar iconsize() const { return (iconsize_); }
37         void iconsize(uchar s) { iconsize_ = s; redraw(); }
38
39         void filter(const char *pattern);
40
41         const char *filter() const { return (pattern_); }
42
43         int load(const char *directory, Fl_File_Sort_F *sort = fl_numericsort);
44
45         int filetype() const { return (filetype_); }
46         void filetype(int t) { filetype_ = t; }
47
48         const char *errmsg() const { return errmsg_; }
49         void errmsg(const char *emsg);
50
51 public:
52     enum { FILES, DIRECTORIES };
53     Fl_File_Browser(int, int, int, int, const char * = 0);
54     ~Fl_File_Browser();
55
56     uchar iconsize() const { return (iconsize_); }
57     void iconsize(uchar s) { iconsize_ = s; redraw(); }
58
59     void filter(const char *pattern);
60     const char *filter() const { return (pattern_); }
61
62     int load(const char *directory, Fl_File_Sort_F *sort = fl_numericsort);
63     int filetype() const { return (filetype_); }
64     void filetype(int t) { filetype_ = t; }
65     const char *errmsg() const { return errmsg_; }
66     void errmsg(const char *emsg);
67     const char *errmsg() const { return errmsg_; }
68
69     // Fl_File_Browser widget.
70 */
71 // FileBrowser definitions.
72 // Copyright 1999-2010 by Michael Sweet.
73 // This library is free software. Distribution and use rights are outlined in
74 // the file "COPYING" which should have been included with this file. If this
75 // file is missing or damaged, see the license at:
76 // https://www.fltk.org/COPYING.php
77 // Please see the following page on how to report bugs and issues:
78 // https://www.fltk.org/bugs.php
79
80
81 Generated by Doxygen
34.44 Fl_File_Chooser.H

```c
1 //
2 // Fl_File_Chooser dialog for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2021 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 // =======================================================================
17 // DO NOT EDIT FL/Fl_File_Chooser.H and src/Fl_File_Chooser.cxx !!!
18 // =======================================================================
19 // Please use fluid to change src/Fl_File_Chooser.fl interactively
20 // and then use fluid to "write code" or edit and use fluid -c .
21 // =======================================================================
22 //
23 // generated by Fast Light User Interface Designer (fluid) version 1.0400
24 //
25 #ifndef Fl_File_Chooser_H
26 #define Fl_File_Chooser_H
27 #include <FL/Fl.H>
28 #include <FL/Fl_Double_Window.H>
29 #include <FL/Fl_Group.H>
30 #include <FL/Fl_Choice.H>
31 #include <FL/Fl_Menu_Button.H>
32 #include <FL/Fl_Button.H>
33 #include <FL/Fl_Preferences.H>
34 #include <FL/Fl_Tile.H>
35 #include <FL/Fl_File_Browser.H>
36 #include <FL/Fl_Box.H>
37 #include <FL/Fl_Check_Button.H>
38 #include <FL/Fl_File_Input.H>
39 #include <FL/Fl_Return_Button.H>
40 #include <FL/fl_ask.H>
41 //
42 class FL_EXPORT Fl_File_Chooser {
43 public:
44    enum { SINGLE = 0, MULTI = 1, CREATE = 2, DIRECTORY = 4 };
45 private:
46    static Fl_Preferences *prefs_;
47    void (*callback_)(Fl_File_Chooser*, void*);
48    void *data_;  
49    char directory_[FL_PATH_MAX];
50    char pattern_[FL_PATH_MAX];
51    char preview_text_[2048];
52    int type_;
53    void favoritesButtonCB();
54    void favoritesCB(Fl_Widget *);
55    void fileListCB();
56    void fileNameCB();
57    void newdir();
58    static void previewCB(Fl_File_Chooser *fc);
59    void showChoiceCB();
60    void update_favorites();
61    void update_preview();
62    public:
63    Fl_File_Chooser(const char *, const char *, int, const char *title);
64 private:
65    Fl_Double_Window *window;
66     static void cb_window_i(Fl_Double_Window *, void*);
67     static void cb_window(Fl_Double_Window *, void*);
68     static Fl_Choice *showChoice;
69     static void cb_showChoice_i(Fl_Choice *, void*);
70     static void cb_showChoice(Fl_Choice *, void*);
71     Fl_Menu_Button *favoritesButton;
72     static void cb_favoritesButton_i(Fl_Menu_Button *, void*);
73     static void cb_favoritesButton(Fl_Menu_Button *, void*);
74 public:
75     Fl_Button *newButton;
76     static void cb__i(Fl_Tile *, void*);
77 private:
78     static void cb_newButton_i(Fl_Button *, void*);
79     static void cb_newButton(Fl_Button *, void*);
80     static void cb__i(Fl_Title *, void*);
```
static void cb_(Fl_Tile *, void*);
Fl_File_Browser *fileList;
inline void cb_fileList_i(Fl_File_Browser *, void*);
static void cb_fileList(Fl_File_Browser *, void*);
Fl_Box *errorBox;
Fl_Box *previewBox;
public:
Fl_Check_Button *previewButton;
private:
inline void cb_previewButton_i(Fl_Check_Button *, void*);
static void cb_previewButton(Fl_Check_Button *, void*);
public:
Fl_Check_Button *showHiddenButton;
private:
inline void cb_showHiddenButton_i(Fl_Check_Button *, void*);
static void cb_showHiddenButton(Fl_Check_Button *, void*);
Fl_File_Input *fileName;
inline void cb_fileName_i(Fl_File_Input *, void*);
static void cb_fileName(Fl_File_Input *, void*);
Fl_Return_Button *okButton;
inline void cb_okButton_i(Fl_Return_Button *, void*);
static void cb_okButton(Fl_Return_Button *, void*);
Fl_Button *cancelButton;
inline void cb_cancelButton_i(Fl_Button *, void*);
static void cb_cancelButton(Fl_Button *, void*);
Fl_Double_Window *favWindow;
Fl_File_Browser *favList;
inline void cb_favList_i(Fl_File_Browser *, void*);
static void cb_favList(Fl_File_Browser *, void*);
Fl_Button *favUpButton;
inline void cb_favUpButton_i(Fl_Button *, void*);
static void cb_favUpButton(Fl_Button *, void*);
Fl_Button * favDeleteButton;
inline void cb_favDeleteButton_i(Fl_Button *, void*);
static void cb_favDeleteButton(Fl_Button *, void*);
Fl_Button * favDownButton;
inline void cb_favDownButton_i(Fl_Button *, void*);
static void cb_favDownButton(Fl_Button *, void*);
Fl_Button * favCancelButton;
inline void cb_favCancelButton_i(Fl_Button *, void*);
static void cb_favCancelButton(Fl_Button *, void*);
Fl_Return_Button *favOkButton;
inline void cb_favOkButton_i(Fl_Return_Button *, void*);
static void cb_favOkButton(Fl_Return_Button *, void*);
public:
~Fl_File_Chooser();
void callback(void (*cb)(Fl_File_Chooser *, void *), void *d = 0);
void color(Fl_Color c);
Fl_Color color();
int count();
directory(const char *d);
directory();
void filter(const char *p);
const char * filter();
int filter_value();
void filter_value(int f);
void hide();
uchar iconsize();
void label(const char *l);
const char * label();
void ok_label(const char *l);
const char * ok_label();
void preview(int e);
int preview()const { return previewButton->value(); }
private:
void showHidden(int e);
void rescan();
void rescan_keep_filename();
void show();
int shown();
void textcolor(Fl_Color c);
Fl_Color textcolor();
void textfont(Fl_Font f);
Fl_Font textfont();
void textsize(Fl_Fontsize s);
Fl_Fontsize textsize();
void type(int t);
int type();
void * user_data() const;
void user_data(void *d);
const char * value(int f = 1);
void value(const char *filename);
int visible();
void position(int x, int y);
int x() const;
int y() const;
int w() const;
int h() const;
void size(int w, int h);
static const char *add_favorites_label;
static const char *all_files_label;
static const char *custom_filter_label;
static const char *existing_file_label;
static const char *favorites_label;
static const char *filename_label;
static const char *filesystems_label;
static const char *manage_favorites_label;
static const char *new_directory_label;
static const char *new_directory_tooltip;
static const char *preview_label;
static const char *save_label;
static const char *show_label;
static const char *hidden_label;
static Fl_File_Sort_F *sort;
private:
Fl_Widget * ext_group;
public:
Fl_Widget * add_extra(Fl_Widget * gr);
protected:
void show_error_box(int val);

FL_EXPORT char *fl_dir_chooser(const char *message,const char *fname,int relative=0);
FL_EXPORT char *fl_file_chooser(const char *message,const char *pat,const char *fname,int relative=0);
FL_EXPORT void fl_file_chooser_callback(void (*cb)(const char*));
FL_EXPORT void fl_file_chooser_ok_label(const char*l);
#endif

Fl_File_Icon.H

// Fl_File_Icon definitions.
// Copyright 1999-2010 by Michael Sweet.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php
//
// Include necessary header files...
//
#define _FL_Fl_File_Icon_H_
#define _FL_Fl_File_Icon_H_
#include "Fl.H"

// Special color value for the icon color.
#define FL_ICON_COLOR (Fl_Color)0xffffffff

// Fl_File_Icon class...

class FL_EXPORT Fl_File_Icon {
private:
static Fl_File_Icon *first_; // Pointer to first icon/filetype
static Fl_File_Icon *next_; // Pointer to next icon/filetype
const char *pattern_; // Pattern string
int type_; // Match only if directory or file?
int num_data_; // Number of data elements
int alloc_data_; // Number of allocated elements
short *data_; // Icon data
public:
enum // File types
{
    ANY, // Any kind of file
    PLAIN, // Only plain files
    FIFO, // Only named pipes
    DEVICE, // Only character and block devices
    LINK, // Only symbolic links
    DIRECTORY // Only directories
};

enum // Data opcodes
{
    END, // End of primitive/icon
    COLOR, // Followed by color value (2 shorts)
    LINE, // Start of line
    CLOSEDLINE, // Start of closed line
    POLYGON, // Start of polygon
    OUTLINEPOLYGON, // Followed by outline color (2 shorts)
    VERTEX // Followed by scaled X,Y
};

Fl_File_Icon(const char *p, int t, int nd = 0, short *d = 0);
~Fl_File_Icon();

short *add(short d);
short *add_color(Fl_Color c)
{ short *d = add((short)COLOR); add((short)(c >> 16)); add((short)c); return (d); }
short *add_vertex(int x, int y)
{ short *d = add((short)VERTEX); add((short)x); add((short)y); return (d); }
short *add_vertex(float x, float y)
{ short *d = add((short)VERTEX); add((short)(x * 10000.0));
    add((short)(y * 10000.0)); return (d); }
void clear() { num_data_ = 0; }
void draw(int x, int y, int w, int h, Fl_Color ic, int active = 1);
void label(Fl_Widget *w);
static void labeltype(const Fl_Label *l, int x, int y, int w, int h, Fl_Align a); void load(const char *f);
int load_fti(const char *fti);
int load_image(const char *i);
Fl_File_Icon *next() { return (next_); }
const char *pattern() { return (pattern_); }
int size() { return (num_data_); }
type() { return (type_); }
short *value() { return (data_); }
static Fl_File_Icon *find(const char *filename, int filetype = ANY);
static Fl_File_Icon *first() { return (first_); }
static void load_system_icons(void);
};

#endif // !_Fl_Fl_File_Icon_H_
# Fl_File_Input.H

```c
18 /* file 
19 Fl_File_Input widget. */
20
21 #ifndef Fl_File_Input_H
22 # define Fl_File_Input_H
23
24 # include <FL/Fl_Input.H>
25
45 class FL_EXPORT Fl_File_Input : public Fl_Input {
46        char ok_entry_; 
47        uchar down_box_; 
48        short buttons_[200]; 
49        short pressed_; 
50
52        void draw_buttons();
53        int handle_button(int event);
54        void update_buttons();
55
56        public:
57
58        Fl_File_Input(int X, int Y, int W, int H, const char *L=0);
59
60        int handle(int event) FL_OVERRIDE;
61
62        protected:
63        void draw() FL_OVERRIDE;
64
65        public:
67        Fl_Boxtype down_box()const { return (Fl_Boxtype)down_box_; }
69        void down_box(Fl_Boxtype b) { down_box_ = b; }
71
80        Fl_Color errorcolor()const { return FL_RED; }
81
90        void errorcolor(Fl_Color c) {(void)c;}
92
99        int value(const char *str);
100        int value(const char *str, int len);
101
102        const char *value() { return Fl_Input_::value(); }
103    }; 
104
105 #endif // !Fl_File_Input_H
```

# Fl_Fill_Dial.H

```c
34.47 Fl_Fill_Dial.H

1 //
2 // Filled dial header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
18 /* file 
19 Fl_Fill_Dial widget. */
20
21 #ifndef Fl_Fill_Dial_H
22 # define Fl_Fill_Dial_H
23
26 class FL_EXPORT Fl_Fill_Dial : public Fl_Dial {
27 public:
29        Fl_Fill_Dial(int X, int Y, int W, int H, const char *L);
30    }; 
31
32 #endif
```

# Fl_Fill_Slider.H

```c
34.48 Fl_Fill_Slider.H

1 //
2 // Filled slider header file for the Fast Light Tool Kit (FLTK).
3 //
```
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php

/*
file
Fl_Fill_Slider widget */

#ifndef Fl_Fill_Slider_H
#define Fl_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Fill_Slider : public Fl_Slider {
public:
    Fl_Fill_Slider(int X,int Y,int W,int H,const char *L=0);
};
#endif

34.49 Fl_Flex.H

1 //
2 // Fl_Flex widget header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 2020 by Karsten Federsen
5 // Copyright 2022 by Bill Spitzak and others.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:
14 //
15 // https://www.fltk.org/bugs.php
16 //
17 #ifndef Fl_Flex_H
18 #define Fl_Flex_H
19 #define Fl_Flex_H
20
21 #include <FL/Fl_Group.H>
22
23 class FL_EXPORT Fl_Flex : public Fl_Group {
24
25 int margin_left_;  
26 int margin_top_;  
27 int margin_right_;  
28 int margin_bottom_;  
29 int gap_;  
30 int fixed_size_size_;  
31 int fixed_size_alloc_;  
32 Fl_Widget **fixed_size_;  
33
34 public:  
35
36 enum { // values for type(int)  
37 VERTICAL = 0,  
38 HORIZONTAL = 1,  
39 COLUMN = 0,  
40 ROW = 1  
41 };  
42
43 // FLTK standard constructor  
44 Fl_Flex(int X, int Y, int W, int H, const char *L = 0);  
45
46 virtual ~Fl_Flex();  
47
48 // original Fl_Flex constructors:  
49 // backwards compatible if direction "names" | ROW | COLUMN | are used  
50 Fl_Flex(int direction);  
51 Fl_Flex(int W, int h, int direction);  
52 Fl_Flex(int x, int y, int w, int h, int direction);  
53 virtual ~Fl_Flex();  
54
55 virtual void end();  
56 void resize(int x, int y, int w, int h) FL_OVERRIDE;  

Generated by Doxygen
void fixed(Fl_Widget &w, int size) {
  fixed(w, size);
}

void fixed(Fl_Widget *w, int size);

int fixed(Fl_Widget *w) const;

protected:

void init(int t = VERTICAL);

virtual int alloc_size(int size) const;

void on_remove(int) FL_OVERRIDE;

public:

int margin() const { return margin_left_; }

int margin(int *left, int *top, int *right, int *bottom) const {
  if (left) *left = margin_left_;   
  if (top) *top = margin_top_;     
  if (right) *right = margin_right_; 
  if (bottom) *bottom = margin_bottom_; 
  return 1;
  return 0;
}

void margin(int m, int g = -1) {
  if (m < 0)
    m = 0;
  margin_left_ = margin_top_ = margin_right_ = margin_bottom_ = m;
  if (g >= 0)
    gap_ = g;
}

void margin(int left, int top, int right, int bottom) {
  margin_left_ = left < 0 ? 0 : left;
  margin_top_ = top < 0 ? 0 : top;
  margin_right_ = right < 0 ? 0 : right;
  margin_bottom_ = bottom < 0 ? 0 : bottom;
}

int gap() const {
  return gap_; 
}

void gap(int g) { 
  gap_ = g < 0 ? 0 : g; 
}

int horizontal() const { 
  return type() == Fl_Flex::HORIZONTAL ? 1 : 0; 
}

void layout() { 
  resize(x(), y(), w(), h()); 
  redraw(); 
}

int spacing() const { 
  return gap_; 
}

void spacing(int i) { 
  gap(i); 
}

34.50 Fl_Float_Input.H
/* Fl_Float_Input widget. */

#ifndef Fl_Float_Input_H
#define Fl_Float_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Float_Input : public Fl_Input {
public:
    Fl_Float_Input(int X, int Y, int W, int H, const char *l = 0);
};

#endif

Generated by Doxygen

34.51 Fl_FormsBitmap.H

/* Fl_FormsBitmap widget. */

#ifndef Fl_FormsBitmap_H
#define Fl_FormsBitmap_H

#include "Fl_Bitmap.H"

class FL_EXPORT Fl_FormsBitmap : public Fl_Widget {
    Fl_Bitmap *b;
protected:
    void draw() FL_OVERRIDE;
public:
    Fl_FormsBitmap(Fl_Boxtype, int, int, int, int, const char * = 0);
    void set(int W, int H, const uchar *bits);
    void bitmap(Fl_Bitmap *B) {b = B;}
    Fl_Bitmap *bitmap() const {return b;}
};

#endif

34.52 Fl_FormsPixmap.H

/* Fl_FormsPixmap widget. */
34.53 Fl_Free.H

1 //
2 // Forms free header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* Fl_Free widget. */
18
19 #ifndef Fl_Free_H
20 #define Fl_Free_H
21 #ifdef Fl_Widget_H
22 #endif
23 #ifndef Fl_Widget_H
24 #include "Fl_Widget.H"
25 #endif
26 #define FL_NORMAL_FREE 1
27 #define FL_SLEEPING_FREE 2
28 #define FL_INPUT_FREE 3
29 #define FL_CONTINUOUS_FREE 4
30 #define FL_ALL_FREE 5
31 typedef int (*FL_HANDLEPTR)(Fl_Widget *, int , float, float, char);
32
33 class FL_EXPORT Fl_Free : public Fl_Widget {
34 FL_HANDLEPTR hfunc;
35 static void step(void *);
36 protected:
37 void draw() FL_OVERRIDE;
38 public:
39 int handle(int e) FL_OVERRIDE;
40 Fl_Free(uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl);
41 ~Fl_Free();
42
43 // old event names for compatibility:
44 #define FL_MOUSE FL_DRAG
45 #define FL_DRAW 100
46 #define FL_STEP 101
47 #define FL_FREEMEM 102
48 #define FL_FREEZE 103
49 #define FL_THAW 104
50 #endif

34.54 Fl_GIF_Image.H

1 //
2 // GIF image header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
```cpp
class Fl_GIF_Image : public Fl_Pixmap {

public:

Fl_GIF_Image(const char * filename);
// deprecated constructor w/o length (for backwards compatibility)
Fl_GIF_Image(const char * imagename, const unsigned char *data);
// constructor with length (since 1.4.0)
Fl_GIF_Image(const char * imagename, const unsigned char *data, const size_t length);

static bool is_animated(const char *name_);
static bool animate;

protected:

Fl_GIF_Image(const char * filename, bool anim);
Fl_GIF_Image(const char * imagename, const unsigned char *data, const size_t length, bool anim);
// Protected default constructor needed for Fl_Anim_GIF_Image.
Fl_GIF_Image();

void load_gif_(class Fl_Image_Reader &rdr, bool anim=false);

void load(const char * filename, bool anim);
void load(const char * imagename, const unsigned char *data, const size_t length, bool anim);

// Internal structure to "glue" animated GIF support into Fl_GIF_Image.
// This data is passed during decoding to the Fl_Anim_GIF_Image class.
struct GIF_FRAME {
  int ifrm, width, height, x, y, w, h,
  clrs, bkgd, trans,
  dispose, delay;
  const uchar *bptr;
  const struct CPAL {
    uchar r, g, b;
  } *cpal;
  GIF_FRAME(int frame, uchar *data) : ifrm(frame), bptr(data) {};
  GIF_FRAME(int frame, int W, int H, int fx, int fy, int fw, int fh, uchar *data) :
    ifrm(frame), width(W), height(H), x(fx), y(fy), w(fw), h(fh), bptr(data) () {
  }
  void disposal(int mode, int delay) { dispose = mode; this->delay = delay; }
  void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
};

// Internal virtual methods, which are called during decoding to pass data
// to the Fl_Anim_GIF_Image class.
virtual void on_frame_data(GIF_FRAME &) {};
virtual void on_extension_data(GIF_FRAME &) {};

private:

void lzw_decode(Fl_Image_Reader &rdr, uchar *Image, int Width, int Height, int CodeSize, int ColorMapSize, int Interlace);
};
```

### 34.55 Fl_Gl_Window.H

```
// OpenGL header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2016 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
```
File Fl_Gl_Window.H
9 // https://www.fltk.org/COPYING.php
11 // Please see the following page on how to report bugs and issues:
14 // https://www.fltk.org/bugs.php
17 //
18 Fl_Gl_Window widget. */
20 #ifndef Fl_Gl_Window_H
21 #define Fl_Gl_Window_H
23 #include "Fl_Window.H"
25 class Fl_Gl_Choice; // structure to hold result of glXChooseVisual
26 class Fl_Gl_Window_Driver;
29 class FL_EXPORT Fl_Gl_Window : public Fl_Window
32 friend class Fl_Gl_Window_Driver;
35 Fl_Gl_Window_Driver *pGlWindowDriver;
38 int mode_; 
38 const int *alist;
40 Fl_Gl_Choice *g;
43 GLContext context_; 
47 char valid_f_; 
48 char damaged_f_; // damage() of back buffer
50 virtual void draw_overlay();
53 void init();
55 void *overlay;
58 static int can_do(int, const int *);
60 int mode(int, const int *);
62 static int gl_plugin_linkage();
64 protected:
65 void draw_begin();
67 void draw_overlay();
69 void draw_overlay();
71 public:
73 Fl_Gl_Window * as_gl_window() FL_OVERRIDE { return this; }
75 Fl_Gl_Window const * as_gl_window() const FL_OVERRIDE { return this; }
77 float pixels_per_unit();
79 int pixel_w() { return int(pixels_per_unit() * w() + 0.5f); }
81 int pixel_h() { return int(pixels_per_unit() * h() + 0.5f); }
84 ~Fl_Gl_Window();
87 Fl_Gl_Window(int W, int H, const char *l=0) : Fl_Window(W,H,l) {init();}
89 Fl_Gl_Window(int X, int Y, int W, int H, const char *l=0)
91 : Fl_Window(X,Y,W,H,l) {init();}
34.56 Fl_Graphics_Driver.H

A graphics driver can implement up to 6 virtual member functions to draw images:

- The 1st group of functions is used when the driver can directly map the image data, sized at data_w() x data_h(), to the image drawing area, sized at w() * scale x h() * scale factor.
- If the driver does not support such scale-and-draw operation for a given image type, it should implement the draw_fixed() function which is called by the library after the image has been internally resized to the drawing size and cached.
- Some drivers implement, for a given image class, only the function of the 1st group or only draw_fixed() as in these examples:

- Fl_Quartz_Graphics_Driver implements only draw_rgb(Fl_RGB_Image *,....) because it can perform the scale-and-draw operation whatever the RGB image and the required scaling.
34.56 Fl/Graphics_Driver.H 1267

119 - Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_Pixmap *, ...). The library
120 takes care of resizing and caching thePixmap to the adequate drawing size.
121 - Some drivers implement, for a given image class, the function of both groups, e.g. :
122 Fl_GDI_Graphics_Driver implements both draw_rgb(Fl_RGB_Image*, ...) and
draw_fixed(Fl_RGB_Image*, ...) because scale-and-draw may require function Alphablend()
123 from MSIMG32.DLL. In the absence of that, the draw_rgb() implementation calls
124 Fl_Graphics_Driver::draw_rgb() which runs Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image*, ...).
125
126 Graphics drivers also implement cache(Fl_Pixmap*), cache(Fl_Bitmap*) and cache(Fl_RGB_Image*)
127 to compute the cached form of all image types, and uncache(Fl_RGB_Image*, ...),
128 Fl_Pixmap, Fl_Bitmap and delete_bitmask(Fl_Bitmap*, ...) to destroy cached image forms.
129 Graphics drivers that use the mask_ variable of class Fl_Pixmap to cache an Fl_Pixmap
130 object also reimplement the uchar * Fl_Graphics_Driver::mask_bitmap() member function.
131 //

133 private:
134 virtual void draw_fixed(Fl_Pixmap *pxm, int XP, int YP, int WP, int HP, int cx, int cy);
135 virtual void draw_fixed(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy, int cf);
136 // the default implementation of make_unused_color() is most probably enough
137 virtual void make_unused_color(unsigned char &r, unsigned char &g, unsigned char &b, int color_count,
138 void **data);
139 // some platforms may need to reimplement this
140 virtual void set_current_();
141 float scale_; // scale between FLTK and drawing coordinates: drawing = FLTK * scale_
142 public:
144 static Fl_Graphics_Driver *newMainGraphicsDriver();
146 struct matrix {double a, b, c, d, x, y;};
148 typedef enum {
149 NATIVE = 1,
150 PRINTER = 2
151 } driver_feature;
152
153 protected:
154 int fl_clip_state_number;
155 static const matrix m0;
156 Fl_Font font_;
158 Fl_Color color_;
static void cache_w_h(Fl_RGB_Image *rgb, int*& pwidth, int*& pheight) {
  pwidth = &(rgb->cache_w_);
  pheight = &(rgb->cache_h_);
}

static void cache_size(Fl_Image *img, int &width, int &height) {
  width = img->width();
  height = img->height();
}

virtual void cache_size_finalize(Fl_Image *img, int &width, int &height);

Fl_Graphics_Driver();
virtual void cache_size(Fl_Image *img, int w, int h);
void cache_size_finalized(Fl_Image *img, int w, int h);

static unsigned need_pixmap_bg_color;
public:
virtual ~Fl_Graphics_Driver();
static Fl_Graphics_Driver &default_driver();

void push_matrix();
void pop_matrix();
void load_identity();
double double_matrix(double a, double b, double c, double d, double x, double y);
void mult_matrix(double a, double b, double c, double d, double x, double y);
void rotate(double d);
void translate(double x, double y);
void push_no_clip(); // has default implementation
虚拟 void pop_clip(); // has default implementation
virtual Fl_Region clip_region(); // has default implementation
virtual void clip_region(Fl_Region r); // has default implementation
virtual void restore_clip();
virtual void begin_points();
virtual void begin_line();
virtual void begin_loop();
virtual void begin_polygon();
virtual void begin_complex_polygon();
virtual void transformed_vertex(double xf, double yf);
virtual void transformed_vertex0(float x, float y);
virtual void vertex(double x, double y);
virtual void end_points();
virtual void end_line();
virtual void end_loop();
virtual void fixloop();
virtual void end_polygon();
virtual bool can_fill_non_convex_polygon() { return true; }
virtual void gap();
virtual void circle(double x, double y, double r);
virtual void arc(double x, double y, double r, double start, double end);
virtual void arc(int x, int y, int w, int h, double a1, double a2);
virtual void pie(int x, int y, int w, int h, double a1, double a2);
virtual void curve(double x0, double y0, double x1, double y1, double x2, double y2, double x3, double y3);
virtual void line_style(int style, int width=0, char * dashes=0);
321 virtual void color(Fl_Color c);
322 virtual void set_color(Fl_Color i, unsigned int c);
323 virtual void free_color(Fl_Color i, int overlay);
324 virtual Fl_Color color();
325 virtual void color(uchar r, uchar g, uchar b);
326 virtual void draw(const char *str, int nChars, int x, int y);
327 virtual void draw(const char *str, int nChars, float x, float y);
328 virtual void draw(int angle, const char *str, int nChars, int x, int y);
329 virtual void rtl_draw(const char *str, int nChars, int x, int y);
330 virtual int has_feature(driver_feature feature);
331 virtual void font(Fl_Font face, Fl_Fontsize fsize);
332 virtual Fl_Font font();
333 virtual Fl_Fontsize size();
334 virtual double width(const char *str, int nChars);
335 virtual double width(unsigned int c);
336 virtual void text_extents(const char *, int n, int &dx, int &dy, int &w, int &h);
337 virtual int height();
338 virtual int descent();
339 virtual void gc(void *);
340 virtual void *gc(void);
341 virtual uchar **mask_bitmap();
342 // default implementation may be enough
343 virtual float scale_font_for_PostScript(Fl_Font_Descriptor *desc, int s);
344 // default implementation may be enough
345 virtual float scale_bitmap_for_PostScript();
346 // each platform implements these 3 functions its own way
347 virtual void add_rectangle_to_region(Fl_Region r, int x, int y, int w, int h);
348 virtual Fl_Region &RectangleRegion(int x, int y, int w, int h);
349 virtual void XDestroyRegion(Fl_Region r);
350 virtual const char * get_font_name(Fl_Font fnum, int* ap);
351 virtual int get_font_sizes(Fl_Font fnum, int *& sizep);
352 virtual Fl_Font set_fonts(const char *name);
353 virtual Fl_Fontdesc * calc_fl_fonts(void);
354 virtual unsigned font_desc_size();
355 virtual const char *font_name(int num);
356 virtual void overlay_rect(int x, int y, int w, int h);
357 virtual float override_scale();
358 virtual void restore_scale(float);
359 virtual PangoFontDescription * pango_font_description() { return NULL; }
360 virtual void antialias(int state);
361 virtual int antialias();
362 virtual void delete_bitmask(fl_uintptr_t bm);
363 }
364
365 #ifndef FL_DOXYGEN
366
367 class Fl_Font_Descriptor {
368 public:
369 Fl_Font_Descriptor *next;
370 Fl_Fontsize size;
371 Fl_Font_Descriptor(const char * fontname, Fl_Fontsize size);
372 virtual FL_EXPORT ~Fl_Font_Descriptor() {};
373 short ascent, descent;
374 unsigned int listbase; // base of display list, 0 = none
375 }
376
377 // This class is not part of FLTK's public API.
378 struct Fl_Fontdesc {
379 const char *name;
380 Fl_Font_Descriptor *first; // linked list of these
381 Fl_Font_Descriptor(const char * fontname, Fl_Fontsize size);
382 }
383
384 class Fl_Font_Descriptor {
385 Fl_Font_Descriptor *next;
386 Fl_Fontsize size;
387 Fl_Font_Descriptor(const char * fontname, Fl_Fontsize size);
388 virtual FL_EXPORT ~Fl_Font_Descriptor() {};
389 short ascent, descent;
390 unsigned int listbase; // base of display list, 0 = none
391 }
392
393 // This class is not part of FLTK's public API.
394 class Fl_Fontdesc {
395 const char *name;
396 char fontname[128]; // "Pretty" font name
397 Fl_Font_Descriptor *first; // linked list of sizes of this style
398 }
399
400 // Abstract class Fl_Scalable_Graphics_Driver is platform-independent.
401 // It supports the scaling of all graphics coordinates by a
402 // float factor helpful to support HiDPI displays.
403 // This class does :
404 // - compute scaled coordinates
405 // - scale the cached offscreen of image objects
406 // - scale the pixel arrays used when performing direct image draws
407 // - call the member functions of a platform-specific
408 //   Fl_Scalable_Graphics_Driver-derived class that do the drawings with adequately
409 //   scaled coordinates. The member functions are named with the _unscaled suffix.
410 // - scale and unscale the clipping region.
411 // This class is presently used by the X11 and Windows platforms to support HiDPI displays.
412 // In the future, it may also be used by other platforms.
class FL_EXPORT Fl_Scalable_Graphics_Driver : public Fl_Graphics_Driver {
public:

  // This function aims to compute accurately \texttt{int}(x \times s) in
  // presence of rounding errors existing with floating point numbers
  // and that sometimes differ between 32 and 64 bits.
  static inline int floor(int x, float s) { return int(x * s + 0.001f); }

  inline int floor(int x) { return FL_Scalable_Graphics_Driver::floor(x, scale()); }

protected:

  int line_width;

  virtual Fl_Graphics_Driver::scale_clip(float f);

  void unscale_clip(Fl_Graphics_Driver::Region r);

  void point(int x, int y) FL_OVERRIDE;

  virtual void point_unscaled(float x, float y);

  void rect(int x, int y, int w, int h) FL_OVERRIDE;

  virtual void rectf_unscaled(float x, float y, float w, float h);

  void line(int x, int y, int x1, int y1) FL_OVERRIDE;

  virtual void line_unscaled(int x, int y, int x1, int y1);

  virtual void line_unscaled(int x, int y, int x1, int y1, int x2, int y2);

  void xline(int x, int y, int x1, int y1) FL_OVERRIDE;

  virtual void xline_unscaled(int x, int y, int x1, int y1);

  virtual void xline_unscaled(int x, int y, int x1, int y1, int x2, int y2);

  void yxline(int x, int y, int x1, int y1, int x2, int y1, int x3, int y3);

  virtual void yxline_unscaled(int x, int y, int x1, int y1, int x2, int y2);

  virtual void yxline_unscaled(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3);

  void yline(int x, int y, int x1, int y1, int x2, int y2) FL_OVERRIDE;

  virtual void yline_unscaled(int x, int y, int x1, int y1, int x2, int y2);

  virtual void yline_unscaled(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3);

  void loop(int x0, int y0, int x1, int y1, int x2, int y2) FL_OVERRIDE;

  virtual void loop_unscaled(int x0, int y0, int x1, int y1, int x2, int y2);

  virtual void polygon_unscaled(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);

  virtual void polygon_unscaled(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3, int L);

  virtual void arc_unscaled(int x, int y, int w, int h, double a1, double a2);

  void arc(double x, double y, double r, double start, double end) FL_OVERRIDE;

  virtual void ellipse_unscaled(double xt, double yt, double rx, double ry);

  void circle(double x, double y, double r) FL_OVERRIDE;

  virtual void ellipse_unscaled(double xt, double yt, double rx, double ry, double r);

  void font(Fl_Font face, Fl_Fontsize size) FL_OVERRIDE;

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);;

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);

  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);

  virtual void text_extents_unscaled(const char *str, int n, int &dx, int &dy, int &w, int &h);

  void text_extents(const char *str, int n, int &dx, int &dy, int &w, int &h) FL_OVERRIDE;

  virtual Fl_Fontsize size_unscaled();

  Fl_Fontsize size() FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image_unscaled(const uchar * buf, int X, int Y, int W, int H, int D=3, int L=0);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=3) FL_OVERRIDE;

  virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(mon Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(mon Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image_unscaled(const uchar * buf, int X, int Y, int W, int H, int D=3, int L=0);

  virtual void draw_image_unscaled(const uchar * buf, int X, int Y, int W, int H, int D=3, int L=0);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=3) FL_OVERRIDE;

  virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL_OVERRIDE;

  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1);

  virtual void draw_image(Fl_Draw_Image_Cb cb, void * data, int X, int Y, int W, int H, int D=1) FL.Override;

  float override_scale(FL_OVERRIDE);

  float restore_scale(float) FL_OVERRIDE;

};
virtual void change_pen_width(int lwidth);
virtual void reset_pen_width(void *data);
}
#endif // FL_DOXYGEN

#endif // FL_GRAPHICS_DRIVER_H

#include "Fl_Widget.H"

Classes

• class Fl_End
  This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

• class Fl_Group
  The Fl_Group class is the FLTK container widget.

34.57.1 Detailed Description

Fl_Group and Fl_End classes.

34.58 Fl_Group.H

Go to the documentation of this file.
void draw() FL_OVERRIDE;
void draw_child(Fl_Widget& widget) const;
void draw_children();
void draw_outside_label(const Fl_Widget& widget) const;
void update_child(Fl_Widget& widget) const;
Fl_Rect* bounds();
int* sizes(); // FLTK 1.3 compatibility
virtual int on_insert(Fl_Widget*, int);
virtual int on_move(int, int);
virtual void on_remove(int);

public:

int handle(int) FL_OVERRIDE;
void begin();
void end();
static Fl_Group* current();
static void current(Fl_Group* g);

int children() const {return children_;;
int find(const Fl_Widget*) const {return find(&o);}
Fl_Widget* const* array() const;

void resize(int, int, int, int) FL_OVERRIDE;
Fl_Group(int, int, int, int, const char* = 0);
virtual ~Fl_Group();
void add(Fl_Widget&);
void add(Fl_Widget* o) {add(*o);}
void insert(Fl_Widget*, int i);
void insert(Fl_Widget& o, Fl_Widget* before) {insert(o, find(before));}
void remove(int index);
void remove(Fl_Widget&);
void remove(Fl_Widget* o) {remove(o);}
void clear();

/* delete child n (by index) */
virtual int delete_child(int n);

void resizable(Fl_Widget& o) {resizable_ = &o;}
void resizable(Fl_Widget* o) {resizable_ = o;}
void resizable() const {return resizable_;;
void add_resizable(Fl_Widget& o) {resizable_ = &o; add(o);}
void init_sizes();

void clip_children(int c) {if (c) set_flag(CLIP_CHILDREN); else clear_flag(CLIP_CHILDREN);}
unsigned int clip_children() {return (flags() & CLIP_CHILDREN) != 0;}

// back compatibility functions:

void focus(Fl_Widget* W) {W->take_focus();}

// dummy class used to end child groups in constructors for complex
// subclasses of Fl_Group:

class FL_EXPORT Fl_End {
public:
Fl_End() {Fl_Group::current()->end();}

34.59 Fl_Help_Dialog.H

// copyright 1998-2021 by bill spitzak and others.

// this library is free software. distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. if this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

// please see the following page on how to report bugs and issues:
// Help Viewer widget definitions.
// Copyright 1997-2010 by Easy Software Products.
// Copyright 2011-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:

// generated by Fast Light User Interface Designer (fluid) version 1.0400

#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Help_View.H>

class FL_EXPORT Fl_Help_Dialog {
public:
    Fl_Help_Dialog();
    ~Fl_Help_Dialog();
    int h();
    void hide();
    int load(const char *f);
    void position(int xx, int yy);
    void resize(int xx, int yy, int ww, int hh);
    void show();
    void show(int argc, char **argv);
    void textsize(Fl_Fontsize s);
    Fl_Fontsize textsize();
    void topline(const char *n);
    void topline(int n);
    void value(const char *f);
    const char * value() const;
    int visible();
    int w();
    int x();
    int y();
};

#include <FL/Fl_Help_View.H>
/*
file
Fl_Help_View widget. */

#ifndef Fl_Help_View_H
#define Fl_Help_View_H

#endif

#include "Fl.H"
#include "Fl_Group.H"
#include "Fl_Scrollbar.H"
#include "fl_draw.H"
#include "filename.H"

class Fl_Shared_Image;

#ifndef Fl_Help_Func_H
#define Fl_Help_Func_H

typedef const char *(Fl_Help_Func)(Fl_Widget *, const char *);

#endif

class Fl_Help_Block {
const char *start, // Start of text
*end; // End of text
uchar border; // Draw border?
Fl_Color bgcolor; // Background color
int x, // Indentation/starting X coordinate
y, // Starting Y coordinate
w, // Width
h; // Height
int line[32]; // Left starting position for each line
int o1; // is ordered list <OL> element
int o1_num; // Item number in ordered list
};

struct Fl_Help_Link {
char filename[192],
name[32];
int x,
y,
w,
h;
};

/* Fl_Help_View font stack opaque implementation */

const size_t MAX_FL_HELP_FS_ELTS = 100;

struct FL_EXPORT Fl_Help_Font_Style {
Fl_Font f;
Fl_Fontsize s;
Fl_Color c;
void get(Fl_Font &afont, Fl_Fontsize &asize, Fl_Color &acolor) {afont=f; asize=s; acolor=c;}
void set(Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor) {f=afont; s=asize; c=acolor;}
Fl_Help_Font_Style(Fl_Font afont, Fl_Fontsize asize, Fl_Color acolor) {set(afont, asize, acolor);}
Fl_Help_Font_Style(){} // For in table use
};

const size_t MAX_FL_HELP_FS_ELTS = 100;

struct FL_EXPORT Fl_Help_Font_Stack {
Fl_Help_Font_Style fs[100]; // Font stack array
int nfonts_; // Number of fonts
}

void init(Fl_Font f, Fl_Fontsize s, Fl_Color c) {
fs[nfonts_].set(f, s, c);
}

Generated by Doxygen
void top(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) { elts_[nfonts_].get(f, s, c); }

void push(Fl_Font f, Fl_Fontsize s, Fl_Color c) {
    if (nfonts_ < MAX_FL_HELP_FS_ELTS-1) nfonts_ ++;
    elts_[nfonts_].set(f, s, c);
    fstack_.push(f, s, c);
}

void pop(Fl_Font &f, Fl_Fontsize &s, Fl_Color &c) {
    if (nfonts_ > 0) nfonts_ --;
    top(f, s, c);
    fstack_.pop(f, s, c);
}

size_t count()const {return nfonts_;} // Gets the current number of fonts in the stack

protected:
size_t nfonts_; // Fl_Help_Font_Style elts_[MAX_FL_HELP_FS_ELTS];

struct Fl_Help_Target {
    char name[32];
    int y;
};

class FL_EXPORT Fl_Help_View : public Fl_Group { // Help viewer widget
    enum { RIGHT = -1, CENTER, LEFT }; // Fl_Help_Font_Style elts_[MAX_FL_HELP_FS_ELTS];
}ENSIONS;

Fl_Help_Block *add_block(const char *s, int xx, int yy, int ww, int hh, uchar border = 0);
void add_link(const char *n, int xx, int yy, int ww, int hh);
void add_target(const char *n, int yy);
static int compare_targets(const Fl_Help_Target *t0, const Fl_Help_Target *t1);

Generated by Doxygen
```cpp
int do_align(Fl_Help_Block *block, int line, int xx, int a, int &l);
protected:
void draw() FL_OVERRIDE;
private:
void format();
void format_table(int *table_width, int *columns, const char *table);
void free_data();
int get_align(const char *p, int a);
const char *get_attr(const char *p, const char *n, char *buf, int bufsize);
Fl_Color get_color(const char *n, Fl_Color c);
Fl_Shared_Image *get_image(const char *name, int W, int H);
int get_length(const char *s);
public:
int handle(int) FL_OVERRIDE;
private:

void hv_draw(const char *t, int x, int y, int entity_extra_length = 0);
char begin_selection();
char extend_selection();
void end_selection(int c = 0);
void clear_global_selection();
Fl_Help_Link *find_link(int, int);
void follow_link(Fl_Help_Link *);

public:
Fl_Help_View(int xx, int yy, int ww, int hh, const char *l = 0);
~Fl_Help_View();
const char *directory() const { if (directory_[0]) return (directory_); else return ((const char *)0); }
const char *filename() const { if (filename_[0]) return (filename_); else return ((const char *)0); }
int find(const char *s, int p = 0);
void link(Fl_Help_Func *fn) { link_ = fn; }
int load(const char *f);
void resize(int,int,int,int) FL_OVERRIDE;
int size() const { return (size_); }
void size(int W, int H) { Fl_Widget::size(W, H); }
void textcolor(Fl_Color c) { if (textcolor_ == defcolor_) textcolor_ = c; defcolor_ = c; }
Fl_Color textcolor() const { return (defcolor_); }
void textfont(Fl_Font f) { textfont_ = f; format(); }
Fl_Font textfont() const { return (textfont_); }
void textsize(Fl_Fontsize s) { textsize_ = s; format(); }
Fl_Fontsize textsize() const { return (textsize_); }
const char *title() { return (title_); }
topline(const char *n);
topline(int);
topline(const char *); [return (topline_);]
leftline(int);
leftline(const char *); [return (leftline_);]
value(const char *val);
value(const char *); [return (value_);]
void clear_selection();
void select_all();
int scrollbar_size() const { return scrollbar_size_; }
void scrollbar_size(int newSize) { scrollbar_size_ = newSize; }
}
Fl_Hold_Browser.H
```

### 34.6.1 Fl_Hold_Browser.H

1 //
2 // Hold browser header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 // Fl_Hold_Browser widget . */
18
34.62 Fl_Hor_Fill_Slider.H

/*
file
Fl_Hor_Fill_Slider widget . */

#ifndef Fl_Hor_Fill_Slider_H
#define Fl_Hor_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Hor_Fill_Slider : public Fl_Slider {
public:
    Fl_Hor_Fill_Slider(int X, int Y, int W, int H, const char *L=0);
};

#endif

34.63 Fl_Hor_Nice_Slider.H

/*
file
Fl_Hor_Nice_Slider widget . */

#ifndef Fl_Hor_Nice_Slider_H
#define Fl_Hor_Nice_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Hor_Nice_Slider : public Fl_Slider {
public:
    Fl_Hor_Nice_Slider(int X, int Y, int W, int H, const char *L=0);
};

#endif
34.64  Fl_Hor_Slider.H

1  //
2  // Horizontal slider header file for the Fast Light Tool Kit (FLTK).
3  //
4  // Copyright 1998-2011 by Bill Spitzak and others.
5  //
6  // This library is free software. Distribution and use rights are outlined in
7  // the file "COPYING" which should have been included with this file. If this
8  // file is missing or damaged, see the license at:
9  //
10  //   https://www.fltk.org/COPYING.php
11  //
12  // Please see the following page on how to report bugs and issues:
13  //
14  //   https://www.fltk.org/bugs.php
15  //
16  
17  /*
18  \file
19  Fl_Hor_Slider widget . */
20  ifndef Fl_Hor_Slider_H
21  #define Fl_Hor_Slider_H
22  
23  #include "Fl_Slider.H"
24  
25  class FL_EXPORT Fl_Hor_Slider : public Fl_Slider {
26     public:
27     Fl_Hor_Slider(int X,int Y,int W,int H,const char *l=0);
28  }
29  
30  #endif

34.65  Fl_Hor_Value_Slider.H

1  //
2  // Horizontal value slider header file for the Fast Light Tool Kit (FLTK).
3  //
4  // Copyright 1998-2010 by Bill Spitzak and others.
5  //
6  // This library is free software. Distribution and use rights are outlined in
7  // the file "COPYING" which should have been included with this file. If this
8  // file is missing or damaged, see the license at:
9  //
10  //   https://www.fltk.org/COPYING.php
11  //
12  // Please see the following page on how to report bugs and issues:
13  //
14  //   https://www.fltk.org/bugs.php
15  //
16  
17  /*
18  \file
19  Fl_Hor_Value_Slider widget . */
20  ifndef Fl_Hor_Value_Slider_H
21  #define Fl_Hor_Value_Slider_H
22  
23  #include "Fl_Value_Slider.H"
24  
25  class FL_EXPORT Fl_Hor_Value_Slider : public Fl_Value_Slider {
26     public:
27     Fl_Hor_Value_Slider(int X,int Y,int W,int H,const char *l=0);
28  }
29  
30  #endif

34.66  FlICOImage.H

1  //
2  // ICO image header file for the Fast Light Tool Kit (FLTK).
3  //
4  // Copyright 2022-2023 by Bill Spitzak and others.
5  //
6  // This library is free software. Distribution and use rights are outlined in
7  // the file "COPYING" which should have been included with this file. If this
8  // file is missing or damaged, see the license at:
9  //
10  //   http://www.fltk.org/COPYING.php
11  //
12  // Please report all bugs and problems on the following page:
13  //
```cpp
#define Fl_ICO_Image_H
#include "Fl_BMP_Image.H"

class FL_EXPORT Fl_ICO_Image : public Fl_BMP_Image {

public:
  struct IconDirEntry {
    int bWidth;
    int bHeight;
    int bColorCount;
    int bReserved;
    int wPlanes;
    int wBitCount;
    int dwBytesInRes;
    int dwImageOffset;
  };

  Fl_ICO_Image(const char *filename, int id = -1, const unsigned char *data = NULL, const size_t datasize = 0);
  ~Fl_ICO_Image();

  int idcount() const { return idcount_; }

  const IconDirEntry * icondirentry() const { return icondirentry_; }

private:
  int idcount_;  
  struct IconDirEntry *icondirentry_;  

  void load_ico_(class Fl_Image_Reader &rdr, int id);
};
#endif // Fl_ICO_Image_H
```

### 34.67 Fl_Image.H File Reference

**Fl_Image, Fl_RGB_Image** classes.

```cpp
#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**
  
  The scaling algorithm to use for RGB images.

#### 34.67.1 Detailed Description

**Fl_Image, Fl_RGB_Image** classes.

#### 34.67.2 Enumeration Type Documentation

Generated by Doxygen
### 34.67.2.1 FL_RGB_Scaling

**enum FL_RGB_Scaling**  
The scaling algorithm to use for RGB images.

**Enumerator**

<table>
<thead>
<tr>
<th>FL_RGB_SCALING_NEAREST</th>
<th>default RGB image scaling algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_RGB_SCALING_BILINEAR</td>
<td>more accurate, but slower RGB image scaling algorithm</td>
</tr>
</tbody>
</table>

### 34.68 Fl_Image.H

**Go to the documentation of this file.**

```
1 // 2 // Image header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 #ifndef Fl_Image_H
17 #define Fl_Image_H
18
19 #include "Enumerations.H"
20 #include "Fl_Widget.H" // for fl_uintptr_t
21
22 class Fl_Widget;
23 class Fl_Pixmap;
24 struct Fl_Menu_Item;
25 struct Fl_Label;
26 class Fl_RGB_Image;
27
28 enum Fl_RGB_Scaling {
29 FL_RGB_SCALING_NEAREST = 0,
30 FL_RGB_SCALING_BILINEAR
31);
32
33 class Fl_EXPORT Fl_Image {
34     // Presently redefined in Fl_SVG_Image
35 public:
36     static const int ERR_NO_IMAGE = -1;
37     static const int ERR_FILE_ACCESS = -2;
38     static const int ERR_FORMAT = -3;
39     static const int ERR_MEMORY_ACCESS = -4;
40     
41     private:
42     int w, h, d, ld, count;
43     int data_w, data_h;
44     
45     const char * const *data;
46     static Fl_RGB_Scaling RGB_scaling; // method used when copying RGB images
47     static Fl_RGB_Scaling scaling_algorithm; // method used to rescale RGB source images before drawing
48     
49     virtual void cache_size_(int & /*width*/, int & /*height*/); {}
50     
51 protected:
52     
53     void w(int W) {w_ = W; data_w_ = W;}
54     void h(int H) {h_ = H; data_h_ = H;}
55     void d(int D) {d_ = D;}
56     void data(const char * const *p, int c) {data_ = p; count_ = c;}
57     void draw_empty(int X, int Y);
58     
59     static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la);
```
static void measure(const Fl_Label *lo, int &lw, int &lh);
int draw_scaled(int X, int Y, int W, int H);

public:

int w()const {return w_;
int h()const {return h_;
int data_w()const {return data_w_;
int data_h()const {return data_h_;
int d()const {return d_;
int ld()const {return ld_;
int count()const {return count_;
const char * const *data()const {return data_;
fail() const;
virtual void release() {
 delete this;
}

virtual class Fl_Shared_Image *as_shared_image() {
 return 0;
}

virtual Fl_Image(int W, int H, int D);
virtual ~Fl_Image();
copy(int W, int H) const;
virtual void color_average(Fl_Color c, float i);
void inactive() {color_average(FL_GRAY, .33f);}
virtual void desaturate();
virtual void label(Fl_Widget *w);
virtual void label(Fl_Menu_Item *m);
virtual void draw(int X, int Y, int W, int H, int cx=0, int cy=0);
virtual void uncache();

// used by fl_define_FL_IMAGE_LABEL() to avoid 'friend' declaration
static Fl_Labeltype define_FL_IMAGE_LABEL();

// set RGB image scaling method
static void RGB_scaling(Fl_RGB_Scaling);
static Fl_RGB_Scaling RGB_scaling();

// set the image drawing size
virtual void scale(int width, int height, int proportional = 1, int can_expand = 0);
static void scaling_algorithm(Fl_RGB_Scaling algorithm) {scaling_algorithm_ = algorithm;}
static Fl_RGB_Scaling scaling_algorithm();

static void max_size(size_t size) {max_size_ = size;}
static size_t max_size() {return max_size_;
virtual Fl_SVG_Image *as_svg_image() { return NULL;}
virtual void normalize() {}
# FlImage_Surface.H

```cpp
#ifndef Fl_Image_Surface_H
#define Fl_Image_Surface_H

#include <FL/Fl_Widget_Surface.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/platform_types.h> // for Fl_Offscreen

class FL_EXPORT Fl_Image_Surface : public Fl_Widget_Surface {
    friend class Fl_Graphics_Driver;

private:
    class Fl_Image_Surface_Driver *platform_surface;
    Fl_Offscreen get_offscreen_before_delete_();

protected:
    void translate(int x, int y) FL_OVERRIDE;
    void untranslate() FL_OVERRIDE;

public:
    Fl_Image_Surface(int w, int h, int high_res = 0, Fl_Offscreen off = 0);
    ~Fl_Image_Surface();
    void set_current() FL_OVERRIDE;
    bool is_current() FL_OVERRIDE;
    Fl_RGB_Image *image();
    Fl_Shared_Image *highres_image();
    void origin(int *x, int *y) FL_OVERRIDE;
    void origin(int x, int y) FL_OVERRIDE;
    int printable_rect(int *w, int *h) FL_OVERRIDE;
    Fl_Offscreen offscreen();
    void rescale();
};

#endif // Fl_Image_Surface_H
```

# Fl_Input.H

```cpp
#ifndef Fl_Input_H
#define Fl_Input_H

#include <FL/Fl_Widget_Surface.H>
#include <FL/Fl_Image_Surface.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/platform_types.h> // for Fl_Offscreen

class FL_EXPORT Fl_Image_Surface : public Fl_Widget_Surface {
    friend class Fl_Graphics_Driver;

private:
    class Fl_Image_Surface_Driver *platform_surface;
    Fl_Offscreen get_offscreen_before_delete_();

protected:
    void translate(int x, int y) FL_OVERRIDE;
    void untranslate() FL_OVERRIDE;

public:
    Fl_Image_Surface(int w, int h, int high_res = 0, Fl_Offscreen off = 0);
    ~Fl_Image_Surface();
    void set_current() FL_OVERRIDE;
    bool is_current() FL_OVERRIDE;
    Fl_RGB_Image *image();
    Fl_Shared_Image *highres_image();
    void origin(int *x, int *y) FL_OVERRIDE;
    void origin(int x, int y) FL_OVERRIDE;
    int printable_rect(int *w, int *h) FL_OVERRIDE;
    Fl_Offscreen offscreen();
    void rescale();
};

#endif // Fl_Input_H
```
class FL_EXPORT Fl_Input : public Fl_Input_ {

friend class Fl_Screen_Driver;
friend class Fl_Cocoa_Screen_Driver; // Not ideal, but probably no other platform will use it
int shift_position(int p);
int shift_up_down_position(int p);

void handle_mouse(int keepmark=0);

// Private keyboard functions
int kf_lines_up(int repeat_num);
int kf_lines_down(int repeat_num);
int kf_page_up();
int kf_page_down();
int kf_insert_toggle();
int kf_delete_word_right();
int kf_delete_word_left();
int kf_delete_sol();
int kf_delete_eol();
int kf_delete_char_right();
int kf_delete_char_left();
int kf_delete_char();
int kf_delete_char_right();
int kf_delete_char_left();
int kf_clear_eol();
int kf_move_sol();
int kf_move_eol();
int kf_move_char_left();
int kf_move_char_right();
int kf_move_word_left();
int kf_move_word_right();
int kf_move_up_and_sol();
int kf_move_down_and_eol();
int kf_top();
int kf_bottom();
int kf_select_all();
int kf_undo();
int kf_redo();
int kf_copy();
int kf_paste();
int kf_copy_cut();

protected:
void draw() FL_OVERRIDE;
int handle_key();

public:
int handle(int) FL_OVERRIDE;
int Fl_Input(int, int, int, const char * = 0);
};
#endif

Generated by Doxygen

34.71 Fl_Input_.H
1 //
2 // Input base class header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2015 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* \file
18 Fl_Input widget . */
19
20 ifndef Fl_Input_
21 define Fl_Input_
22
23 include "Fl_Input_.H"
24
25 class FL_EXPORT Fl_Input : public Fl_Input_ {
26
friend class Fl_Screen_Driver;
friend class Fl_Cocoa_Screen_Driver; // Not ideal, but probably no other platform will use it
27
int shift_position(int p);
28
void handle_mouse(int keepmark=0);
29
// Private keyboard functions
30 int kf_lines_up(int repeat_num);
31 int kf_lines_down(int repeat_num);
32 int kf_page_up();
33 int kf_page_down();
34 int kf_insert_toggle();
35 int kf_delete_word_right();
36 int kf_delete_word_left();
37 int kf_delete_sol();
38 int kf_delete_eol();
39 int kf_delete_char_right();
40 int kf_delete_char_left();
41 int kf_delete_char();
42 int kf_delete_char_right();
43 int kf_delete_char_left();
44 int kf_clear_eol();
45 int kf_move_sol();
46 int kf_move_eol();
47 int kf_move_char_left();
48 int kf_move_char_right();
49 int kf_move_word_left();
50 int kf_move_word_right();
51 int kf_move_up_and_sol();
52 int kf_move_down_and_eol();
53 int kf_top();
54 int kf_bottom();
55 int kf_select_all();
56 int kf_undo();
57 int kf_redo();
58 int kf_copy();
59 int kf_paste();
60 int kf_copy_cut();
61
protected:
62 void draw() FL_OVERRIDE;
63 int handle_key();
64
public:
65 int handle(int) FL_OVERRIDE;
66 Fl_Input(int, int, int, const char * = 0);
67
};
68
#endif
34.71 Fl_Input_.H
1 //
2 // Input base class header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2015 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* \file
18 Fl_Input_ widget . */
19
20 ifndef Fl_Input_
21 define Fl_Input_
22
23 include "Fl_Input_.H"
24
25 class FL_EXPORT Fl_Input : public Fl_Input_ {
26
friend class Fl_Screen_Driver;
friend class Fl_Cocoa_Screen_Driver; // Not ideal, but probably no other platform will use it
27
int shift_position(int p);
28
void handle_mouse(int keepmark=0);
29
// Private keyboard functions
30 int kf_lines_up(int repeat_num);
31 int kf_lines_down(int repeat_num);
32 int kf_page_up();
33 int kf_page_down();
34 int kf_insert_toggle();
35 int kf_delete_word_right();
36 int kf_delete_word_left();
37 int kf_delete_sol();
38 int kf_delete_eol();
39 int kf_delete_char_right();
40 int kf_delete_char_left();
41 int kf_delete_char();
42 int kf_delete_char_right();
43 int kf_delete_char_left();
44 int kf_clear_eol();
45 int kf_move_sol();
46 int kf_move_eol();
47 int kf_move_char_left();
48 int kf_move_char_right();
49 int kf_move_word_left();
50 int kf_move_word_right();
51 int kf_move_up_and_sol();
52 int kf_move_down_and_eol();
53 int kf_top();
54 int kf_bottom();
55 int kf_select_all();
56 int kf_undo();
57 int kf_redo();
58 int kf_copy();
59 int kf_paste();
60 int kf_copy_cut();
61
protected:
62 void draw() FL_OVERRIDE;
63 int handle_key();
64
public:
65 int handle(int) FL_OVERRIDE;
66 Fl_Input(int, int, int, const char * = 0);
67
};
68
#endif
```c
#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

#define FL_NORMAL_INPUT 0
#define FL_FLOAT_INPUT 1
#define FL_INT_INPUT 2
#define FL_HIDDEN_INPUT 3
#define FL_MULTILINE_INPUT 4
#define FL_SECRET_INPUT 5
#define FL_INPUT_TYPE 7
#define FL_INPUT_READONLY 8
#define FL_NORMAL_OUTPUT (FL_NORMAL_INPUT | FL_INPUT_READONLY)
#define FL_MULTILINE_OUTPUT (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
#define FL_INPUT_WRAP 16
#define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)
#define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)

class Fl_Input_Undo_Action;
class Fl_Input_Undo_Action_List;

class FL_EXPORT Fl_Input_ : public Fl_Widget {
  const char * value_;
  char * buffer;
  int size_;
  int bufsize;
  int position_;  // Mark a range of characters for update.
  int mark_;  // Mark a range of characters for update.
  int tab_nav_;  // Copy the value from a possibly static entry into the internal buffer.
  int bufsize;
  int maximum_size_;  // Set the current font and font size.
  int shortcut_;  // Find the start of a word.
  uchar erase_cursor_only;
  Fl_Font textfont_;  // Calculates the width in pixels of part of a text buffer.
  Fl_Fontsize textsize_;  // Set the current font and font size.
  Fl_Color textcolor_;  // Set the current font and font size.
  Fl_Color cursor_color_;  // Set the current font and font size.
  Fl_Input_Undo_Action * undo_;  // Set the current font and font size.
  Fl_Input_Undo_Action_List * undo_list_;  // Set the current font and font size.
  Fl_Input_Undo_Action_List * redo_list_;  // Set the current font and font size.
  static double up_down_pos;
  static int was_up_down;

  // Convert a given text segment into the text that will be rendered on screen.
  const char * expand(const char*, char*) const;
  // Calculates the width in pixels of part of a text buffer.
  double expandpos(const char*, const char*, const char*, int*) const;
  // Mark a range of characters for update.
  void minimal_update(int, int);
  // Mark a range of characters for update.
  void minimal_update(int);
  // Copy the value from a possibly static entry into the internal buffer.
  void put_in_buffer(int newsize);
  // Set the current font and font size.
  void setfont() const;

protected:

  // Find the start of a word.
  int word_start(int i) const;

};
``
int word_end(int i) const;
int line_start(int i) const;
int line_end(int i) const;
int up_down_position(int, int keepmark=0);
int handle_text(int, int, int, int, int, int, int);
int maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_UNKNOWN);
int xscroll() const {return xscroll_;}
int yscroll() const {return yscroll_;
void yscroll(int yOffset) { yscroll_ = yOffset; damage(FL_DAMAGE_EXPOSE);}
int linesPerPage();
int apply_undo();
void resize(int, int, int, int) FL_OVERRIDE;
~Fl_Input_();
int value(const char *);
int value(const char *, int);
int static_value(const char *);
int static_value(const char *, int);
const char * value() const {return value_;
unsigned int index(int i) const;
int size() const {return size_;
int maximum_size() const {return maximum_size_;}
int insert_position() const { return position_; }
int mark() const { return mark_; }
int insert_position(int p, int m);
int position(int p, int m); }
int position(int p) { return insert_position(p); }
int mark(int m) { return insert_position(insert_position(), m); }
/* Deletes text from \p b to \p e and inserts the new string \text. */
int replace(int b, int e, const char *text, int ilen=0);
int cut() { return replace(insert_position(), mark(), 0); }
int cut(int n) { return replace(insert_position(), insert_position()+n, 0); }
int cut(int a, int b) { return replace(a, b, 0); }
/* Append text at the end. */
int append(const char *t, int l=0, char keep_selection=0);
/* Put the current selection into the clipboard. */
int copy(int clipboard);
/* Undo previous changes to the text buffer. */
int undo();
/* Return true if the last operation can be undone. */
bool can_undo() const;
/* Redo previous undo operations. */
int redo();
/* Return true if there is a redo action in the list. */
bool can_redo() const;
/* Copy the yank buffer to the clipboard. */
int copy_cuts();
void shortcut(int s) { shortcut_ = s; }
Fnt font(const char *t, int l=0) { return textfont_; }
Fnt font(Fl_Font s) { textfont_ = s; }
Fntsize fontsize(const char *t, int l=0) { textsize_ = s; }
Fnt cursor_color(const char *t) { return textcolor_; }
vcolor cursor_color() const { return cursor_color_; }
int input_type() const { return type() & FL_INPUT_TYPE; }
void input_type(int t) { type((uchar)(t | readonly())); }
int readonly() const { return type() & FL_INPUT_READONLY; }
void readonly(int b) { if (b) type((uchar)(t | readonly())); }
int wrap() const { return type() & FL_INPUT_WRAP; }
void wrap(int b) { if (b) type((uchar)(t | FL_INPUT_WRAP)); }
void tab_nav(int val) { if (val) return tab_nav_; tab_nav_ = val; }
int tab_nav() const { return tab_nav_; }
34.72 Fl_Input_Choice.H 

3 // Copyright 2004 by Greg Ercolano. 
4 // Copyright 1998-2021 by Bill Spitzak and others. 
5 // This library is free software. Distribution and use rights are outlined in 
6 // the file "COPYING" which should have been included with this file. If this 
7 // file is missing or damaged, see the license at: 
8 // https://www.fltk.org/COPYING.php 
9 // Please see the following page on how to report bugs and issues: 
10 // https://www.fltk.org/bugs.php 
11 */ 
12 
13 /*
14 file Fl_Input_Choice widget . */ 
15 
16 #ifndef Fl_Input_Choice_H 
17 #define Fl_Input_Choice_H 
18 
19 #include <FL/Fl.H> 
20 #include <FL/Fl_Group.H> 
21 #include <FL/Fl_Input.H> 
22 #include <FL/Fl_Menu_Button.H> 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
33 
34 A combination of the input widget and a menu button. 
35 The user can either type into the input area, or use the 
36 menu button chooser on the right to choose an item which loads 
37 the input area with the selected text. 
38 Note: doxygen docs in src/Fl_Input_Choice.cxx 
39 */ 
40 
41 class FL_EXPORT Fl_Input_Choice : public Fl_Group { 
42 // Private class to handle slightly 'special' behavior of menu button 
43 class InputMenuButton : public Fl_Menu_Button { 
44 void draw() FL_OVERRIDE; 
45 const Fl_Menu_Item * popup(); 
46 public: 
47 InputMenuButton(int X, int Y, int W, int H, const char *L=0); 
48 int handle(int e) FL_OVERRIDE; 
49 }; 
50 
51 Fl_Input *inp_; 
52 InputMenuButton *menu_; 
53 
54 
55 // note: this is used by the Fl_Input_Choice ctor. 
56 static void menu_cb(Fl_Widget *, void *data); 
57 
58 // note: this is used by the Fl_Input_Choice ctor. 
59 static void inp_cb(Fl_Widget *, void *data); 
60 
61 protected: 
62 // Custom resize behavior -- input stretches, menu button doesn’t 
63 
64 virtual int inp_x()const { return(x() + Fl::box_dx(box())); } 
65 virtual int inp_y()const { return(y() + Fl::box_dy(box())); } 
66 virtual int inp_w()const { return(w() - Fl::box_dw(box()) - 20); } 
67 virtual int inp_h()const { return(h() - Fl::box_dh(box())); } 
68 
69 virtual int menu_x()const { return(x() + w() - 20 - Fl::box_dx(box())); } 
70 virtual int menu_y()const { return(y() + Fl::box_dy(box())); } 
71 virtual int menu_w()const { return(20); } 
72 virtual int menu_h()const { return(h() - Fl::box_dh(box())); } 
73 
74 public: 
75 Fl_Input_Choice(int X, int Y, int W, int H, const char *L=0); 
76 void resize(int X, int Y, int W, int H) FL_OVERRIDE; 
77 
78 void add(const char *s) { menu_->add(s); } 
79 
80 int changed()const { return inp_->changed() | Fl_Widget::changed(); } 
81 
82 // Clears the changed() state of both input and menu button widgets. 
83 void clear_changed(); 
84 
85 // Sets the changed() state of both input and menu button widgets. 
86 
87 Generated by Doxygen
```cpp
void set_changed();
void clear() { menu_->clear(); }

Fl_Boxtype down_box() const { return (menu_->down_box()); }

void down_box(Fl_Boxtype b) { menu_->down_box(b); }

const Fl_Menu_Item *menu() { return (menu_->menu()); }

void menu(const Fl_Menu_Item *m) { menu_->menu(m); }

Fl_Color textcolor() const { return (inp_->textcolor()); }
void textcolor(Fl_Color c) { inp_->textcolor(c); }

Fl_Font textfont() const { return (inp_->textfont()); }
void textfont(Fl_Font f) { inp_->textfont(f); }

Fl_Fontsize textsize() const { return (inp_->textsize()); }
void textsize(Fl_Fontsize s) { inp_->textsize(s); }
const char * value() const { return (inp_->value()); }
void value(const char *val) { inp_->value(val); }

/* Chooses item \p val in the menu, and sets the Fl_Input text field
to that value. Any previous text is cleared. */
void value(int val);

Fl_Int_Input::Fl_Int_Input(int X, int Y, int W, int H, const char *l = 0) {}

Fl_Menu_Button *menubutton() { return menu_; }
Fl_Input *input() { return inp_; }
```

34.73  

### Fl_Int_Input.H

// Integer input header file for the Fast Light Tool Kit (FLTK).

// Copyright 1998-2010 by Bill Spitzak and others.

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:

// https://www.fltk.org/COPYING.php

// Please see the following page on how to report bugs and issues:

// https://www.fltk.org/bugs.php

17 /* file
18 Fl_Int_Input widget . */
19
20 #ifndef Fl_Int_Input_H
21 #define Fl_Int_Input_H
22
23 class FL_EXPORT Fl_Int_Input : public Fl_Input {
24 public:
25 
26 Fl_Int_Input(int X, int Y, int W, int H, const char *l = 0);
27 public;
28 Fl_Int_Input();
29 
30 #endif
```

34.74  

### Fl_Int_Vector.H File Reference

An STL-ish vector implemented without templates.

#include <FL/Fl_Export.H>
```
Classes

- class Fl_Int_Vector

  An STL-ish vector without templates.

### 34.74.1 Detailed Description

An STL-ish vector implemented without templates.

### 34.75 Fl_Int_Vector.H

Go to the documentation of this file.

```cpp
#ifndef Fl_Int_Vector_H
#define Fl_Int_Vector_H

#include <FL/Fl_Export.H>

class FL_EXPORT Fl_Int_Vector {
  int *arr_;  
  unsigned int size_;  

  void init() {  
    arr_ = 0;  
    size_ = 0;  
  }

  void copy(int *newarr, unsigned int newsize);

 public:
  Fl_Int_Vector() {  
    init();  
  }

  ~Fl_Int_Vector();

  Fl_Int_Vector(Fl_Int_Vector &o) {  
    init();  
    copy(o.arr_, o.size_);  
  }

  Fl_Int_Vector &operator=(Fl_Int_Vector &o) {  
    init();  
    copy(o.arr_, o.size_);  
    return *this;  
  }

  int operator[](int x) const {  
    return arr_[x];  
  }

  int &operator[](int x) {  
    return arr_[x];  
  }

  unsigned int size() const {  
    return size_;  
  }

  void size(unsigned int count);

  int pop_back() {  
    int tmp = arr_[size_ - 1];  
    size_--;  
    return tmp;  
  }

};

#endif
```

Generated by Doxygen
void push_back(int val) {
    unsigned int x = size_; 
    size_(size_ + 1);
    arr_[x] = val;
}

int back(const) { return arr_[size_ - 1]; }

bool empty(const) { return (size_ == 0) ? true : false; }
};

#endif // Fl_Int_Vector_H

---

# Fl_JPEG_Image.H

1 // 2 // JPEG image header file for the Fast Light Tool Kit (FLTK). 3 // 4 // Copyright 1998-2010 by Bill Spitzak and others. 5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php 15 // 16 // 17 /* \file 18 Fl_JPEG_Image class. */ 19 #ifndef Fl_JPEG_Image_H 20 #define Fl_JPEG_Image_H 21 #include "Fl_RGB_Image.H" 22 #include "FL_image.H" 23 24 class FL_EXPORT Fl_JPEG_Image : public Fl_RGB_Image {
25 public:
26    Fl_JPEG_Image(const char *filename);
27    Fl_JPEG_Image(const char *name, const unsigned char *data, int data_length=-1); 28      protected:
29        load_jpg_(const char *filename, const char *sharename, const unsigned char *data, int data_length=-1); 30 }; 31 #endif

---

# Fl_Light_Button.H

1 // 2 // Lighted button header file for the Fast Light Tool Kit (FLTK). 3 // 4 // Copyright 1998-2022 by Bill Spitzak and others. 5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php 15 // 16 // 17 /* \file 18 Fl_Light_Button widget. */ 19 #ifndef Fl_Light_Button_H 20 #define Fl_Light_Button_H 21 #endif
```cpp
#include "Fl_Button.H"

class FL_EXPORT Fl_Light_Button : public Fl_Button {
protected:
  void draw() FL_OVERRIDE;
public:
  int handle(int) FL_OVERRIDE;
  Fl_Light_Button(int x, int y, int w, int h, const char *l = 0);
};

#endif
```

---

### Fl_Line_Dial.H

1 //
2 // Line dial header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //
17 //"File
18 Fl_Line_Dial widget. */
19#ifndef Fl_Line_Dial_H
20#define Fl_Line_Dial_H
21#endif
22
class FL_EXPORT Fl_Line_Dial : public Fl_Dial {
public:
  Fl_Line_Dial(int X, int Y, int W, int H, const char *L = 0);
};

#endif
```

---

### Fl_Menu.H

1 //
2 // Old menu header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //
17 // this include file is for back compatibility only
18 #include "Fl_Menu_Item.H"

---

### Fl_Menu_.H

1 //
2 // Menu base class header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2019 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php

Generated by Doxygen
FILE Documentation

11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 /*
17 file
18 Fl_Menu_ widget. */
19
20 ifndef Fl_Menu_H
21 #define Fl_Menu_H
22
23 ifndef Fl_Widget_H
24 #include "Fl_Widget.H"
25 #endif
26 #include "Fl_Menu_Item.H"
27
28 class FL_EXPORT Fl_Menu_ : public Fl_Widget {
29   Fl_Menu_Item *menu_;
30   const Fl_Menu_Item *value_;
31
32   protected:
33
34   uchar alloc; // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
35   uchar down_box_;          // down_box_ is Fl_Boxtype;
36   Fl_Font textfont_;        // font for the menu;
37   Fl_Fontsize textsize_;     // size for the text;
38   Fl_Color textcolor_;      // color for the menu text;
39
40   int item_pathname_(char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const;
41   public:
42     Fl_Menu_(int,int,int,int,const char * =0);
43     ~Fl_Menu_();
44
45     int item_pathname(char *name, int namelen, const Fl_Menu_Item *finditem=0) const;
46     const Fl_Menu_Item * picked(const Fl_Menu_Item*);
47     const Fl_Menu_Item * find_item(const char *name);
48     const Fl_Menu_Item * find_item(Fl_Callback*);
49     int find_index(const char *name) const;
50     int find_index(const Fl_Menu_Item *item) const;
51     int find_index(Fl_Callback *cb) const;
52
53     const Fl_Menu_Item * test_shortcut() {return picked(menu()->test_shortcut());}
54     void global();
55
56     const Fl_Menu_Item *menu()const {return menu_;
57     const Fl_Menu_Item *menu_end(); // in src/Fl_Menu_add.cxx
58     void copy(const Fl_Menu_Item *m, void *user_data = 0);
59     int insert(int index, const char *, int shortcut, Fl_Callback*, void* = 0, int = 0);
60     int add(const char *, int shortcut, Fl_Callback*, void* = 0, int = 0); // see src/Fl_Menu_add.cxx
61     int add(const char * a, const char* b, Fl_Callback* c, void* d = 0, int e = 0) {
62     return add(a,fl_old_shortcut(b),c,d,e);
63     }
64     int insert(int index, char* a, char* b, Fl_Callback* c, void* d = 0, int e = 0) {
65     return insert(index,a,fl_old_shortcut(b),c,d,e);
66     }
67     int add(const char *);
68     int size() const ;
69     void size(int W, int H) { Fl_Widget::size(W, H); }
70     void clear();
71     int clear_submenu(int index);
72     void replace(int,Const char *);
73     void remove(int);
74     void shortcut(int i, int s) { menu_[i].shortcut(s); }
75     void mode(int i, int fl) { menu_[i].flags = fl; }
76     int mode(int i) const { return menu_[i].flags; }
77
78     const Fl_Menu_Item *mvalue()const {return value_;
79     int value()const {return value_ ? (int)(value_-menu_) : -1;}  
80     int value(const Fl_Menu_Item *);
81     int value(int i) {return value(menu_+i);}
82     const char *text()const {return value_ ? value_->text : 0;}
83     const char *text(int i)const {return menu_[i].text;
84
85     Fl_Font textfont()const {return textfont_;
86     void textfont(Fl_Font c) {textfont_ =c; }
87     Fl_Fontsize textsize()const {return textsize_;
88     void textsize(Fl_Fontsize c) {textsize_ =c; }
89     Fl_Color textcolor()const {return textcolor_;
90     void textcolor(Fl_Color c) {textcolor_ =c; }
91
92     Fl_Boxtype down_box()const {return (Fl_Boxtype)down_box_;}
93     void down_box(Fl_Boxtype b) { down_box_ = b; }
94
95     */
96
34.81 Fl_Menu_Bar.H

1 //
2 // Menu bar header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2017 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* file
18 Fl_Menu_Bar widget. */
19
20 ifndef Fl_Menu_Bar_H
21 #define Fl_Menu_Bar_H
22
23 #include "Fl_Menu_.H"
24
25 class FL_EXPORT Fl_Menu_Bar : public Fl_Menu_ {
26 friend class Fl_Sys_Menu_Bar_Driver;
27
28 protected:
29     void draw() FL_OVERRIDE;
30    
31 public:
32    int handle(int) FL_OVERRIDE;
33 Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);
34
35    virtual void update() {}
36 }
37 #endif

34.82 Fl_Menu_Button.H

1 //
2 // Menu button header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* file
18 Fl_Menu_Button widget. */
19
20 ifndef Fl_Menu_Button_H
21 #define Fl_Menu_Button_H
22
23 #include "Fl_Menu_.H"
24
25 class FL_EXPORT Fl_Menu_Button : public Fl_Menu_ {
26     enum popup_buttons {POPUP1 = 1,
27     POPUP2, 
28     POPUP12, 
29     POPUP3, 
30     POPUP13,
31     POPUP23,
32
33     protected:
34     void draw() FL_OVERRIDE;
35     static Fl_Menu_Button* pressed_menu_button;
36     public:
37         enum popup_buttons (POPUP1 = 1,
38         POPUP2,
39         POPUP12,
40         POPUP3, 
41         POPUP13, 
42         POPUP23,
# Fl_Menu_Item.H File Reference

```c
#include "Fl_Widget.H"
#include "Fl_Image.H"
#include <FL/platform_types.h>
```

## Classes
- **struct Fl_Menu_Item**

  The `Fl_Menu_Item` structure defines a single menu item that is used by the `Fl_Menu_` class.

## Typedefs
- typedef `Fl_Menu_Item` Fl_Menu

## Enumerations
- 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_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_Shortcut fl_oldShortcut (const char *)**

  Emulation of XForms named shortcuts.

### 34.83.1 Enumeration Type Documentation

#### 34.83.1.1 anonymous enum

Anonymous enum

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_MENU_INACTIVE</td>
<td>Deactivate menu item (gray out)</td>
</tr>
<tr>
<td>FL_MENU_TOGGLE</td>
<td>Item is a checkbox toggle (shows checkbox for on/off state)</td>
</tr>
<tr>
<td>FL_MENU_VALUE</td>
<td>The on/off state for checkbox/radio buttons (if set, state is 'on')</td>
</tr>
<tr>
<td>FL_MENU_RADIO</td>
<td>Item is a radio button (one checkbox of many can be on)</td>
</tr>
<tr>
<td>FL_MENU_INVISIBLE</td>
<td>Item will not show up (shortcut will work)</td>
</tr>
</tbody>
</table>
### 34.84 Fl_Menu_Item.H

Go to the documentation of this file.

```c
#ifndef Fl_Menu_Item_H
#define Fl_Menu_Item_H

#include "Fl_Widget.H"
#include "Fl_Image.H"
#include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL

// doxygen needs the following line to enable e.g. ::FL_MENU_TOGGLE to link to the enums
enum { // values for flags:
    FL_MENU_INACTIVE = 1,
    FL_MENU_TOGGLE = 2,
    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
};

extern FL_EXPORT Fl_Shortcut fl_old_shortcut(const char*);

class Fl_Menu;

struct FL_EXPORT Fl_Menu_Item {
    const char *text;
    int shortcut_; 
    Fl_Callback *callback_; 
    void *user_data_; 
    int flags;
    uchar labeltype_; 
    Fl_Font labelfont_; 
    Fl_Fontsize labelsize_; 
    Fl_Color labelcolor_; 

    // advance N items, skipping submenus:
    const Fl_Menu_Item *next(int=1) const;

    Fl_Menu_Item *next(int i=1) { return (Fl_Menu_Item *)((const Fl_Menu_Item*)this)->next(i); } 

    const Fl_Menu_Item *first(){ return next(0); } 

    // methods on menu items:
    const char *label();
    void label(const char* a) { text=a; }

};
```

---

**Enumerator**

<table>
<thead>
<tr>
<th>FL_SUBMENU_POINTER</th>
<th>Indicates user_data() is a pointer to another menu array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_SUBMENU</td>
<td>Item is a submenu to other items.</td>
</tr>
<tr>
<td>FL_MENU_DIVIDER</td>
<td>Creates divider line below this item. Also ends a group of radio buttons.</td>
</tr>
<tr>
<td>FL_MENU_HORIZONTAL</td>
<td>??? – reserved, internal (do not use)</td>
</tr>
<tr>
<td>FL_MENU_RESERVED</td>
<td>These bits are reserved for internal or future usage (do not use)</td>
</tr>
</tbody>
</table>
154   void label(Fl_Labeltype a, const char* b) {labeltype_ = a; text = b;}
155
156   Fl_Labeltype labeltype() const { return (Fl_Labeltype)labeltype_; }
157
158   void labeltype(Fl_Labeltype a) { labeltype_ = a; }
159
160   Fl_Color labelcolor() const { return labelcolor_; }
161
162   void labelcolor(Fl_Color a) { labelcolor_ = a; }
163
164   Fl_Font labelfont() const { return labelfont_; }
165
166   void labelfont(Fl_Font a) { labelfont_ = a; }
167
168   Fl_Fontsize labelsize() const { return labelsize_; }
169
170   void labelsize(Fl_Fontsize a) { labelsize_ = a; }
171
172   Fl_Callback_p callback() const { return callback_; }
173
174   void callback(Fl_Callback* c, void* p) { callback_ = c; user_data_ = p; }
175
176   void callback(Fl_Callback* c) { callback_ = c; }
177
178   void callback(Fl_Callback0* c) { callback_ = (Fl_Callback*)c; }
179
180   void callback(Fl_Callback1* c, long p=0) { callback_=c; user_data_=(void*)(fl_intptr_t)p; }
181
182   void user_data() const { return user_data_; }
183
184   void user_data(void* v) { user_data_ = v; }
185
186   int argument() const { return (long)(fl_intptr_t)user_data_; }
187
188   void argument(long v) { user_data_ = (void*)(fl_intptr_t)v; }
189
190   int shortcut() const { return shortcut_; }
191
192   void shortcut(int s) { shortcut_ = s; }
193
194   int submenu() const { return flags&(FL_SUBMENU|FL_SUBMENU_POINTER); }
195
196   int checkbox() const { return flags&FL_MENU_TOGGLE; }
197
198   int radio() const { return flags&FL_MENU_RADIO; }
199
200   int value() const { return (flags & FL_MENU_VALUE) ? 1 : 0; }
201
202   void value(int v) { v ? set() : clear(); }
203
204   void set() { flags |= FL_MENU_VALUE; }
205
206   void clear() { flags &= ~FL_MENU_VALUE; }
207
208   void setonly(Fl_Menu_Item const* first = NULL); 
209
210   int visible() const { return !(flags&FL_MENU_INVISIBLE); }
211
212   void show() { flags &= ~FL_MENU_INVISIBLE; }
213
214   int active() const { return !(flags&FL_MENU_INACTIVE); }
215
216   void activate() { flags &= ~FL_MENU_INACTIVE; }
217
218   void deactivate() { flags = FL_MENU_INACTIVE; }
219
220   int activevisible() const { return !(flags & (FL_MENU_INACTIVE|FL_MENU_INVISIBLE)); }
221
222   // compatibility for FLUID so it can set the image of a menu item...
223
224   void image(Fl_Image* image) { image->label(this); }
225
226   void image(Fl_Image& image) { image.label(this); }
227
228   // used by menubar:
229   int measure(int* h, const Fl_Menu_* const); 
230
231   void draw(int x, int y, int w, int h, const Fl_Menu_* const, int t=0) const; 
232
233   // popup menus without using an Fl_Menu_ widget:
234   const Fl_Menu_Item* popup()
235     { int X, int Y, 
236       char *title = 0,
237       const Fl_Menu_Item* picked=0,
238       const Fl_Menu_* = 0; const;
239       const Fl_Menu_Item* pulldown();
240     int X, int Y, int W, int H, 
241     const Fl_Menu_Item* picked = 0,
242     const Fl_Menu_* = 0, 
243     const Fl_Menu_Item* title = 0, 
244     int menubar=0; const;
245     const Fl_Menu_Item* test_shortcut() const;
246     const Fl_Menu_Item* findShortcut(int *ip=0, const bool require_alt = false) const;
247     void do_callback(Fl_Widget* o) const { Fl::callback_reason_=FL_REASON_SELECTED; callback_=(};
248
249   // File Documentation
250
251   void image(Fl_Image* image) { image->label(this); }
252
253   void image(Fl_Image& image) { image.label(this); }
254
255   // compatibility for FLUID so it can set the image of a menu item...
256   void image(Fl_Image* image) { image->label(this); }
257
258   void image(Fl_Image& image) { image.label(this); }
259
260   // used by menubar:
261   int measure(int* h, const Fl_Menu_* const); 
262
263   void draw(int x, int y, int w, int h, const Fl_Menu_* const, int t=0) const; 
264
265   // popup menus without using an Fl_Menu_ widget:
266   const Fl_Menu_Item* popup()
267     { int X, int Y, 
268       char *title = 0,
269       const Fl_Menu_Item* picked=0,
270     const Fl_Menu_* = 0; const;
271     const Fl_Menu_Item* pulldown();
272     int X, int Y, int W, int H, 
273     const Fl_Menu_Item* picked = 0,
274     const Fl_Menu_* = 0, 
275     const Fl_Menu_Item* title = 0, 
276     int menubar=0; const;
277     const Fl_Menu_Item* test_shortcut() const;
278     const Fl_Menu_Item* findShortcut(int *ip=0, const bool require_alt = false) const;
279     void do_callback(Fl_Widget* o) const { Fl::callback_reason_=FL_REASON_SELECTED; callback_=(};
280
281   // File Documentation
282
283   void image(Fl_Image* image) { image->label(this); }
284
285   void image(Fl_Image& image) { image.label(this); }
286
287   // used by menubar:
288   int measure(int* h, const Fl_Menu_* const); 
289
289   void draw(int x, int y, int w, int h, const Fl_Menu_* const, int t=0) const; 
290
291   // popup menus without using an Fl_Menu_ widget:
292   const Fl_Menu_Item* popup()
293     { int X, int Y, 
294       char *title = 0,
295       const Fl_Menu_Item* picked=0,
296     const Fl_Menu_* = 0; const;
297     const Fl_Menu_Item* pulldown();
298     int X, int Y, int W, int H, 
299     const Fl_Menu_Item* picked = 0,
300     const Fl_Menu_* = 0, 
301     const Fl_Menu_Item* title = 0, 
302     int menubar=0; const;
303     const Fl_Menu_Item* test_shortcut() const;
304     const Fl_Menu_Item* findShortcut(int *ip=0, const bool require_alt = false) const;
305     void do_callback(Fl_Widget* o) const { Fl::callback_reason_=FL_REASON_SELECTED; callback_=(}
34.85 Fl_Menu_Window.H

typedef Fl_Menu_Item Fl_Menu; // back compatibility
enum { // back-compatibility enum:
  FL_PUP_NONE = 0,
  FL_PUP_GREY = FL_MENU_INACTIVE,
  FL_PUP_GRAY = FL_MENU_INACTIVE,
  FL_MENU_BOX = FL_MENU_TOGGLE,
  FL_PUP_BOX = FL_MENU_TOGGLE,
  FL_MENU_CHECK = FL_MENU_VALUE,
  FL_PUP_CHECK = FL_MENU_VALUE,
  FL_PUP_RADIO = FL_MENU_RADIO,
  FL_PUP_INVISIBLE = FL_MENU_INVISIBLE,
  FL_PUP_SUBMENU = FL_SUBMENU_POINTER
};

34.86 fl_message.H

1 //
2 // Standard message header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2023 by Bill Spitzak and others.
#ifndef _FL_fl_message_H_
#define _FL_fl_message_H_

#include "fl_ask.H"

#endif

34.87 Fl_Multi_Browser.H

1 //
2 // Multi browser header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 /*
17 Fl_Multi_Browser widget. */
18
ifndef Fl_Multi_Browser_H
#define Fl_Multi_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Multi_Browser : public Fl_Browser {
public:
    Fl_Multi_Browser(int X, int Y, int W, int H, const char *L=0);
};

#endif

34.88 Fl_Multi_Label.H

1 //
2 // Multi-label header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2015 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 /*
17 Fl_Multi_Label.H
18 ifndef Fl_Multi_Label_H
19 define Fl_Multi_Label_H
20 class Fl_Widget;
21 struct Fl_Menu_Item;
22
23 struct FL_EXPORT Fl_Multi_Label {
24 const char* labela;
25 const char* labelb;
26 uchar typea;
27 uchar typeb;

# Fl_Multiline_Input.H

```cpp
void label(Fl_Widget *);
void label(Fl_Menu_Item *);
};

#endif
```

## Fl_Multiline_Output.H

```cpp
class FL_EXPORT Fl_Multiline_Output : public Fl_Output {
public:
  Fl_Multiline_Output(int X, int Y, int W, int H, const char *l = 0);
};

#endif
```

### Fl_Native_File_Chooser.H File Reference

```
#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>
```

**Fl_Native_File_Chooser widget.**

```cpp
#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>
```
Classes

- class Fl_Native_File_Chooser

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

34.91.1 Detailed Description

Fl_Native_File_Chooser widget.

34.92 Fl_Native_File_Chooser.H

Go to the documentation of this file.

```
1 //
2 // FLTK native OS file chooser widget
3 //
4 // Copyright 1998-2016 by Bill Spitzak and others.
5 // Copyright 2004 Greg Ercolano.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:
14 //
15 // https://www.fltk.org/bugs.php
16 //
17 //
18 /* Implementation note:
19
20 class Fl_Native_File_Chooser <== public API used by applications
21
22 class Fl_Native_File_Chooser_Driver <== virtual API that a platform may implement
23 this API has a do-nothing default implementation
24
25 class Fl_Native_File_Chooser_FLTK_Driver <== this API implementation is the default FLTK file chooser
26 class Fl_Kdialog_Native_File_Chooser_Driver <== this API implementation runs a KDE file chooser
27 it is determined at run-time if the GTK dynamic libraries are available
28 and the KDE file chooser runs under the KDE desktop
29
30 class Fl_Quartz_Native_File_Chooser_Driver <== this API implementation runs a Mac OS X file chooser
31
32 class Fl_WinAPI_Native_File_Chooser_Driver <== this API implementation runs a Windows file chooser
33
34 Each platform must implement the constructor of the Fl_Native_File_Chooser class.
35 This particular implementation:
36 Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
37    platform_fnfc = new Fl_Native_File_Chooser_FLTK_Driver(val);
38 }
39
can be used by any platform.
40 No more code is required. The cross-platform Fl_Native_File_Chooser_FLTK.cxx file must be compiled in
41 libfltk,
42 and the default FLTK file chooser will be used.
43
44 This other implementation:
45 Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
46    platform_fnfc = 0;
47 }
48
can be used by a platform that needs no file chooser.
49 */
50
51 #ifndef FL_NATIVE_FILE_CHOOSER_H
52 #define FL_NATIVE_FILE_CHOOSER_H
53
54 #include <FL/Fl_Export.H>
55 #include <FL/Fl_File_Chooser.H>
56
57 class Fl_Native_File_Chooser_Driver;
58
59 #endif
60 #endif
61```
```cpp
enum Type {
    BROWSE_FILE = 0,
    BROWSE_DIRECTORY,
    BROWSE_MULTI_FILE,
    BROWSE_MULTI_DIRECTORY,
    BROWSE_SAVE_FILE,
    BROWSE_SAVE_DIRECTORY
};

enum Option {
    NO_OPTIONS = 0x0000,
    SAVEAS_CONFIRM = 0x0001,
    NEW_FOLDER = 0x0002,
    PREVIEW = 0x0004,
    USE_FILTER_EXT = 0x0008
};

static const char *file_exists_message;

Fl_Native_File_Chooser(int val = BROWSE_FILE); // each platform implements it
~Fl_Native_File_Chooser();
void type(int t);
int type() const;
void options(int o);
int options() const;
int count() const;
const char *filename() const;
const char *filename(int i) const;
void directory(const char *val);
const char *directory() const;
void title(const char *t);
const char *title() const;
const char *filter() const;
void filter(const char *f);
int filters() const;
void filter_value(int i);
int filter_value() const;
void preset_file(const char *f);
const char *preset_file() const;
const char *errmsg() const;
int show();

Fl_Native_File_Chooser_Driver {
protected:
    static void chrcat(char *s, char c);
    static char *strapp(char *s, const char *val);
    static char *strfree(char *val);
    static char *strnew(const char *val);
public:
    Fl_Native_File_Chooser_Driver(int) {};
    virtual ~Fl_Native_File_Chooser_Driver() {};
    virtual void type(int) {};
    virtual int type() const { return 0; }
    virtual void options(int) {};
    virtual int options() const { return 0; }
    virtual int count() const { return 0; }
    virtual const char *filename() const { return 0; }
    virtual const char *filename(int) const { return 0; }
    virtual void directory(const char *) {};
    virtual const char *directory() const { return 0; }
    virtual void title(const char *) {};
    virtual const char *title() const { return 0; }
    virtual const char *filter() const { return 0; }
    virtual void filter(const char *) {};
    virtual int filters() const { return 0; }
    virtual void filter_value(int) {};
    virtual int filter_value() const { return 0; }
    virtual void preset_file(const char *) {};
    virtual const char *preset_file() const { return 0; }
    virtual const char *errmsg() const { return 0; }
    virtual int show() { return 1; }
}

class Fl_Native_File_Chooser_FLTK_Driver : public Fl_Native_File_Chooser_Driver {
    private:
        int _nfiles;
        int _options;
        int _filtvalue;
        char _filter[;]
    protected:
        void _show_fl_file(int val);
    public:
        virtual ~Fl_Native_File_Chooser_FLTK_Driver() {};
        virtual int type() const { return 0; }
        virtual void options(int) {};
        virtual int options() const { return 0; }
        virtual int count() const { return 0; }
        virtual const char *filename() const { return 0; }
        virtual const char *filename(int) const { return 0; }
        virtual void directory(const char *) {};
        virtual const char *directory() const { return 0; }
        virtual void title(const char *) {};
        virtual const char *title() const { return 0; }
        virtual const char *filter() const { return 0; }
        virtual void filter(const char *) {};
        virtual int filters() const { return 0; }
        virtual void filter_value(int) {};
        virtual int filter_value() const { return 0; }
        virtual void preset_file(const char *) {};
        virtual const char *preset_file() const { return 0; }
        virtual const char *errmsg() const { return 0; }
        virtual int show() { return 1; }
};

protected:
    int _btype; // kind-of browser to show
    int _options; // general options
    char _filter[]; // user supplied filter
    char _parsedfilt[]; // parsed filter
    int _filtvalue; // selected filter
    char _preset_file[];

Generated by Doxygen
```
char * _prevvalue; // Returned filename
char * _directory;
char * _errmsg; // error message
Fl_File_Chooser * _file_chooser;

public:
Fl_Native_File_Chooser_FLTK_Driver(int val);
virtual ~Fl_Native_File_Chooser_FLTK_Driver();

void type(int t) FL_OVERRIDE;
int type() const FL_OVERRIDE;
void options(int o) FL_OVERRIDE;
int options() const FL_OVERRIDE;
int count() const FL_OVERRIDE;
const char * filename() const FL_OVERRIDE;
const char * filename(int i) const FL_OVERRIDE;
void directory(const char *val) FL_OVERRIDE;
const char * directory() const FL_OVERRIDE;
void title(const char * t) FL_OVERRIDE;
const char * title() const FL_OVERRIDE;
const char * filter() const FL_OVERRIDE;
void filter(const char * f) FL_OVERRIDE;
int filters() const FL_OVERRIDE;
void filter_value(int i) FL_OVERRIDE;
int filter_value() const FL_OVERRIDE;
void preset_file(const char * f) FL_OVERRIDE;
const char * preset_file() const FL_OVERRIDE;
const char * errmsg() const FL_OVERRIDE;
int show() FL_OVERRIDE;

#endif /*FL_NATIVE_FILE_CHOOSER_H*/

34.93 Fl_Nice_Slider.H

1 //
2 // "Nice" slider header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* \file
18 Fl_Nice_Slider widget . */
19
20 #ifndef Fl_Nice_Slider_H
21 #define Fl_Nice_Slider_H
22
23 #include "Fl_Slider.H"
24
25 class FL_EXPORT Fl_Nice_Slider : public Fl_Slider {
26 public:
27 Fl_Nice_Slider(int X, int Y, int W, int H, const char *L=0);
28
29 #endif

34.94 Fl_Object.H

1 //
2 // Old Fl_Object header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
34.95 Fl_Output.H

1 // This file is provided for back compatibility only. Please use Fl_Widget
2 ifndef Fl_Object
3 #define Fl_Object Fl_Widget
4 #endif
5 #include "Fl_Widget.H"

34.96 Fl_Overlay_Window.H

1 //
2 // Overlay window header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 //
17 /*
18 Fl_Overlay_Window class. */
19 #ifndef Fl_Overlay_Window_H
20 #define Fl_Overlay_Window_H
21 #ifndef FL_DOXYGEN
22 friend class _Fl_Overlay;
23 friend class Fl_Window_Driver;
24 #endif
25 #endif
26 class FL_EXPORT Fl_Overlay_Window : public Fl_Double_Window {
27 public:
28   Fl_Overlay_Window(int X, int Y, int W, int H, const char *l = 0);
29   Fl_Overlay_Window();
30   ~Fl_Overlay_Window();
31   int can_do_overlay();
32   void redraw_overlay();
33 protected:
34   virtual void draw_overlay() = 0;
35   virtual void show() FL_OVERRIDE;
36   virtual void hide() FL_OVERRIDE;
37   virtual void resize(int, int, int, int) FL_OVERRIDE;
38   virtual ~Fl_Overlay_Window();
39 private:
40   Fl_Double_Window *overlay_;
# Fl_Pack.H

```cpp
1 //
2 // Pack header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2020 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 //
17 /*
18 * Fl_Pack widget. */
19 #ifndef Fl_Pack_H
20 #define Fl_Pack_H
21 #define Fl_Pack_H
22 #include <FL/Fl_Group.H>
23 class FL_EXPORT Fl_Pack : public Fl_Group {
24     int spacing_; 
25     enum { // values for type(int)
26    VERTICAL = 0,
27    HORIZONTAL = 1
28    }; 
29 protected:
30     void draw() FL_OVERRIDE;
31 public:
32     Fl_Pack(int X, int Y, int W, int H, const char *L = 0);
33     int spacing()const {return spacing_;}
34     void spacing(int i) {spacing_ = i;}
35     uchar horizontal()const {return type();}
36     void resize(int X, int Y, int W, int H) FL_OVERRIDE;
37 };
38 #endif
```

---

# Fl_Paged_Device.H File Reference

Declaration of class `Fl_Paged_Device`.

```cpp
#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`.

34.98.1 Detailed Description
description of class `Fl_Paged_Device`.

34.99 `Fl_Paged_Device.H`

Go to the documentation of this file.

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

#define NO_PAGE_FORMATS 30 /* MSVC6 compilation fix */

class FL_EXPORT Fl_Paged_Device : public Fl_Widget_Surface {
protected:
    Fl_Paged_Device() : Fl_Widget_Surface(NULL) {}
public:
    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
    };  
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    }
public:
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
        REVERSED = 0x200,
        ORIENTATION = 0x300
    };
    typedef struct {
        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
    };
    enum Page_Layout {
        PORTRAIT = 0,
        LANDSCAPE = 0x100,
```cpp
int width;
int height;
const char *name;
)
page_format;
virtual ~Fl_Paged_Device() {}
virtual int begin_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char **perr_message = NULL) {
    return begin_job(pagecount, frompage, topage, perr_message);
}
virtual int begin_page(void);
int start_page() {return begin_page();}
virtual void margins(int *left, int *top, int *right, int *bottom);
virtual void scale(float scale_x, float scale_y = 0.);
virtual void rotate(float angle);
void print_widget(Fl_Widget *widget, int delta_x = 0, int delta_y = 0) {draw(widget, delta_x, delta_y);}
void print_window(Fl_Window *win, int x_offset = 0, int y_offset = 0) {
draw_decorated_window(win, x_offset, y_offset);
}
virtual int end_page (void);
virtual void end_job (void);
};

#endif // Fl_Paged_Device_H

---

34.100 Fl_Pixmap.H

1 //
2 // Pixmap header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2017 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /* file
18 Fl_Pixmap widget. */
19
20 #ifndef Fl_Pixmap_H
21 #define Fl_Pixmap_H
22 # include "Fl_Image.H"
23
24 class Fl_Widget;
25 struct Fl_Menu_Item;
26
27 // Older C++ compilers don't support the explicit keyword... :(.
28 #if defined(__sgi) && !defined(_COMPILER_VERSION)
29 # define explicit
30 #endif // __sgi && !_COMPILER_VERSION
31
32 class Fl_EXPORT Fl_Pixmap : public Fl_Image {
33     friend class Fl_Graphics_Driver;
34     void copy_data();
35     void delete_data();
36     void set_data(const char * const *p);
37     protected:
38     void measure();
39     public:
40     int alloc_data; // Non-zero if data was allocated
41     private:
42     // for internal use
43     Fljumbotron id_;
44     Fljumbotron mask_;
45     int cache_w, cache_h; // size of pixmap when cached
46     public:
47     explicit Fl_Pixmap(char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
48     {set_data((const char*const*)D); measure();}
```

Generated by Doxygen
34.101 Fl_Plugin.H

```cpp
explicit Fl_Pixmap(char const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
(set_data((const char*)D); measure());
explicit Fl_Pixmap(char const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
(set_data((const char*)D)); measure());
explicit Fl_Pixmap(char const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
(set_data((const char*)D); measure());
virtual ~Fl_Pixmap();
Fl_Image *copy(int W, int H) const FL_OVERRIDE;
Fl_Image *copy()const { return Fl_Image::copy(); }
void color_average(Fl_Color c, float i) FL_OVERRIDE;
void desaturate() FL_OVERRIDE;
void draw(int X, int Y, int W, int H, int cx=0, int cy=0) FL_OVERRIDE;
void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);} 
void label(Fl_Widget *w) FL_OVERRIDE;
void label(Fl_Menu_Item *m) FL_OVERRIDE;
void uncache() FL_OVERRIDE;
};
```

```cpp
#endif
```

---

34.101 Fl_Plugin.H

```cpp
/*
file
Fl_Plugin class . */
```

```cpp

class FL_EXPORT Fl_Plugin {

   Fl_Preferences::ID id;

   public:

   Fl_Plugin(const char *klass, const char *name);

   virtual ~Fl_Plugin();

};
```

```cpp
class FL_EXPORT Fl_Plugin_Manager : public Fl_Preferences {

   public:

   Fl_Plugin_Manager(const char *klass);

   ~Fl_Plugin_Manager();

   int plugins() { return groups(); } 

   Fl_Plugin *plugin(int index);

   Fl_Plugin *plugin(const char *name);

   Fl_Preferences::ID addPlugin(const char *name, Fl_Plugin *plugin);

   static void removePlugin(Fl_Preferences::ID id);

   static int load(const char *filename);

   static int loadAll(const char *filepath, const char *pattern=0);

};
```

```cpp
#endif // !Fl_Preferences_H
```

---

34.102 Fl_PNG_Image.H

```cpp
/*
file
PNG image header file for the Fast Light Tool Kit (FLTK).
```

```cpp

class Fl_PNG_Image {

   Fl_Preferences::ID id;

   public:

   Fl_PNG_Image(const char *filename);

   virtual ~Fl_PNG_Image();

};
```

```cpp
```
```cpp
/*
file
FL_PNG_Image class.
*/

#ifndef Fl_PNG_Image_H
#define Fl_PNG_Image_H
#include "Fl_Image.H"

class FL_EXPORT Fl_PNG_Image : public Fl_RGB_Image {
friend class Fl_ICO_Image;
public:

Fl_PNG_Image(const char * filename);
Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int datasize);
private:
Fl_PNG_Image(const char *filename, int offset); // used by Fl_ICO_Image
void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);
};

// Support functions to write PNG image files (since 1.4.0)
FL_EXPORT int fl_write_png(const char *filename, Fl_RGB_Image *img);
FL_EXPORT int fl_write_png(const char *filename, const char *pixels, int w, int h, int d=3, int ld=0);
FL_EXPORT int fl_write_png(const char *filename, const unsigned char *pixels, int w, int h, int d=3, int ld=0);

#endif
```

34.103 Fl_PNM_Image.H

```cpp
/*
file
FL_PNM_Image class.
*/

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H
#include "Fl_Image.H"

class FL_EXPORT Fl_PNM_Image : public Fl_RGB_Image {
public:

Fl_PNM_Image(const char * filename);
};

#endif
```

34.104 Fl_Positioner.H

```cpp
/*
file
FL_Positioner class.
*/

#ifndef Fl_Positioner_H
#define Fl_Positioner_H
#include "Fl_Image.H"

#endif
```
class FL_EXPORT Fl_Positioner : public Fl_Widget {
  double xmin, ymin;
  double xmax, ymax;
  double xvalue_, yvalue_;}

protected:

// these allow subclasses to put the dial in a smaller area:
void draw(int, int, int, int);  // Allows subclasses to draw in a smaller area.
int handle(int, int, int, int, int);  // Handles events.
void draw() FL_OVERRIDE;  // Override method for drawing.

public:

int handle(int) FL_OVERRIDE;  // Override method for handling events.
Fl_Positioner(int x, int y, int w, int h, const char *l=0);  // Constructor.
double xvalue()const {return xvalue_;}  // Get x value.
double yvalue()const {return yvalue_;}  // Get y value.
double xminimum()const {return xmin;}  // Get x minimum.
double xmaximum()const {return xmax;}  // Get x maximum.
double yminimum()const {return ymin;}  // Get y minimum.
double ymaximum()const {return ymax;}  // Get y maximum.
int xvalue(double);  // Set x value.
int yvalue(double);  // Set y value.
int value(double, double);  // Set x and y value.
void xbounds(double, double);  // Set x bounds.
void ybounds(double, double);  // Set y bounds.
void xstep(double a) {xstep_ = a;}  // Set x step.
void ystep(double a) {ystep_ = a;}  // Set y step.
}

#endif
34.105.1 Detailed Description
declaration of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface.

34.105.2 Typedef Documentation

34.105.2.1 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().

34.106 Fl_PostScript.H

Go to the documentation of this file.

1 //
2 // Support for graphics output to PostScript file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 2010-2020 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 ifndef Fl_PostScript_H
17 #define Fl_PostScript_H
18
19 #include <FL/Fl_Paged_Device.H>
20 #include <FL/fl_draw.H>
21 #include <stdarg.h>
22
23 extern "C" {
24 typedef int ( * Fl_PostScript_Close_Command)(FILE *);
25 }
26
27 class Fl_PostScript_Graphics_Driver;
28
29 class FL_EXPORT Fl_PostScript_File_Device : public Fl_Paged_Device {
30 private:
31 73 // memorize the display’s current font to restore it when the object ceases being current
32 74 Fl_Font display_font_;  // remember the display’s current font to restore it when the object ceases being current
33 75 Fl_Fontsize display_size_;  // remember the display’s current font to restore it when the object ceases being current
34    82 inline Fl_PostScript_Graphics_Driver *driver() { return (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); }
35    83 public:
36    84 Fl_PostScript_File_Device();
37    85 ~Fl_PostScript_File_Device();
38 89 int begin_job(int pagecount, int* pagecount, int* to, char** perr_message) FL_OVERRIDE;
39 99 int begin_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
40 100 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);
41 103 int start_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
42 104 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
43 105 return begin_job(pagecount, format, layout);  // return begin_job(pagecount, format, layout);
44 106 }
45 int begin_job(FILE *ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
46 107 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);
47 int start_job(FILE *ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
48 109 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
49 110 return begin_job(ps_output, pagecount, format, layout);
50 111 }
51 int begin_page (void) FL_OVERRIDE;
52 Generated by Doxygen
```c
int printable_rect(int *w, int *h) FL_OVERRIDE;
void margins(int *left, int *top, int *right, int *bottom) FL_OVERRIDE;
void origin(int *x, int *y) FL_OVERRIDE;
void origin(int x, int y) FL_OVERRIDE;
void scale (float scale_x, float scale_y = 0.) FL_OVERRIDE;
void rotate(float angle) FL_OVERRIDE;
void translate(int x, int y) FL_OVERRIDE;
void untranslate(void) FL_OVERRIDE;
int end_page (void) FL_OVERRIDE;
void end_job(void) FL_OVERRIDE;
static const char *file_chooser_title;
FILE *file();
void close_command(Fl_PostScript_Close_Command cmd);
void set_current() FL_OVERRIDE;
void end_current() FL_OVERRIDE;
}

class FL_EXPORT Fl_EPS_File_Surface : public Fl_Widget_Surface {
protected:
    inline Fl_PostScript_Graphics_Driver *driver() { return
        (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); } 
public:
    Fl_EPS_File_Surface(int width, int height, FILE *eps_output,
        Fl_Color background = FL_WHITE, Fl_PostScript_Close_Command closef = NULL);
    ~Fl_EPS_File_Surface();
    int printable_rect(int *w, int *h) FL_OVERRIDE;
    FILE *file();
    void origin(int *x, int *y) FL_OVERRIDE;
    void translate(int x, int y) FL_OVERRIDE;
    void untranslate() FL_OVERRIDE;
    int close();
};

// Preferences implementation for the Fast Light Tool Kit (FLTK).
// Copyright 2002-2023 by Matthias Melcher.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

typedef void *ID;

class Fl_Preferences { 
public:
    enum Root { 
        UNKNOWN_ROOT_TYPE = -1, 
        SYSTEM = 0, 
        USER, 
        MEMORY, 
        BOOT_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 
    }; 
    typedef void *ID;
};
```
static const char *new_UUID();
static const unsigned int NONE = 0x0000;
static const unsigned int USER_READ_OK = 0x0001;
static const unsigned int USER_WRITE_OK = 0x0002;
static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK;
static const unsigned int SYSTEM_READ_OK = 0x0010;
static const unsigned int SYSTEM_WRITE_OK = 0x0020;
static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK;
static const unsigned int APP_OK = SYSTEM_OK | USER_OK;
static const unsigned int CORE_READ_OK = 0x0040;
static const unsigned int CORE_WRITE_OK = 0x0080;
static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK;
static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK;
static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK;
static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK;

static void file_access(unsigned int flags);
static unsigned int file_access();
static Root filename(char *buffer, size_t buffer_size, Root root, const char *vendor, const char *application);

Fl_Preferences( Root root, const char *vendor, const char *application );
Fl_Preferences( Fl_Preferences &parent, const char *group );
Fl_Preferences( Fl_Preferences *parent, const char *group );
Fl_Preferences( Fl_Preferences &parent, int groupIndex );
Fl_Preferences( Fl_Preferences *parent, int groupIndex );
Fl_Preferences(const Fl_Preferences&);
Fl_Preferences( ID id );
virtual ~Fl_Preferences();

Root filename(char *buffer, size_t buffer_size);
ID id() { return (ID)node; }

static char remove(ID id_) { return ((Node *)id_)->remove(); }
const char *name() { return node->name(); }
const char *path() { return node->path(); }
int groups();
const char *group(int num_group );
char group_exists(const char *key );
char delete_group(const char *group );
char delete_all_groups();

int entries();
const char *entry(int index );
char entry_exists(const char *key );
char delete_entry(const char *entry );
char delete_all_entries();
char clear();

char set(const char *entry, int value );
char set(const char *entry, float value );
char set(const char *entry, float value, int precision );
char set(const char *entry, double value );
char set(const char *entry, double value, int precision );
char set(const char *entry, const char *value );
char set(const char *entry, const void *value, int size );
char set(const char *entry, const Fl_String &value );

char get(const char *entry, int &value, int defaultValue );
char get(const char *entry, float &value, float defaultValue );
char get(const char *entry, double &value, double defaultValue );
char get(const char *entry, char &value, char defaultValue );
char get(const char *entry, Fl_String &value );
char get(const char *entry, void &value, void defaultValue, int defaultValueSize, int maxSize );
char get(const char *entry, void &value, void defaultValue, int *size );
char get(const char *entry, Fl_String &value, const Fl_String &defaultValue );

int size(const char *entry );

char get_userdata_path(char *path, int pathlen );
int flush();
int dirty();

static const char *newUUID() { return new_UUID(); }
char groupExists(const char *key ) { return group_exists(key); }
char deleteGroup(const char *group ) { return delete_group(group); }
```cpp
263   char deleteAllGroups() { return delete_all_groups(); }
264   char entryExists( const char *key ) { return entry_exists(key); }
265   char deleteEntry( const char *entry ) { return delete_entry(entry); }
266   char deleteAllEntries() { return delete_all_entries(); }
267   char getUserdataPath( char *path, int pathlen ) { return get_userdata_path(path, pathlen); }
282   class FL_EXPORT Name {
283     char *data_;
284   
285   public:
287     Name( unsigned int n );
288     Name( const char *format, ... );
289     
294     operator const char *() { return data_; }
295     ~Name();
296   
297   struct Entry {
298     char *name, *value;
299   
302   private:
303     FL_Preferences() : node(0), rootNode(0) {}
304     FL_Preferences &operator=(const FL_Preferences&);
305     
307     static char nameBuffer[128];
308     static char uuidBuffer[40];
309     static FL_Preferences *runtimePrefs;
310     static unsigned int fileAccess_;  
311   public: // older Sun compilers need this (public definition of the following classes)
312     class RootNode;
313     
315     class FL_EXPORT Node { // a node contains a list to all its entries
316     // and all means to manage the tree structure
317     Node *first_child_, *next_;
318     union {
319       Node *parent_; // top_ bit clear
320       RootNode *root_node_; // top_ bit set
321     }
322     char *path_;  
323     Entry *entry_;  
324     int nEntry_, NEntry_;  
325     unsigned char dirty_:1;
326     unsigned char top_:1;
327     unsigned char indexed_:1;
328     // indexing routines
329     Node **index_;  
330     int nIndex_, NIndex_;  
331     void createIndex();  
332     void updateIndex();  
333     void deleteIndex();
334   public:
335     static int lastEntrySet;
336   public:
337     Node( const char *path );  
338     ~Node();
339     // node methods
340     int write( FILE *f );
341     const char *name();
342     const char *path() { return path_; }
343     Node *find( const char *path );
344     Node *firstChild( const char *path );
345     void setParent( Node *parent );
346     RootNode *findRoot();
347     char remove();
348     char dirty();
349     void clearDirtyFlags();
350     void deleteAllChildren();
351     // entry methods
353     int nChildren();
354     void set( const char *name, const char *value );
355     void set( const char *line );
356     void add( const char *line );
357     const char *get( const char *name );
358     int getEntry( const char *name );
359     char deleteEntry( const char *name );
360     void deleteAllEntries();
361     int nEntry();
362     Entry entry(int i) { return entry_[i]; }
363   };  
364   friend class Node;  
```
class FL_EXPORT RootNode { // the root node manages file paths and basic reading and writing
  Fl_Preferences *prefs_;
  char *filename_, *vendor_, *application_
  Root root_type_;
public:
  RootNode( Fl_Preferences *, Root root, const char *vendor, const char *application_);
  RootNode( Fl_Preferences *, const char *path, const char *vendor, const char *application_);
  RootNode( Fl_Preferences *);
  ~RootNode();
  int read();
  int write();
  char getPath(char *path, int pathlen);
  char *filename() { return filename_; }
  Root root() { return root_type_; }
};
friend class RootNode;

protected:
  Node *node;
  RootNode *rootNode;
};

#ifndef Fl_Preferences_H
#define Fl_Preferences_H

#include <FL/Fl_Paged_Device.H>

class FL_EXPORT Fl_Prefrences {
private:
  char *filename_, *vendor_, *application_
  Root root_type_;
public:
  Fl_Prefrences( Fl_Prefrences *, Root root, const char *vendor, const char *application_);
  Fl_Prefrences( Fl_Prefrences *, const char *path, const char *vendor, const char *application_);
  Fl_Prefrences( Fl_Prefrences *);
  ~Fl_Prefrences();
  int read();
  int write();
  char getPath(char *path, int pathlen);
  char *filename() { return filename_; }
  Root root() { return root_type_; }
};

#endif // !Fl_Preferences_H

34.108 Fl_Printer.H File Reference

declaration of class Fl_Printer.
#include <FL/Fl_Paged_Device.H>

Classes

• class Fl_Printer

OS-independent print support.

34.108.1 Detailed Description

declaration of class Fl_Printer.

34.109 Fl_Printer.H

Go to the documentation of this file.
1 //
2 // Printing support for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 2010-2016 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 #ifndef Fl_Printer_H
17 #define Fl_Printer_H
18 #include <FL/Fl_Paged_Device.H>
19
20 #ifndef FL_EXPORT
21 #define FL_EXPORT
22 #endif
23
24 class FL_EXPORT Fl_Printer : public Fl_Paged_Device {
25  private:
26   Fl_Paged_Device *printer;
27   static Fl_Paged_Device *newPrinterDriver(void);
28  public:
29   Fl_Printer(void);
30   int begin_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char **perr_message = NULL)
31     FL_OVERRIDE;
32   int end_job(void) FL_OVERRIDE;
33   int printable_rect(int *w, int *h) FL_OVERRIDE;
34
Generated by Doxygen
void margins(int *left, int *top, int *right, int *bottom) FL_OVERRIDE;
void origin(int *x, int *y) FL_OVERRIDE;
void scale(float scale_x, float scale_y = 0.) FL_OVERRIDE;
void rotate(float angle) FL_OVERRIDE;
void translate(int x, int y) FL_OVERRIDE;
void untranslate(void) FL_OVERRIDE;
int end_page (void) FL_OVERRIDE;
void end_job (void) FL_OVERRIDE;
void set_current(void) FL_OVERRIDE;
bool is_current() FL_OVERRIDE;

static const char *dialog_title;
static const char *dialog_printer;
static const char *dialog_range;
static const char *dialog_copies;
static const char *dialog_all;
static const char *dialog_pages;
static const char *dialog_from;
static const char *dialog_to;
static const char *dialog_properties;
static const char *dialog_copyNo;
static const char *dialog_print_button;
static const char *dialog_cancel_button;
static const char *dialog_print_to_file;
static const char *property_title;
static const char *property_pagesize;
static const char *property_mode;
static const char *property_use;
static const char *property_save;
static const char *property_cancel;
~Fl_Printer(void);

// Progress bar widget definitions.
// Copyright 2000-2010 by Michael Sweet.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

/*
Fl_Progress widget. */

#ifndef _Fl_Progress_H_
#define _Fl_Progress_H_

// Include necessary headers.

#include "Fl_Widget.H"

class FL_EXPORT Fl_Progress : public Fl_Widget {

private:
  float value_,
  minimum_,
  maximum_;
34.111  Fl_Radio_Button.H

1 // 2 // Radio button header file for the Fast Light Tool Kit (FLTK).  3 // 4 // Copyright 1998-2014 by Bill Spitzak and others.  5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php

17 /* \file
18 Fl_Radio_Button widget. */
19 #ifndef Fl_Radio_Button_H
20 #define Fl_Radio_Button_H
21 #include "Fl_Button.H"
22 #include "Fl_Light_Button.H"
23 class FL_EXPORT Fl_Radio_Button : public Fl_Button {
24 public:
25 Fl_Radio_Button(int X, int Y, int W, int H, const char *L=0);
26 };  
29 
30 #endif

34.112  Fl_Radio_Light_Button.H

1 // 2 // Radio light button header file for the Fast Light Tool Kit (FLTK).  3 // 4 // Copyright 1998-2014 by Bill Spitzak and others.  5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php

17 /* \file
18 Fl_Radio_Light_Button widget. */
19 #ifndef Fl_Radio_Light_Button_H
20 #define Fl_Radio_Light_Button_H
21 #include "Fl_Button.H"
22 #include "Fl_Light_Button.H"
23 class FL_EXPORT Fl_Radio_Light_Button : public Fl_Light_Button {
24 public:
25 Fl_Radio_Light_Button(int X, int Y, int W, int H, const char *L=0);
26 };  
29 
30 #endif
34.113 Fl_Radio_Round_Button.H

1 // 2 // Radio round button header file for the Fast Light Tool Kit (FLTK). 3 // 4 // Copyright 1998-2014 by Bill Spitzak and others. 5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php 15 // 16 /* 17 file Fl_Radio_Round_Button widget. */ 18 #ifndef Fl_Radio_Round_Button_H 19 #define Fl_Radio_Round_Button_H 20 #include "Fl_Round_Button.H" 21 #ifdef Fl_Radio_Round_Button_H
22 #include "Fl_Round_Button.H" 23 24 class FL_EXPORT Fl_Radio_Round_Button : public Fl_Round_Button { 25 public: 26 Fl_Radio_Round_Button(int X,int Y,int W,int H,const char *L=0); 27 }; 28 29 #endif

34.114 Fl_Rect.H

1 // 2 // Fl_Rect header file for the Fast Light Tool Kit (FLTK). 3 // 4 // Copyright 1998-2022 by Bill Spitzak and others. 5 // 6 // This library is free software. Distribution and use rights are outlined in 7 // the file "COPYING" which should have been included with this file. If this 8 // file is missing or damaged, see the license at: 9 // 10 // https://www.fltk.org/COPYING.php 11 // 12 // Please see the following page on how to report bugs and issues: 13 // 14 // https://www.fltk.org/bugs.php 15 // 16 #ifndef Fl_Rect_H 17 #define Fl_Rect_H 18 #include <FL/Fl_Widget.H> // for c'tor based on Fl_Widget 19 20 #include <FL/Fl_Widget.H> // for c'tor based on Fl_Widget 21 22 class FL_EXPORT Fl_Rect { 23 public: 24 int x_; 25 int y_; 26 int w_; 27 int h_; 28 29 Fl_Rect() : x_(0), y_(0), w_(0), h_(0) {} 30 Fl_Rect(int W, int H) : x_(0), y_(0), w_(W), h_(H) {} 31 Fl_Rect(int X, int Y, int W, int H) : x_(X), y_(Y), w_(W), h_(H) {} 32 Fl_Rect (int X, int Y, int W, int H, Fl_Boxtype bt) 33 : x_(X), y_(Y), w_(W), h_(H) [ 34 inset(bt); 35 } 36 37 public: 38 Fl_Rect () 39 : x_(0), y_(0), w_(0), h_(0) {} 40 Fl_Rect (int W, int H) 41 : x_(0), y_(0), w_(W), h_(H) {} 42 Fl_Rect (int X, int Y, int W, int H) 43 : x_(X), y_(Y), w_(W), h_(H) {} 44 45 Fl_Rect (const Fl_Widget& widget) 46 : x_(widget.x()), y_(widget.y()), w_(widget.w()), h_(widget.h()) {} 47 Fl_Rect (const Fl_Widget* const widget) 48 : x_(widget->x()), y_(widget->y()), w_(widget->w()), h_(widget->h()) {} 49 50 } 51

Generated by Doxygen
```cpp
int x()const { return x_; }
int y()const { return y_; }
int w()const { return w_; }
int h()const { return h_; }

int r()const { return x_ + w_; }
int b()const { return y_ + h_; }

void x(int X) { x_ = X; }
void y(int Y) { y_ = Y; }
void w(int W) { w_ = W; }
void h(int H) { h_ = H; }

void inset(int d) {
    x_ += d;
    y_ += d;
    w_ -= 2 * d;
    h_ -= 2 * d;
}

void inset(Fl_Boxtype bt) {
    x_ += Fl::box_dx(bt);
    y_ += Fl::box_dy(bt);
    w_ -= Fl::box_dw(bt);
    h_ -= Fl::box_dh(bt);
}

void inset(int left, int top, int right, int bottom) {
    x_ += left;
    y_ += top;
    w_ -= (left + right);
    h_ -= (top + bottom);
}

}; // class Fl_Rect

```
34.117 Fl_RGB_Image.H

1 // RGB Image header file for the Fast Light Tool Kit (FLTK).
2 //
3 // Copyright 1998-2010 by Bill Spitzak and others.
4 //
5 // This library is free software. Distribution and use rights are outlined in
6 // the file "COPYING" which should have been included with this file. If this
7 // file is missing or damaged, see the license at:
8 //
9 // https://www.fltk.org/COPYING.php
10 //
11 // Please see the following page on how to report bugs and issues:
12 //
13 // https://www.fltk.org/bugs.php
14 //
15 //
16 #ifndef Fl_RGB_Image_H
17 # define Fl_RGB_Image_H
18 # include "Fl_Image.H"
19 #endif // !Fl_RGB_Image_H

34.118 Fl_Roller.H

1 //
2 // Roller header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 #ifndef Fl_Roller_H
18 # define Fl_Roller_H
19 # include "Fl_Valuator.H"
20 #endif // !Fl_Roller_H

Generated by Doxygen
class FL_EXPORT Fl_Roller : public Fl_Valuator {
    protected:
    void draw() FL_OVERRIDE;
    public:
    int handle(int) FL_OVERRIDE;
    Fl_Roller(int X, int Y, int W, int H, const char* L = 0);
};

#endif

34.119 Fl_Round_Button.H

// Round button header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef Fl_Round_Button_H
#define Fl_Round_Button_H
#include "Fl_Light_Button.H"

class FL_EXPORT Fl_Round_Button : public Fl_Light_Button {
    public:
    Fl_Round_Button(int x, int y, int w, int h, const char *l = 0);
};

#endif

34.120 Fl_Round_Clock.H

// Round clock header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifdef Fl_Round_Clock_H
#define Fl_Round_Clock_H
#include "Fl_Clock.H"

class FL_EXPORT Fl_Round_Clock : public Fl_Clock {
    public:
    Fl_Round_Clock(int X, int Y, int W, int H, const char *L = 0);
};

#endif

34.121 Fl_Scheme.H

// Scheme header for the Fast Light Tool Kit (FLTK).
//
#ifndef FL_Fl_Scheme_H_
#define FL_Fl_Scheme_H_

#include <FL/Fl.H>

class Fl_Scheme {

private:
    static const char **names_; // registered scheme names
    static int num_schemes_; // number of registered schemes
    static int alloc_size_; // number of allocated scheme name entries

protected:
    // const char *name_; // the scheme’s name
    // protected constructor - not yet implemented
    // Fl_Scheme(const char *name);

public:

    // Static methods.
    // Some of these methods will replace the scheme related methods of class Fl,
    // for instance Fl::scheme() and Fl::is_scheme().
    // Backwards compatibility must be kept though.
    static const char **names();

    static int num_schemes() {
        if (!names_) names(); // force initialization
        return num_schemes_;
    }

    static int add_scheme_name(const char *name);

}; // class Fl_Scheme

#endif // FL_Fl_Scheme_H_

---

# Fl_Scheme_Choice.H

1 //
2 // Scheme Choice header for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 2022-2023 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16
17 ifndef FL_Fl_Scheme_Choice_H_
18 #define FL_Fl_Scheme_Choice_H_
19
20 #include <FL/Fl.H>
21 #include <FL/Fl_Scheme.H>
22 #include <FL/Fl_Choice.H>
23
24 class FL_EXPORT Fl_Scheme_Choice : public Fl_Choice {
25
26 protected:
27    static const char **names_; // registered scheme names
28    static int num_schemes_; // number of registered schemes
29    static int alloc_size_; // number of allocated scheme name entries
30
31    // const char *name_; // the scheme’s name
32
33    // protected constructor - not yet implemented
34    // Fl_Scheme(const char *name);
35
36    // Static methods.
37    // Some of these methods will replace the scheme related methods of class Fl,
38    // for instance Fl::scheme() and Fl::is_scheme().
39    // Backwards compatibility must be kept though.
40    static const char **names();
41
42    static int num_schemes() {
43        if (!names_) names(); // force initialization
44        return num_schemes_;
45    }
46
47    static int add_scheme_name(const char *name);
48
49    // Adding a scheme name must be a public static method in FLTK 1.4.0.
50    // This will later be protected or replaced by another method name.
51    static int add_scheme_name(const char *name);
52
53}; // class Fl_Scheme
54
55#endif // FL_Fl_Scheme_Choice_H_
static void scheme_cb_(Fl_Widget *, void *);

public:

Fl_Scheme_Choice(int X, int Y, int W, int H, const char *L = 0);

int handle(int event) FL_OVERRIDE;

virtual void init_value();

}; // class Fl_Scheme_Choice

#endif // FL_Fl_Scheme_Choice_H_

class FL_EXPORT Fl_Scroll : public Fl_Group {

protected: // (STR#1895)

typedef struct { int x,y,w,h; } Fl_Region_XYWH;

typedef struct {
    int l;
    int r;
    int t;
    int b;
} Fl_Region_LRTB;

typedef struct {
    int x,y,w,h;
    int pos;
    int size;
    int first;
    int total;
} Fl_Scrollbar_Data;

typedef struct {
    int scrollsize;
    Fl_Region_XYWH innerbox;
    Fl_Region_XYWH innerchild;
    Fl_Region_LRTB child;
    int hneeded;
    int vneeded;
    Fl_Scrollbar_Data hscroll;
    Fl_Scrollbar_Data vscroll;
} ScrollInfo;

void recalc_scrollbars(ScrollInfo &si) const;

protected:

int on_insert(Fl_Widget *, int) FL_OVERRIDE;

int on_move(int, int) FL_OVERRIDE;

void fix_scrollbar_order();

};
void bbox(int&, int&, int&, int&) const;
void draw() FL_OVERRIDE;

public:
Fl_Scrollbar scrollbar;
Fl_Scrollbar hscrollbar;

void resize(int X, int Y, int W, int H) FL_OVERRIDE;
int handle(int) FL_OVERRIDE;

Fl_Scroll(int X, int Y, int W, int H, const char *L = 0);
virtual ~Fl_Scroll();

enum { // values for type()
    HORIZONTAL = 1,
    VERTICAL = 2,
    BOTH = 3,
    ALWAYS_ON = 4,
    HORIZONTAL_ALWAYS = 5,
    VERTICAL_ALWAYS = 6,
    BOTH_ALWAYS = 7
};

int xposition() const { return xposition_; }
int yposition() const { return yposition_; }
void scroll_to(int, int);
void clear();

/* delete child n (by index) */
int delete_child(int n) FL_OVERRIDE;

int scrollbar_size() const {
    return scrollbar_size_;
}

protected:
void increment_cb();

#include "Fl_Slider.H"

class FL_EXPORT Fl_Scrollbar : public Fl_Slider {
  int linesize_;
  int pushed_;
  static void timeout_cb(void*);
  void increment_cb();
  protected:
  void draw() FL_OVERRIDE;
public:
Fl_Scrollbar(int X, int Y, int W, int H, const char *L = 0);
~Fl_Scrollbar();
int handle(int) FL_OVERRIDE;
int scrollbar_size() const { return scrollbar_size_; }

Generated by Doxygen
int value(int p) { return int(Fl_Slider::value((double)p)); }

int value(int pos, int windowSize, int first_line, int total_lines) {
    return scrollvalue(pos, windowSize, first_line, total_lines);
}

int linesize() const { return linesize_; }

void linesize(int i) { linesize_ = i; }

};

#endif

// Secret input header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2011 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// Fl_Secret_Input widget . */
#endif
#define Fl_Secret_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Secret_Input : public Fl_Input {
public:
    Fl_Secret_Input(int X, int Y, int W, int H, const char *l = 0);
    int handle(int) FL_OVERRIDE;
};

#endif

// Select browser header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// Fl_Select_Browser widget . */
#endif
#define Fl_Select_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Select_Browser : public Fl_Browser {
public:
    Fl_Select_Browser(int X, int Y, int W, int H, const char *L = 0);
    int handle(int) FL_OVERRIDE;
};

#endif
34.127 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

• void fl_register_images ()
  Register the known image formats.

34.127.1 Detailed Description

Fl_Shared_Image class.

34.127.2 Typedef Documentation

34.127.2.1 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:
static Fl_Image ∗check_my_image(const char ∗name,
                                  uchar ∗header,
                                  int headerlen) {
    // (test image type using header and headerlen)
    if (known) {
        // (load image data from file \p name)
        return new Fl_RGB_Image(data, ...);
    } else
        return 0;
}
// add your handler:
Fl_Shared_Image::add_handler(check_my_image);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>filename to be checked and opened if applicable</td>
</tr>
<tr>
<td>header</td>
<td>portion of the file that has already been read</td>
</tr>
<tr>
<td>headerlen</td>
<td>length of provided header data</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

valid Fl_Image or NULL.

See also

Fl_Shared_Image::add_handler()

### 34.127.3 Function Documentation

#### 34.127.3.1 fl_register_images()

```c
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().

### 34.128 Fl_Shared_Image.H

Go to the documentation of this file.

```c
1 //
2 // Shared image header file for the Fast Light Tool Kit (FLTK).  
3 //
4 // Copyright 1998-2021 by Bill Spitzak and others.  
5 //
6 // This library is free software. Distribution and use rights are outlined in 
7 // the file "COPYING" which should have been included with this file. If this 
8 // file is missing or damaged, see the license at:  
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues: 
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 #ifndef Fl_Shared_Image_H
18 # define Fl_Shared_Image_H
19 
20 # include "Fl_Image.H"
21 
22
typedef Fl_Image *(*Fl_Shared_Handler)(const char *name,
23 uchar *header,
24 int headerlen);
25
class FL_EXPORT Fl_Shared_Image : public Fl_Image {
26 
27 friend class Fl_JPEG_Image;
28 friend class Fl_PNG_Image;
29 friend class Fl_SVG_Image;
30 friend class Fl_Graphics_Driver;
31 
32 static Fl_Shared_Image **images_; // Shared images
33 static int num_images_; // Number of shared images
34 static int alloc_images_; // Allocated shared images
35 static Fl_Shared_Handler *handlers_; // Additional format handlers
36 static int num_handlers_; // Number of format handlers
37 static int alloc_handlers_; // Allocated format handlers
38 
39 const char *name_; // Name of image file
40 int original_; // Original image?
41 int refcount_; // Number of times this image has been used
42 Fl_Image *image_; // The image that is shared
43 int alloc_image_; // Was the image allocated?
44 
45 static int compare(Fl_Shared_Image **i0, Fl_Shared_Image **i1);
46 
47 static Fl_Shared_Image() ;
48 static Fl_Shared_Image(const char *n, Fl_Image *img = 0);
49 virtual ~Fl_Shared_Image();
50 
```
public:
126  const char *name() { return name_; }
127
131  int refcount() { return refcount_; }
132
140  int original() { return original_; }
141
142  void release() FL_OVERRIDE;
143  virtual void reload();
144
145  Fl_Shared_Image *as_shared_image() FL_OVERRIDE {
146    return this;
147  }
148
149  Fl_Image *copy(int W, int H) const FL_OVERRIDE;
150  Fl_Image *copy() const { return Fl_Image::copy(); }
151  void color_average(Fl_Color c, float i) FL_OVERRIDE;
152  void desaturate() FL_OVERRIDE;
153  void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) FL_OVERRIDE;
154  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
155  void uncache() FL_OVERRIDE;
156
159  static Fl_Shared_Image *find(const char *name, int W = 0, int H = 0);
160  static Fl_Shared_Image *get(const char *name, int W = 0, int H = 0);
161  static Fl_Shared_Image *get(Fl_RGB_Image *rgb, int own_it = 1);
162  static Fl_Shared_Image **images();
163  static int num_images();
164  static void add_handler(Fl_Shared_Handler f);
165  static void remove_handler(Fl_Shared_Handler f);
188  const Fl_Image *image() const { return image_; }
189
190  }; // class Fl_Shared_Image
191
192  // The following function is provided in the fltk_images library and
193  // registers all of the "extra" image file formats that are not part
194  // of the core FLTK library...
195  //
196  FL_EXPORT extern void fl_register_images();
197  //
198  #endif // !Fl_Shared_Image_H

34.129 Fl_Shortcut_Button.H

/*
* This file is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:
*/
//
// https://www.fltk.org/COPYING.php

#endif // !Fl_Shortcut_Button_H
#define Fl_Shortcut_Button_H

#include <FL/Fl_Button.H>

class FL_EXPORT Fl_Shortcut_Button : public Fl_Button {
private:
  bool hot_, pre_hot_, default_set_, handle_default_button_;  
  Fl_Shortcut pre_esc_;  
  Fl_Shortcut defaultShortcut_;  
protected:
  FL_SHORTCUT shortcut_value;
  void do_end_hot_callback();
  int handle(int) FL_OVERRIDE;
  void draw() FL_OVERRIDE;
public:
  Fl_Shortcut_Button(int X, int Y, int W, int H, const char* l = 0);
  void value(Fl_Shortcut shortcut);
34.130  fl_show_colormap.H File Reference

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

Functions

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

34.130.1 Detailed Description

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

34.131  fl_show_colormap.H

Go to the documentation of this file.

1 //
2 // Colormap picker header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 #ifndef fl_show_colormap_H
17 #define fl_show_colormap_H
18
19 /* doxygen comment here to avoid exposing ColorMenu in fl_show_colormap.cxx
20 */
21
22 FL_EXPORT Fl_Color fl_show_colormap(Fl_Color oldcol);
23
24 #endif

34.132  fl_show_input.H

1 //
2 // Standard input dialog header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 #ifndef fl_show_input_H
17 #define fl_show_input_H
18
19 /* doxygen comment here to avoid exposing ColorMenu in fl_show_input.cxx
20 */
21
22 FL_EXPORT Fl_Color fl_show_input( Fl_Menu Bar * box, Fl_Color oldcol);
23
24 #endif
34.133 Fl_Simple_Counter.H

1 //
2 // Simple counter header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 /*\file
17 Fl_Simple_Counter widget . */
18
19 #ifndef Fl_Simple_Counter_H
20 #define Fl_Simple_Counter_H
21 #include "Fl_Counter.H"
22
23 class FL_EXPORT Fl_Simple_Counter : public Fl_Counter {
24 public:
25 Fl_Simple_Counter(int X,int Y,int W,int H, const char *L = 0);
26 }
27
28 #endif
29
30

34.134 Fl_Simple_Terminal.H

1 //
2 // A simple terminal widget for Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2011 by Bill Spitzak and others.
5 // Copyright 2017 by Greg Ercolano.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:
14 //
15 // https://www.fltk.org/bugs.php
16 //
17 /*\file
18 Fl_Simple_Terminal widget . */
19
20 #ifndef Fl_Simple_Terminal_H
21 #define Fl_Simple_Terminal_H
22 #include "Fl_Export.H"
23 #include <FL/Fl_Text_Display.H>
24
25 class FL_EXPORT Fl_Simple_Terminal : public Fl_Text_Display {
26 protected:
27 Fl_Text_Buffer *buf; // text buffer
28 Fl_Text_Buffer *sbuf; // style buffer
29
30 private:
31 // Private class to handle parsing ESC sequences
32 // Holds all state information for parsing esc sequences,
33 // so sequences can span multiple block read(2) operations, etc.
34 //
35 class Fl_Escape_Seq {
36 public:
37 static const int maxbuf = 80;
38 static const int maxvals = 10;
39 // Return codes
40 static const int success = 0; // operation succeeded
41 static const int fail = -1; // operation failed
42 static const int completed = 1; // multi-step operation completed successfully
43
44 private:
45 char esc_mode_; // escape parsing mode state
46 char buf[2][maxbuf]; // escape sequence being parsed
47 char *bufp_; // parsing ptr into buf[]
48 char *bufendp_; // end of buf[] (ptr to last valid buf char)
49 char *valbufp_; // pointer to first char in buf of integer being parsed

Generated by Doxygen
int vals_[maxvals]; // value array for parsing #’s in ESC[#;#;#...
int val_;   // parsing index into vals[], 0 if none

int append_buf(char c);
int append_val();

public:
    Fl_Escape_Seq();
    void reset();
    char esc_mode() const;
    void esc_mode(char val);
    int total_vals() const;
    int val(int i) const;
    bool parse_in_progress() const;
    int parse(char c);
};

private:
    int history_lines_; // max lines allowed in screen history
    bool stay_at_bottom_; // lets scroller chase last line in buffer
    // scroll management
    int lines_; // #lines in buffer (optimization: Fl_Text_Buffer slow to calc this)
    bool scrollaway_; // true when user changed vscroll away from bottom
    bool scrolling_; // true while scroll callback active

    // Fl_Text_Display's callback-data
    Fl_Callback *orig_vscroll_cb_;
    void *orig_vscroll_data_;

    // Style table
    const Fl_Text_Display::Style_Table_Entry *stable_; // the active style table
    int stable_size_; // active style table size (in bytes)
    int normal_style_index_; // "normal" style used by *\033[0m* reset sequence
    int current_style_index_; // current style used for drawing text
    char current_style_; // current ‘style char’ (e.g. ’A’ = first style entry)
    // ANSI escape seq
    Fl_Escape_Seq escseq; // escape sequence state handler
    bool ansi_; // enables ANSI sequences
    bool ansi_show_unknown_; // show ‘¿’ for unknown ESC sequences (default: off)

    // String parsing vars initialized/used by append(), used by handle_backspace() etc.
    char *ntm_; // new text memory (ntm) - malloc()ed by append() for output text
    char *ntp_; // new text ptr (ntp) - points into ntm buffer
    char *nsm_; // new style memory (nsm) - malloc()ed by append() for output style
    char *nsp_; // new style ptr (nsp) - points into nsm buffer

public:
    Fl_Simple_Terminal(int X,int Y,int W,int H,const char *l=0);
    ~Fl_Simple_Terminal();

    // Terminal options
    void stay_at_bottom(bool);
    bool stay_at_bottom() const;
    void history_lines(int);
    int history_lines() const;
    void ansi(bool val);
    bool ansi() const;
    void ansi_show_unknown(bool val);
    bool ansi_show_unknown() const;
    void style_table(Fl_Text_Display::Style_Table_Entry *stable, int stable_size, int normal_style_index=0);
    const Fl_Text_Display::Style_Table_Entry *style_table() const;
    int style_table_size() const;
    void normal_style_index(int);
    int normal_style_index() const;
    void current_style_index(int);
    int current_style_index() const;
    int current_style() const;

    // Terminal text management
    void append(const char *s, int len=-1);
    void text(const char *s, int len=-1);
    const char * text() const;
    void printf(const char *fmt, ...);
    void vprintf(const char *fmt, va_list ap);
    void clear();
    void remove_lines(int start, int count);

    // Methods blocking public access to the subclass
    // These are subclass methods that would give unexpected
    // results if used. By making them private, we effectively
    // "block" them.

    private:
        void insert(const char *); { }

protected:
34.135 Fl_Single_Window.H

1 // Single-buffered window header file for the Fast Light Tool Kit (FLTK).
2 // Copyright 1998-2015 by Bill Spitzak and others.
3 // This library is free software. Distribution and use rights are outlined in
4 // the file "COPYING" which should have been included with this file. If this
5 // file is missing or damaged, see the license at:
6 // https://www.fltk.org/COPYING.php
7 // Please see the following page on how to report bugs and issues:
8 // https://www.fltk.org/bugs.php
9 //
10 #ifndef Fl_Single_Window_H
11 #define Fl_Single_Window_H
12
13 class FL_EXPORT Fl_Single_Window : public Fl_Window {
14 public:
15  void show() FL_OVERRIDE;
16  void show(int a, char **b) {Fl_Window::show(a,b);}
17  Fl_Single_Window(int W, int H, const char *l=0);
18  Fl_Single_Window(int X, int Y, int W, int H, const char *l=0);
19
20 #endif

34.136 Fl_Slider.H

1 // Slider header file for the Fast Light Tool Kit (FLTK).
2 // Copyright 1998-2010 by Bill Spitzak and others.
3 // This library is free software. Distribution and use rights are outlined in
4 // the file "COPYING" which should have been included with this file. If this
5 // file is missing or damaged, see the license at:
6 // https://www.fltk.org/COPYING.php
7 // Please see the following page on how to report bugs and issues:
8 // https://www.fltk.org/bugs.php
9 //
10 #ifndef Fl_Slider_H
11 #define Fl_Slider_H
12
13 #ifndef Fl_Valuator_H
14 #include "Fl_Valuator.H"
15 #endif

17 // Fl_Slider widget . */
18 #ifndef FL_Slider_H
19 #define FL_Slider_H
20 #define FL_Valuator_H
21
22 #ifndef FL_Valuator_H
23 #include "FL_Valuator.H"
24 #endif
25 #endif
26
27 // Fltk
28 void draw() FL_OVERRIDE;
29 // Internal methods
30 void enforce_stay_at_bottom();
31 void enforce_history_lines();
32 void vscroll_ch2(Fl_Widget*, void*);
33 static void vscroll_ch(Fl_Widget*, void*);
34 void backspace_buffer(unsigned int count);
35 void handle_backspace();
36 void append_ansi(const char *, int len);
37 void unknown_escape();
38 };
39
40 #endif
// values for type(), lowest bit indicate horizontal:
#define FL_VERT_SLIDER 0
#define FL_HOR_SLIDER 1
#define FL_VERT_FILL_SLIDER 2
#define FL_HOR_FILL_SLIDER 3
#define FL_VERT_NICE_SLIDER 4
#define FL_HOR_NICE_SLIDER 5

class FL_EXPORT Fl_Slider : public Fl_Valuator {
  float slider_size_;  
  uchar slider_;  
  void _Fl_Slider(); 
  void draw_bg(int, int, int, int); 
protected:  
  // these allow subclasses to put the slider in a smaller area: 
  void draw(int, int, int, int); 
  int handle(int, int, int, int, int); 
  void draw() FL_OVERRIDE; 
public:  
  int handle(int) FL_OVERRIDE; 
  Fl_Slider(int X,int Y,int W,int H, const char *L = 0); 
  Fl_Slider(uchar t,int X,int Y,int W,int H, const char *L); 
  int scrollvalue(int pos,int size,int first,int total); 
  void bounds(double a, double b); 
  float slider_size()const {return slider_size_;} 
  void slider_size(double v); 
  Fl_Boxtype slider()const {return (Fl_Boxtype)slider_;} 
  void slider(Fl_Boxtype c) {slider_ = c;} 
};

34.137 Fl_Spinner.H

/* file Fl_Spinner widget */

#include <FL/Enumerations.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Repeat_Button.H>

class FL_EXPORT Fl_Spinner : public Fl_Group {
  double value_;  
  double minimum_;  
  double maximum_;  
  double step_;  
  const char *format_;  
  int wrap_;  
private:  
  static void sb_cb(Fl_Widget *w, Fl_Spinner *sb); // internal callback 
  void update();  
};
This class works like Fl_Input but ignores FL_Up and FL_Down key presses so they are handled by its parent, the Fl_Spinner widget. // See STR #2989.

class FL_EXPORT Fl_Spinner_Input : public Fl_Input {
public:

Fl_Spinner_Input(int X, int Y, int W, int H) : Fl_Input(X, Y, W, H) {}  
int handle(int event) FL_OVERRIDE; // implemented in src/Fl_Spinner.cxx  

Fl_Spinner_Input input_; // Input field for the value  
Fl_Repeat_Button up_button_, // Up button  
down_button_;  

void draw() FL_OVERRIDE;

protected:

// This class works like Fl_Input but ignores FL_Up and FL_Down key presses so they are handled by its parent, the Fl_Spinner widget.  
// See STR #2989.

Fl_Spinner_input input_; // Input field for the value  
Fl_Repeat_Button up_button_, // Up button  
down_button_;  

void draw() FL_OVERRIDE;

public:

// Constructor  
Fl_Spinner(int X, int Y, int W, int H, const char *L = 0);  
// Event handling  
int handle(int event) FL_OVERRIDE;  
// Resize group and subwidgets  
void resize(int X, int Y, int W, int H) FL_OVERRIDE;

const char *format() const { return (format_); }  
void format(const char *f) { format_ = f; update(); }  
double maximum() const { return (maximum_); }  
void maximum(double m) { maximum_ = m; }  
double minimum() const { return (minimum_); }  
void minimum(double m) { minimum_ = m; }  
void range(double a, double b) { minimum_ = a; maximum_ = b; }  

// Sets the amount to change the value when the user clicks a button.  
// Docs in src/Fl_Spinner.cxx  
void step(double s);

double step() const { return (step_); }  

void wrap(int set) { wrap_ = set ? 1 : 0; }  

int wrap() const { return wrap_; }  

doctype(uchar v);

double value() const { return (value_); }  
void value(double v) { value_ = v; update(); }  

void type() const { return (input_.type()); }  
uchar type() const { return (input_.type()); }  
double value() const { return (value_); }  
void value(double v) { value_ = v; update(); }  

Fl_Color textcolor() const { return (input_.textcolor()); }  
void textcolor(Fl_Color c) { input_.textcolor(c); }  

Fl_Font textfont() const { return (input_.textfont()); }  
void textfont(Fl_Font f) { input_.textfont(f); }  

Fl_Fontsize textsize() const { return (input_.textsize()); }  
void textsize(Fl_Fontsize s) { input_.textsize(s); }  

// Sets the numeric representation in the input field.  
// Docs see src/Fl_Spinner.cxx  
void type(uchar v);

doctype(uchar v);

doctype(double value);

Generated by Doxygen
34.138  Fl_String.H File Reference

Basic Fl_String class for FLTK.
#include "Fl_Export.H"

Classes

• class Fl_String
   Fl_String is the basic string class for FLTK.

Functions

• Fl_String operator+ (const Fl_String &lhs, const char *rhs)
  Concatenate two strings.
• Fl_String operator+ (const Fl_String &lhs, const Fl_String &rhs)
  Concatenate two strings.
• bool operator== (const Fl_String &lhs, const Fl_String &rhs)
  Compare two strings.

34.138.1 Detailed Description

Basic Fl_String class for FLTK.

34.138.2 Function Documentation

34.138.2.1 operator+() [1/2]

Fl_String operator+ (
   const Fl_String &lhs,
   const char * rhs )

Concatenate two strings.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>lhs</th>
<th>first string</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rhs</td>
<td>second C-style string</td>
</tr>
</tbody>
</table>

Returns

self

34.138.2.2 operator+() [2/2]

Fl_String operator+ (
   const Fl_String &lhs,
   const Fl_String & rhs )

Concatenate two strings.
34.138.2.3 `operator==()`

```cpp
bool operator== (  
    const Fl_String & lhs,  
    const Fl_String & rhs )
```

Compare two strings.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>lhs</th>
<th>first string</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rhs</td>
<td>second string</td>
</tr>
</tbody>
</table>

Returns

true if strings are the same size and have the same content
/**
 * Fl_String() ;
 * Fl_String(const Fl_String &str);
 * Fl_String(const char *cstr);
 * Fl_String(const char *str, int size);
 * ~Fl_String();
 * Fl_String& operator=(const Fl_String &str);
 * Fl_String& operator=(const char *cstr);
 * Fl_String &assign(const Fl_String &str);
 * Fl_String &assign(const char *cstr);
 * Fl_String &assign(const char *str, int size);
 *
 * // ---- Element Access
 * char at(int pos) const;
 * char operator[](int n) const;
 * const char *data() const;
 * char *data();
 * const char *c_str() const;
 * void push_back(char c);
 * void pop_back();
 * Fl_String &append(const char *src, int n_ins=npos);
 * Fl_String &append(const Fl_String &src);
 * Fl_String &append(char c);
 * Fl_String &operator+=(const char *src);
 * Fl_String &operator+=(const Fl_String &src);
 * Fl_String &operator+=(char c);
 * Fl_String &replace(int at, int n_del, const char *src, int n_ins=npos);
 * Fl_String &replace(int at, int n_del, const Fl_String &src);
 * Fl_String substr(int pos=0, int n=npos) const;
 * void resize(int n);
 *
 * // --- Non Standard
 * int strlen() const;
 * void debug(const char *info = 0) const;
 * void hexdump(const char *info = 0) const;
 * } // class Fl_String
 */

#include "Fl_Export.H"
#include "fl_types.h"

Public header for FLTK's own platform agnostic string handling.
#include "Fl_Export.H"
#include "fl_types.h"

Functions

- char * fl_strdup (const char *s)

  Cross platform interface to POSIX function strdup().

34.140.1 Detailed Description

Public header for FLTK's own platform agnostic string handling.
# fl_string_functions.h

Go to the documentation of this file.

```c
/*
 * Platform agnostic string portability functions for the Fast Light Tool Kit (FLTK).
 * Copyright 2020-2022 by Bill Spitzak and others.
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file. If this
 * file is missing or damaged, see the license at:
 * https://www.fltk.org/COPYING.php
 * Please see the following page on how to report bugs and issues:
 * https://www.fltk.org/bugs.php
 */
#ifndef _FL_fl_string_functions_h_
#define _FL_fl_string_functions_h_

#include "Fl_Export.H"
#include "fl_types.h"

#ifdef __cplusplus
extern "C" {
#endif

FL_EXPORT char* fl_strdup(const char *s);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* _FL_fl_string_functions_h_ */
```

## FL_SVG_File_Surface.H

```c
/*****************************/
#endif /* Fl_SVG_File_Surface_H */
```
34.143  

**FL_SVG_Image.H**

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

class FL_EXPORT Fl_SVG_Image : public Fl_RGB_Image {
private:
  typedef struct {
    NSVGimage * svg_image;
    int ref_count;
  } counted_NSVGimage;
  counted_NSVGimage * counted_svg_image_;
  bool rasterized_;
  int raster_w_, raster_h_;;
  bool to_desaturate_;;
  Fl_Color average_color_;;
  float average_weight_;;
  float svg_scaling_(int W, int H);
  void rasterize_(int W, int H);
  void cache_size_(int &width, int &height) FL_OVERRIDE;
  void init_(const char *name, const unsigned char *filedata, size_t length);
  Fl_SVG_Image(const Fl_SVG_Image *source);
public:
  bool proportional;
  Fl_SVG_Image(const char *filename);
  Fl_SVG_Image(const char *sharedname, const char *svg_data);
  Fl_SVG_Image(const char *sharedname, const unsigned char *svg_data, size_t length);
  virtual ~Fl_SVG_Image();
  Fl_Image *copy(int W, int H) const FL_OVERRIDE;
  Fl_Image *copy() const {
    return Fl_Image::copy();
  }
  void resize(int width, int height);
  void desaturate() FL_OVERRIDE;
  void color_average(Fl_Color c, float i) FL_OVERRIDE;
  void draw(int X, int Y, int W, int H, int ex = 0, int cy = 0) FL_OVERRIDE;
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  Fl_SVG_Image *as_svg_image() FL_OVERRIDE { return this; }
  void normalize() FL_OVERRIDE;
};
```

34.144  

**FL_Sys_Menu_Bar.H** File Reference

Definition of class Fl_Sys_Menu_Bar.

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

### Classes

- **class Fl_Sys_Menu_Bar**

  A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

### Variables

- **Fl_Sys_Menu_Bar * fl_sys_menu_bar**
The system menu bar.

34.144.1 Detailed Description

Definition of class Fl_Sys_Menu_Bar.

34.145 Fl_Sys_Menu_Bar.H

Go to the documentation of this file.

#include <FL/Fl_Menu_Bar.H>

class Fl_Sys_Menu_Bar_Driver;

class FL_EXPORT Fl_Sys_Menu_Bar : public Fl_Menu_Bar {
public:
typedef enum {
  no_window_menu = 0,
  tabbing_mode_none,
  tabbing_mode_automatic,
  tabbing_mode_preferred
} window_menu_style_enum;
Fl_Sys_Menu_Bar(int x,int y,int w,int h,const char *l=0);
virtual ~Fl_Sys_Menu_Bar();
const Fl_Menu_Item *menu() const {return Fl_Menu_::menu();}
void menu(const Fl_Menu_Item *m);
void update() FL_OVERRIDE;
int add(const char * label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0);
int add(const char * label, const char* shortcut, Fl_Callback* cb, void *user_data=0, int flags=0) {
  return add(label, fl_old_shortcut(shortcut), cb, user_data, flags);
}
int add(const char * str);
int insert(int index, const char * label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0) {
  return insert(index, label, fl_old_shortcut(shortcut), cb, user_data, flags);
}
void remove(int n);
void replace(int index, const char *name);
void clear();
int clear_submenu(int index);
int mode(int i) const { return Fl_Menu_::mode(i); }
void shortcut(int i, int s);
void setonly (Fl_Menu_Item *item);
static void about(Fl_Callback *cb, void *data);
static window_menu_style_enum window_menu_style();
static void window_menu_style(window_menu_style_enum style);
static void create_window_menu();
};

extern Fl_Sys_Menu_Bar *fl_sys_menu_bar;

#endif // Fl_Sys_Menu_Bar_H

Generated by Doxygen
// Fl_Table -- A table widget for the Fast Light Tool Kit (FLTK).
// Copyright 2002 by Greg Ercolano.
// Copyright (c) 2004 O'ksi'D
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php

#ifndef _FL_TABLE_H
#define _FL_TABLE_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Scroll.H>
#include <FL/Fl_Int_Vector.H>

class FL_EXPORT Fl_Table : public Fl_Group {
public:
    enum TableContext {
        CONTEXT_NONE = 0,
        CONTEXT_STARTPAGE = 0x01,
        CONTEXT_ENDPAGE = 0x02,
        CONTEXT_ROW_HEADER = 0x04,
        CONTEXT_COL_HEADER = 0x08,
        CONTEXT_CELL = 0x10,
        CONTEXT_TABLE = 0x20,
        CONTEXT_RC_RESIZE = 0x40
    };

private:
    int _rows, _cols; // total rows/cols
    int _row_header_w; // width of row header
    int _col_header_h; // height of column header
    int _row_position; // last row_position set (not necessarily == toprow!)
    int _col_position; // last col_position set (not necessarily == leftcol!)
    char _row_header; // row header enabled?
    char _col_header; // col header enabled?
    char _row_resize; // row resizing enabled?
    char _col_resize; // col resizing enabled?
    int _row_resize_min; // row minimum resizing height (default=1)
    int _col_resize_min; // col minimum resizing width (default=1)
    // OPTIMIZATION: partial row/column redraw variables
    int _redraw_toprow;
    int _redraw_botrow;
    int _redraw_leftcol;
    int _redraw_rightcol;
    Fl_Color _row_header_color;
    Fl_Color _col_header_color;
    Fl_Cursor _last_cursor; // last mouse cursor before changed to 'resize' cursor
    // handle() state variables.
    // Put here instead of local statics in handle(), so more
    // than one Fl_Table can exist without crosstalk between them.
    int _resizing_col; // column being dragged
    int _resizing_row; // row being dragged
    int _dragging_x; // starting x position for horiz drag
    int _dragging_y; // starting y position for vert drag
    int _last_row; // last row we FL_PUSH'ed
};
void _redraw_cell(TableContext context, int R, int C);
void _start_auto_drag();
void _stop_auto_drag();
void _auto_drag_cb();
static void _auto_drag_cb2(void *d);

protected:
enum ResizeFlag {
  RESIZE_NONE = 0,
  RESIZE_COL_LEFT = 1,
  RESIZE_COL_RIGHT = 2,
  RESIZE_ROW_ABOVE = 3,
  RESIZE_ROW_BELOW = 4
};

int table_w;
int table_h;
int toprow;
int botrow;
int leftcol;
int rightcol;

// selection
int current_row;
int current_col;
int select_row;
int select_col;

// OPTIMIZATION: Precomputed scroll positions for the toprow/leftcol
int toprow_scrollpos;
int leftcol_scrollpos;

// Data table’s inner dimension
int tix;
int tiy;
int tiw;
int tih;

// Data table’s outer dimension
int tox;
int toy;
int tow;
int toh;

// Table widget’s inner dimension
int wix;
int wiy;
int wiw;
int wiw;

Fl_Scroll *table;
Fl_Scrollbar *vscrollbar;
Fl_Scrollbar *hscrollbar;

// Fltk
int handle(int e) FL_OVERRIDE; // fltk handle() FL_OVERRIDE
void recalc_dimensions();
void table_resized(); // table resized; recalc
void table_scrolled(); // table scrolled; recalc
int find_cell(TableContext context, int R, int C, ResizeFlag &resizeflag);
int row_col_clamp(TableContext context, int &R, int &C);

virtual void draw_cell(TableContext context, int R=0, int C=0,
int X=0, int Y=0, int W=0, int H=0);

int is_fltk_container() { // does table contain fltk widgets?
return( Fl_Group::children() > 3 ); // (ie. more than box and 2 scrollbars?)
}

// overridden by deriving class

static void scroll_cb(Fl_Widget *, void *);

int row_scroll_position(int row); // find scroll position of row (in pixels)
int col_scroll_position(int col); // find scroll position of col (in pixels)

// overridden by deriving class

virtual void draw_cell(TableContext context, int R=0, int C=0,
int X=0, int Y=0, int W=0, int H=0);

{ (void)context; (void)R; (void)C; (void)X; (void)Y; (void)W; (void)H; }

// overridden by deriving class

long row_scroll_position(int row); // find scroll position of row (in pixels)
long col_scroll_position(int col); // find scroll position of col (in pixels)

int is_fltk_container() { // does table contain fltk widgets?
return( Fl_Group::children() > 3 ); // (ie. more than box and 2 scrollbars?)
}

static void scroll_cb(Fl_Widget *, void *); // h/v scrollbar callback
void damage_zone(int r1, int c1, int r2, int c2, int r3 = 0, int c3 = 0);

void redraw_range(int topRow, int botRow, int leftCol, int rightCol) {
    if (_redraw_toprow == -1) {
        // Initialize redraw range
        _redraw_toprow = topRow;
        _redraw_botrow = botRow;
        _redraw_leftcol = leftCol;
        _redraw_rightcol = rightCol;
    } else {
        // Extend redraw range
        if (topRow < _redraw_toprow) _redraw_toprow = topRow;
        if (botRow > _redraw_botrow) _redraw_botrow = botRow;
        if (leftCol < _redraw_leftcol) _redraw_leftcol = leftCol;
        if (rightCol > _redraw_rightcol) _redraw_rightcol = rightCol;
    }
    // Indicate partial redraw needed of some cells
    damage(FL_DAMAGE_CHILD);
}

// draw() has to be protected per FLTK convention (was public in 1.3.x)
void draw() FL_OVERRIDE;

public:
    Fl_Table(int X, int Y, int W, int H, const char *l=0);
    ~Fl_Table();

    virtual void clear() {
        rows(0);
        cols(0);
        table->clear();
    }

    // todo: add topline(), middleline(), bottomline()
    inline void table_box(Fl_Boxtype val) {
        table->box(val);
        table_resized();
    }

    inline Fl_Boxtype table_box() {
        return(table->box());
    }

    virtual void rows(int val); // set/get number of rows
    inline int rows() {
        return(_rows);
    }

    virtual void cols(int val); // set/get number of columns
    inline int cols() {
        return(_cols);
    }

    inline void visible_cells(int& r1, int& r2, int& c1, int& c2) {
        r1 = toprow;
        r2 = botrow;
        c1 = leftcol;
        c2 = rightcol;
    }

    int is_interactive_resize() {
        return(_resizing_row != -1 || _resizing_col != -1);
    }

    inline int row_resize() {
        return(_row_resize);
    }

    inline int col_resize() {
        return(_col_resize);
    }

    inline int col_resize_min() { // column minimum resizing width
        return(_col_resize_min);
    }

    virtual void row_resize(int flag) { // enable row resizing
        _row_resize = flag;
    }

    virtual void col_resize(int flag) { // enable col resizing
        _col_resize = flag;
    }

    inline int col_resize_min() { // column minimum resizing width
        return(_col_resize_min);
    }

    Generated by Doxygen
void col_resize_min(int val) {
    _col_resize_min = (val < 1) ? 1 : val;
}

inline int row_resize_min() {  // column minimum resizing width
    return(_row_resize_min);
}

void row_resize_min(int val) {
    _row_resize_min = (val < 1) ? 1 : val;
}

inline int row_header() {  // set/get row header enable flag
    return(_row_header);
}

void row_header(int flag) {
    _row_header = flag;
    table_resized();
    redraw();
}

inline int row_header_width() {  // set/get row header width
    return(_row_header_w);
}

inline void row_header_width(int width) {  // set/get row header width
    _row_header_w = width;
    table_resized();
    redraw();
}

inline int row_header_color() {  // set/get row header color
    return(_row_header_color);
}

inline void row_header_color(Fl_Color val) {  // set/get row header color
    _row_header_color = val;
    redraw();
}

void row_height(int row, int height);  // set/get row height

inline int row_height(int row) {
    return((row<0 || row>=int(_rowheights.size())) ? 0 : _rowheights[row]);
}

void col_width(int col, int width);  // set/get a column’s width

inline int col_width(int col) {
    return((col<0 || col>=int(_colwidths.size())) ? 0 : _colwidths[col]);
}

inline void col_width(int col) {  // set/get a column’s width
    _col_width = flag;
    table_resized();
    redraw();
}

inline int col_width(int col) {  // set/get a column’s width
    return(_col_width);
}

inline void col_header_height(int height) {  // set/get col header height
    _col_header_h = height;
    table_resized();
    redraw();
}

inline int col_header_height() {  // set/get col header height
    return(_col_header_h);
}

void row_height_all(int height) {  // set all row/col heights
    for (int r=0; r<rows(); r++) {
        row_height(r, height);
    }
}

Generated by Doxygen
void col_width_all(int width) {
    for (int c=0; c<cols(); c++) {
        col_width(c, width);
    }
}

int row_position() {
    return(_row_position);
}

int col_position() {
    return(_col_position);
}

inline int top_row() {
    return(row_position());
}

int is_selected(int r, int c); // selected cell
void get_selection(int &row_top, int &col_left, int &row_bot, int &col_right);
void set_selection(int row_top, int col_left, int row_bot, int col_right);

void move_cursor(int R, int C, int shiftselect);

int move_cursor(int R, int C);

void resize(int X, int Y, int W, int H) FL_OVERRIDE; // fltk resize() FL_OVERRIDE

// Child group management

void add(Fl_Widget& wgt) {
    table->add(wgt);
    if (table->children() > 2) {
        table->show();
    } else {
        table->hide();
    }
}

void add(Fl_Widget* wgt) {
    add( *wgt);
}

void insert(Fl_Widget& wgt, int n) {
    table->insert(wgt,n);
}

void insert(Fl_Widget& wgt, Fl_Widget * w2) {
    table->insert(wgt,w2);
}

void remove(Fl_Widget& wgt) {
    table->remove(wgt);
}

// (doxygen will substitute Fl_Group's docs here)
void begin() {
    table->begin();
}

// (doxygen will substitute Fl_Group's docs here)
void end() {
    table->end();
    // HACK: Avoid showing Fl_Scroll seems to erase screen
    // causing unnecessary flicker, even if its box() is FL_NO_BOX.
}

if (table->children() > 2) {

table->show();  
} else {  
  Fl_Group::current(Fl_Group::parent());  
}

Fl_Widget *const* array() {  
  return(table->array());  
}

Fl_Widget *child(int n)const {  
  return(table->child(n));  
}

int children()const {  
  return(table->children()-2); // -2: skip Fl_Scroll's h/v scrollbar widgets
}

// (doxygen will substitute Fl_Group's docs here)
int find(const Fl_Widget *wgt)const {  
  return(table->find(wgt));  
}

// (doxygen will substitute Fl_Group's docs here)
int find(const Fl_Widget &wgt)const {  
  return(table->find(wgt));  
}

// CALLBACKS
int callback_row() {  
  return(_callback_row);  
}

int callback_col() {  
  return(_callback_col);  
}

TableContext callback_context() {  
  return(_callback_context);  
}

void do_callback(TableContext context, int row, int col) {  
  _callback_context = context;  
  _callback_row = row;  
  _callback_col = col;  
  Fl_Widget::do_callback();  
}

void when(Fl_When flags);  

// ifdef FL_DOXYGEN
void callback(Fl_Widget *, void*);  
// endif FL_DOXYGEN

int scrollbar_size()const {  
  return(_scrollbar_size);  
}

void scrollbar_size(int newSize) {  
  if ( newSize != _scrollbar_size ) redraw();  
  _scrollbar_size = newSize;  
}

void tab_cell_nav(int val) {  
  if ( val ) flags_ |= TABCELLNAV;  
  else flags_ &= ~TABCELLNAV;  
}

int tab_cell_nav()const {  
  return((flags_ & TABCELLNAV) ? 1 : 0);  
}

// ifdef FL_DOXYGEN
// endif FL_DOXYGEN

#endif /*_FL_TABLE_H*/

// _FL_TABLE_ROW_H

1 //
2
3 ifndef _FL_TABLE_ROW_H
4 #define _FL_TABLE_ROW_H

Generated by Doxygen
Fl_Table_Row -- A row oriented table widget for the Fast Light Tool Kit (FLTK).

A class specializing in a table of rows. Handles row-specific selection behavior.

Copyright 2002 by Greg Ercolano.

This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

#include <FL/Fl_Table.H>

class FL_EXPORT Fl_Table_Row : public Fl_Table { public:

eenum TableRowSelectMode {
  SELECT_NONE, // no selection allowed
  SELECT_SINGLE, // single row selection
  SELECT_MULTI // multiple row selection (default)
};

private:
  // An STL-ish vector without templates
  class FL_EXPORT CharVector {
  char *arr;
  int _size;
  void init() {
    arr = 0;
    _size = 0;
  }
  void copy(char *newarr, int newsize);

  CharVector() { // CTOR
    init();
  }
  ~CharVector(); // DTOR
  CharVector(CharVector&o) { // COPY CTOR
    init();
    copy(o.arr, o._size);
  }
  CharVector& operator=(CharVector&o) { // ASSIGN
    init();
    copy(o.arr, o._size);
    return(*this);
  }

  char operator[](int x) const {
    return(arr[x]);
  }

  int size() {
    return(_size);
  }

  void size(int count);
  char pop_back() {
    char tmp = arr[_size-1];
    _size--;
    return(tmp);
  }
  void push_back(char val) {
    int x = _size;
    size(_size+1);
    arr[x] = val;
  }

  char back() {
    return(arr[_size-1]);
  }

  CharVector _rowselect; // selection flag for each row

  // handle() state variables.
  int _dragging_select; // dragging out a selection?
  int _last_row;
  int _last_y; // last event’s Y position
}
```cpp
#!_FL_TABLE_ROW_H

# ifndef Fl_Table_Row_H
# define Fl_Table_Row_H

int _last_push_x; // last PUSH event's X position
int _last_push_y; // last PUSH event's Y position

TableRowSelectMode _selectmode;

protected:
int handle(int event) FL_OVERRIDE;
int find_cell(TableContext context, // find cell's x/y/w/h given r/c
                   int R, int C, int &X, int &Y, int &W, int &H) {
  return(Fl_Table::find_cell(context, R, C, X, Y, W, H));
}

public:
Fl_Table_Row(int X, int Y, int W, int H, const char *l=0) : Fl_Table(X,Y,W,H,l) {
  _dragging_select = 0;
  _last_row = -1;
  _last_y = -1;
  _last_push_x = -1;
  _last_push_y = -1;
  _selectmode = SELECT_MULTI;
}

~Fl_Table_Row() { }

void rows(int val) FL_OVERRIDE; // set number of rows
int rows() { // get number of rows
  return(Fl_Table::rows());
}

void type(TableRowSelectMode val); // set selection mode
TableRowSelectMode type()const { // get selection mode
  return(_selectmode);
}

int row_selected(int row); // is row selected? (0=no, 1=yes, -1=range err)
int select_row(int row, int flag=1); // select state for row: flag:0=off, 1=on, 2=toggle
                      // returns: 0=no change, 1=changed, -1=range err
void select_all_rows(int flag=1); // all rows to a known state
void clear() FL_OVERRIDE {
  rows(0); // implies clearing selection
cols(0);
  Fl_Table::clear(); // clear the table
}

# endif /*_FL_TABLE_ROW_H*/
```

---

34.148 **FL_Tabs.H**

1 // 2 // Tab header file for the Fast Light Tool Kit (FLTK).
3 // 4 // Copyright 1998-2023 by Bill Spitzak and others.
5 // 6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 // 10 //  https://www.fltk.org/COPYING.php
11 // 12 // Please see the following page on how to report bugs and issues:
13 // 14 //  https://www.fltk.org/bugs.php
15 // 16 // 17 /* \file
18 Fl_Tabs widget . */
19 #ifndef Fl_Tabs_H
20 #define Fl_Tabs_H
21 #include "Fl_Group.H"
22 #include "Fl_Menu_Item.H"
23 struct Fl_Menu_Item;
24 class FL_EXPORT Fl_Tabs : public Fl_Group {
25  Fl_Widget *push_
26 protected:
27```
int overflow_type;
int tab_offset;
int *tab_pos; // array of x-offsets of tabs per child + 1
int *tab_width; // array of widths of tabs per child + 1
int *tab_flags; // array of tab flag of tabs per child + 1
int tab_count; // array size
Fl_Alg tab_align_; // tab label alignment
int has_overflow_menu;
Fl_Menu_Item * overflow_menu;

void check_overflow_menu();
void handle_overflow_menu();
void draw_overflow_menu_button();

int on_insert(Fl_Widget *, int) FL_OVERRIDE;
int on_move(int, int) FL_OVERRIDE;
void on_remove(int) FL_OVERRIDE;
void resize(int, int, int, int) FL_OVERRIDE;

virtual void redraw_tabs();
virtual int tab_positions(); // allocate and calculate tab positions
virtual void clear_tab_positions();
virtual void draw_tab(int x1, int x2, int W, int H, Fl_Widget * o, int flags, int sel);
virtual int tab_height();
virtual int hit_close(Fl_Widget *o, int event_x, int event_y);

doctor

int on_overflow(Fl_Widget *, int) FL_OVERRIDE;
Fl_Widget * value();
int value(Fl_Widget *);

Fl_Widget * push()const {return push_};
int push(Fl_Widget *);

virtual Fl_Widget * which(int event_x, int event_y);

void client_area(int &rx, int &ry, int &rw, int &rh, int tabh=0);

void tab_align(Fl_Alg a) {tab_align_ = a;}

Fl_Alg tab_align()const {return tab_align_;}

enum {
  OVERFLOW_COMPRESS = 0,
  OVERFLOW_CLIP,
  OVERFLOW_PULLDOWN,
  OVERFLOW_DRAG
};

virtual void handle_overflow(int ov);

Fl_Alg tab_align(const Fl_Alg & a);

34.149  Fl_Text_Buffer.H

1 // 2 // Header file for Fl_Text_Buffer class.
3 //
4 // Copyright 2001-2023 by Bill Spitzak and others.
5 // Original code Copyright Mark Edel. Permission to distribute under
6 // the LGPL for the FLTK library granted by Mark Edel.
7 //
8 // This library is free software. Distribution and use rights are outlined in
9 // the file "COPYING" which should have been included with this file. If this
10 // file is missing or damaged, see the license at:
11 //
12 // https://www.fltk.org/COPYING.php
13 //
14 // Please see the following page on how to report bugs and issues:
15 //
16 // https://www.fltk.org/bugs.php
17 //
18 //
19 //
20 Fl_Text_Buffer, Fl_Text_Selection widget . */
34.149 Fl_Text_Buffer.H

21 #ifndef FL_TEXT_BUFFER_H
22 #define FL_TEXT_BUFFER_H
23
24 #include <stdarg.h> /* va_list */
25 #include "fl_attr.h" /* Doxygen can't find <FL/fl_attr.h> */
26
27 #undef ASSERT_UTF8
28
29 #ifdef ASSERT_UTF8
30 # include <assert.h>
31 # define IS_UTF8_ALIGNED(a) if (a && *a) assert(fl_utf8len(*(a))>0);
32 # define IS_UTF8_ALIGNED2(a, b) if (b>=0 && b<a->length()) assert(fl_utf8len(a->byte_at(b))>0);
33 #else
34 # define IS_UTF8_ALIGNED(a)
35 # define IS_UTF8_ALIGNED2(a, b)
36 # endif
37 #endif
38
39 /*
40 * "character size" is the size of a UTF-8 character in bytes
41 * "character width" is the width of a Unicode character in pixels
42 * "column" was orginally defined as a character offset from the left margin.
43 * It was identical to the byte offset. In UTF-8, we have neither a byte offset
44 * nor truly fixed width fonts (a). Column could be a pixel value multiplied with
45 * an average character width (which is a bearable approximation).
46 * in Unicode, there are no fixed width fonts! Even if the ASCII characters may
47 * happen to be all the same width in pixels, Chinese characters surely are not.
48 * There are plenty of exceptions, like ligatures, that make special handling of
49 * "fixed" character widths a nightmare. I decided to remove all references to
50 * fixed fonts and see "columns" as a multiple of the average width of a
51 * character in the main font.
52 * - Matthias
53 */
54
55 /* Maximum length in characters of a tab or control character expansion
56 of a single buffer character */
57 #define FL_TEXT_MAX_EXP_CHAR_LEN 20
58
59 #include "FL_Export.H"
60
61 class Fl_Text_Undo_Action_List;
62 class Fl_Text_Undo_Action;
63
64 class FL_EXPORT Fl_Text_Selection {
65 friend class Fl_Text_Buffer;
66
67 public:
68 // Sets the selection range and selected()
69 void set(int startpos, int endpos);
70
71 // Updates a selection after text was modified.
72 void update(int pos, int nDeleted, int nInserted);
73
74 int start()const { return mSelected ? mStart : 0; }
75
76 int end()const { return mSelected ? mEnd : 0; }
77
78 bool selected()const { return mSelected; }
79
80 int length()const { return mSelected ? mEnd - mStart : 0; }
81
82 int includes(int pos) const;
83
84 int selected(int *startpos, int *endpos) const;
85 FL_DEPRECATED("in 1.4.0 - use selected(startpos, endpos) instead",
86 int position(int *startpos, int *endpos) const) { return selected(startpos, endpos); }
87
88 protected:
89
90 int mStart;
91 int mEnd;
92
93 bool mSelected;
94
95 private:
96
97 Int mStart;
98 Int mEnd;
99 #define void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,
100 int nRestyled, const char * deletedText,
101 void * cbArg);
typedef void (*Fl_Text_Predelete_Cb)(int pos, int nDeleted, void* cbArg);

class FL_EXPORT Fl_Text_Buffer {
    public:

    Fl_Text_Buffer(int requestedSize = 0, int preferredGapSize = 1024);

    ~Fl_Text_Buffer();

    int length() const { return mLength; }

    char* text() const;

    char* text_range(int start, int end) const;

    char* text_at(int pos) const;

    char byte_at(int pos) const;

    bool can_undo() const;

    bool can_redo() const;

    void canUndo(char flag=1);

    int insertfile(const char *file, int pos, int buflen = 128*1024);

    int appendfile(const char *file, int buflen = 128*1024)
    { return insertfile(file, length(), buflen); }

    int loadfile(const char *file, int buflen = 128*1024)
    { select(0, length()); remove_selection(); return appendfile(file, buflen); }

    int outputfile(const char *file, int start, int end, int buflen = 128*1024);

    int savefile(const char *file, int buflen = 128*1024)
    { return outputfile(file, 0, length(), buflen); }

    int tab_distance() const { return mTabDist; }

    void tab_distance(int tabDist);

    void select(int start, int end);

    void unselect();

    int selection_position(int* start, int* end);

    char* selection_text();

    void remove_selection();

    void replace_selection(const char* text);

    void secondary_select(int start, int end);

    int secondary_selected() { return mSecondary.selected(); }

    int selected() const { return mPrimary.selected(); }

    void replace_selection(const char* text);

    void secondary_select(int start, int end);

    int secondary_selected() { return mSecondary.selected(); }

    Generated by Doxygen
void secondary_unselect();
int secondary_selection_position(int * start, int * end);
char * secondary_selection_text();
void remove_secondary_selection();
void replace_secondary_selection(const char * text);
int highlight(int start, int end);
int highlight_position(int * start, int * end);
char * highlight_text();
void add_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void * cbArg);
void remove_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void * cbArg);
void call_modify_callbacks() { call_modify_callbacks(0, 0, 0, 0, 0); }
void add_predelete_callback(Fl_Text_Predelete_Cb bufPredelCB, void * cbArg);
void remove_predelete_callback(Fl_Text_Predelete_CB bufPredelCB, void * cbArg);
void call_predelete_callbacks() { call_predelete_callbacks(0, 0); }
char * line_text(int pos) const;
int line_start(int pos) const;
int line_end(int pos) const;
int word_start(int pos) const;
int word_end(int pos) const;
int count_displayed_characters(int lineStartPos, int targetPos) const;
int skip_displayed_characters(int lineStartPos, int nChars);
int count_lines(int startPos, int endPos) const;
int skip_lines(int startPos, int nLines);
int rewind_lines(int startPos, int nLines);
int findchar_forward(int startPos, unsigned searchChar, int * foundPos) const;
int findchar_backward(int startPos, unsigned int searchChar, int * foundPos) const;
int search_forward(int startPos, const char * searchString, int * foundPos,
int matchCase = 0) const;
int search_backward(int startPos, const char * searchString, int * foundPos,
int matchCase = 0) const;
const Fl_Text_Selection * primary_selection() const { return &mPrimary; }
Fl_Text_Selection * primary_selection() { return &mPrimary; }
const Fl_Text_Selection * secondary_selection() const { return &mSecondary; }
const Fl_Text_Selection * highlight_selection() const { return &mHighlight; }
int prev_char(int ix) const;
int prev_char_clipped(int ix) const;
int next_char(int ix) const;
int next_char_clipped(int ix) const;
int utf8_align(int) const;
int input_file_was_transcoded;
static const char * file_encoding_warning_message;
void (*transcoding_warning_action)(Fl_Text_Buffer*);
bool is_word_separator(int pos) const;
protected:
```c
void call_modify_callbacks(int pos, int nDeleted, int nInserted,
    int nRestyled, const char* deletedText) const;

void call_predelete_callbacks(int pos, int nDeleted) const;

int insert_(int pos, const char* text, int insertedLength = -1);

void remove_(int start, int end);

void redisplay_selection(Fl_Text_Selection* oldSelection,
    Fl_Text_Selection* newSelection) const;

void move_gap(int pos);

void reallocate_with_gap(int newGapStart, int newGapLen);

char* selection_text_(Fl_Text_Selection* sel) const;

void remove_selection_(Fl_Text_Selection* sel);

void replace_selection_(Fl_Text_Selection* sel, const char* text);

void update_selections(int pos, int nDeleted, int nInserted);

int apply_undo(Fl_Text_Undo_Action* action, int* cursorPos);

Fl_Text_Selection mPrimary;
Fl_Text_Selection mSecondary;
Fl_Text_Selection mHighlight;
int mLength;
char* mBuf;
int mGapStart;
int mGapEnd;
// The hardware tab distance used by all displays for this buffer,
// and used in computing offsets for rectangular selection operations.
int mTabDist;
int mNModifyProcs;
Fl_Text_Modify_Cb* mModifyProcs;
void** mModifyCbs;
int mNPredeleteProcs;
Fl_Text_Predelete_Cb* mPredeleteProcs;
void** mPredeleteCbs;
int mCursorPosHint;
char mCanUndo;
int mPreferredPageSize;
Fl_Text_Undo_Action* mUndo;
Fl_Text_Undo_Action_List* mUndoList;
Fl_Text_Undo_Action_List* mRedoList;
```

34.150 Fl_Text_Display.H

1 //
2 // Header file for Fl_Text_Display class.
3 //
4 // Copyright 2001-2023 by Bill Spitzak and others.
5 // Original code Copyright Mark Edel. Permission to distribute under
6 // the LGPL for the FLTK library granted by Mark Edel.
7 //
8 // This library is free software. Distribution and use rights are outlined in
9 // the file "COPYING" which should have been included with this file. If this
10 // file is missing or damaged, see the license at:
11 //
12 //  https://www.fltk.org/COPYING.php
13 //
14 // Please see the following page on how to report bugs and issues:
15 //
16 //  https://www.fltk.org/bugs.php
17 //
18 //
19 /* \file
20 Fl_Text_Display widget . */
21 #ifndef FL_TEXT_DISPLAY_H
22 #define FL_TEXT_DISPLAY_H
23 #define FLTEXT_DISPLAY_H
24 #include <FL/Fl.H> // Fl::scrollbar_size()
25 #include "FL/Fl.H" // Fl::scrollbar_size()
26 #include "Fl_Draw.H"
27 #include "Fl_Group.H"
28 #include "Fl_Widget.H"
29 #include "Fl_Scrollbar.H"
30 #include "Fl_Text_Buffer.H"
31 #endif
```
class FL_EXPORT Fl_Text_Display: public Fl_Group {

enum {
  NORMAL_CURSOR,
  CARET_CURSOR,
  DIM_CURSOR,
  BLOCK_CURSOR,
  HEAVY_CURSOR,
  SIMPLE_CURSOR
};

enum {
  CURSOR_POS,
  CHARACTER_POS
};

enum {
  DRAG_NONE = -2,
  DRAG_START_DND = -1,
  DRAG_CHAR = 0,
  DRAG_WORD = 1,
  DRAG_LINE = 2
};

typedef void (*Unfinished_Style_Cb)(int, void *);

struct Style_Table_Entry {
  Fl_Color color;
  Fl_Font font;
  Fl_Fontsize size;
  unsigned attr;
  Fl_Color bgcolor;
};

enum {
  ATTR_BGCOLOR = 0x0001,
  ATTR_BGCOLOR_EXT_ = 0x0002,
  ATTR_BGCOLOR_EXT = 0x0003,
  ATTR_UNDERLINE = 0x0004,
  ATTR_GRAMMAR = 0x0008,
  ATTR_SPELLING = 0x000C,
  ATTR_STRIKE_THROUGH = 0x0010,
  ATTR_LINES_MASK = 0x001C
};

Fl_Text_Display(int X, int Y, int W, int H, const char *l = 0);

~Fl_Text_Display();

int handle(int e) FL_OVERRIDE;

void buffer(Fl_Text_Buffer* buf);

void buffer(Fl_Text_Buffer& buf) { buffer(&buf); }

Fl_Text_Buffer* buffer()const { return mBuffer; }

void redisplay_range(int start, int end);

void scroll(int topLineNum, int horizOffset);

void insert(const char * text);

void overstrike(const char * text);

void insert_position(int newPos);

int insert_position()const { return mCursorPos; }

int in_selection(int x, int y) const;

void show_insert_position();

int move_right();

int move_left();

int move_up();

int move_down();

int count_lines(int start, int end, bool start_pos_is_line_start) const;

int line_start(int pos) const;
int line_end(int startPos, bool startPosIsLineStart) const;
int skip_lines(int startPos, int nLines, bool startPosIsLineStart);
void next_word(void);
void previous_word(void);

void show_cursor(int b = 1);

void hide_cursor() { show_cursor(0); }

int cursor_style() const { return mCursorStyle; }

int scrollbar_width() const {
    return scrollbar_width_ ? scrollbar_width_ : Fl::scrollbar_size();
}

void scrollbar_width(int width) {
    Fl::scrollbar_size(width);
    scrollbar_width_ = 0;
}

int position_style(int lineStartPos, int lineLen, int lineIndex) const;

int wrapped_column(int row, int column) const;

virtual void recalc_display();
void resize(int X, int Y, int W, int H) FL_OVERRIDE;
double col_to_x(double col) const;
void linenumber_width(int width);
int linenumber_width() const;
void linenumber_font(Fl_Font val);
Fl_Font linenumber_font() const;
void linenumber_size(Fl_Fontsize val);
Fl_Fontsize linenumber_size() const;
void linenumber_fgcolor(Fl_Color val);
Fl_Color linenumber_fgcolor() const;
void linenumberbgcolor(Fl_Color val);
Fl_Color linenumberbgcolor() const;
void linenumber_align(Fl_Align val);
Fl_Align linenumber_align() const;
void linenumber_format(const char * val);
const char * linenumber_format() const;

protected:
// Most (all?) of this stuff should only be called from resize() or
draw().
// Anything with "vline" indicates thats it deals with currently
visible lines.

void draw() FL_OVERRID;
void draw_text(int X, int Y, int W, int H);
void draw_range(int start, int end);
void draw_cursor(int, int);
void draw_string(int style, int x, int y, int toX, const char *string,
int nChars) const;
void draw_vline(int visLineNum, int leftClip, int rightClip,
int leftCharIndex, int rightCharIndex);
int find_x(const char *s, int len, int style, int x) const;
enum {
DRAW_LINE,
FIND_INDEX,
FIND_INDEX_FROM_ZERO,
GET_WIDTH,
FIND_CURSOR_INDEX // STR $2788
};
int handle_vline(int mode,
int lineStart, int lineLen, int leftChar, int rightChar,
int topClip, int bottomClip,
int leftClip, int rightClip) const;

void draw_line_numbers(bool clearAll);
void clear_rect(int style, int x, int y, int width, int height) const;
void display_insert();
void offset_line_starts(int newTopLineNum);
void calc_line_starts(int startLine, int endLine);
void update_line_starts(int Pos, int nInserted, int nDeleted,
int linesInserted, int linesDeleted, int *scrolled);
void calc_last_char();
int position_to_line(int pos, int * lineNum) const;
double string_width(const char * string, int length, int style) const;
static void scroll_timer_cb(void *);
static void buffer_predelete_cb(int pos, int nDeleted, void * cbArg);
static void buffer_modified_cb(int pos, int nInserted, int nDeleted,
int nRestyled, const char * deletedText,
void * cbArg);
static void h_scrollbar_cb(Fl_Scrollbar * w, Fl_Text_Display* d);
static void v_scrollbar_cb( Fl_Scrollbar * w, Fl_Text_Display* d);
void update_v_scrollbar();
void update_h_scrollbar();
int measure_vline(int visLineNum) const;
int longest_vline() const;
int empty_vlines() const;
int vline_length(int visLineNum) const;
int xy_to_position(int x, int y, int PosType = CHARACTER_POS) const;
void xy_to_rowcol(int x, int y, int PosType = CHARACTER_POS) const;
void maintain_absolute_top_line_number(int state);
```cpp
538 int get_absolute_top_line_number() const;
539 void absolute_top_line_number(int oldFirstChar);
540 int maintaining_absolute_top_line_number() const;
541 void reset_absolute_top_line_number();
542 int position_to_linecol(int pos, int* lineNum, int* column) const;
543
544 void extend_range_for_styles(int* start, int* end);
545
546 void find_wrap_range(const char* deletedText, int pos, int nInserted,
547 int nDeleted, int* modRangeStart, int* modRangeEnd,
548 int* linesInserted, int* linesDeleted);
549 void measure_deleted_lines(int pos, int nDeleted);
550 int scroll_(int topLineNum, int horizOffset);
551
552 int damage_range1_start, damage_range1_end;
553 int damage_range2_start, damage_range2_end;
554 int mCursorPos;
555 int mCursorOn;
556 int mCursorOldY; /* Y pos. of cursor for blanking */
557 int mCursorToHint; /* Tells the buffer modified callback
558 where to move the cursor, to reduce
559 the number of redraw calls */
560 int mCursorStyle; /* One of enum cursorStyles above */
561 int mCursorPreferredXPos; /* Pixel position for vert. cursor movement */
562 int mNVisibleLines; /* # of visible (displayed) lines. This is
563 also the size of the mLineStarts[] array. */
564 int mNBufferLines; /* # of newlines in the buffer */
565 int mTopLineNum; /* Line number of top displayed line
566 of file (first line of file is 1) */
567 int mAbsTopLineNum; /* In continuous wrap mode, the line
568 number of the top line if the text
569 were not wrapped (note that this is
570 only maintained as needed). */
571 int mNeedAbsTopLineNum; /* Externally settable flag to continue
572 maintaining absTopLineNum even if
573 it isn't needed for line # display */
574 int mHorizOffset; /* Horizontal scroll pos. in pixels */
575 int mTopLineNumHint; /* Line number of top displayed line
576 of file [first line of file is 1] */
577 int mHorizOffsetHint; /* Horizontal scroll pos. in pixels */
578 int mNStyles; /* Number of entries in styleTable */
579 int mNUnstyles; /* Table of fonts and colors for
580 coloring/syntax-highlighting */
581 int mUnfinishedStyle; /* Style buffer entry which triggers
582 on-the-fly reparsing of region */
583 Unfinished_Style_CB mUnfinishedHighlightCB; /* Callback to parse "unfinished" */
584 Fl_Text_Buffer* mBuffer; /* Contains text to be displayed */
585 Fl_Text_Buffer* mStyleBuffer; /* Optional parallel buffer containing
586 color and font information */
587 int mFirstChar, mLastChar; /* Buffer positions of first and last
588 displayed character (lastChar points
589 to a newline or one character
590 beyond the end of the buffer) */
591 int mContinuousWrap; /* Wrap long lines when displaying */
592 int mWrapMarginPix; /* Margin in # of pixels for
593 wrapping in continuousWrap mode */
594 int mLineStarts; /* Array of the size mNVisibleLines.
595 This array only keeps track of lines
596 within the display area. Each entry
597 contains the starting character offset
598 from the beginning of the text buffer
599 for each /visible/ line.
600 If wrap enabled, points to the beginning
601 of each wrap. So a long line wrapping
602 into 3 separate lines in the display
603 will take up 3 separate array entries. */
604 int mTopLineNum; /* Line number of top displayed line
605 of file [first line of file is 1] */
606 int mAbsTopLineNum; /* In continuous wrap mode, the line
607 number of the top line if the text
608 were not wrapped (note that this is
609 only maintained as needed). */
610 int mNNeedAbsTopLineNum; /*Externally settable flag to continue
611 maintaining absTopLineNum even if
612 it isn't needed for line # display */
613 int mSuppressResync; /* Suppress resynchronization of line
614 starts during buffer updates */
615 int mNLinesDeleted; /* Number of lines deleted during
616 buffer modification (only used
617 when resynchronization is suppressed) */
618 int mModifyingTabDistance; /* Whether tab distance is being modified XXX: UNUSED */
```
mutable double mColumnScale; /* Width in pixels of an average character. This
value is calculated as needed (lazy eval); it
needs to be mutable so that it can be calculated
within a method marked as "const" */

Fl_Color mCursor_color;

Fl_Scrollbar * mHScrollBar;
Fl_Scrollbar * mVScrollBar;
int scrollbar_width_; // size of scrollbar trough (behavior changed in 1.4)
Fl_Align scrollbar_align;
dragPos, dragType, dragging;
display_insert_position_hint;
struct { int x, y, w, h; } text_area;

int shortcut_;

Fl_Font textfont, linenumber_font;
Fl_Fontsize textsize, linenumber_size;
Fl_Color textcolor, linenumber_fgcolor, linenumber_bgcolor;
Fl_Align linenumber_align, linenumber_format;

// Line number margin and width
int mLineNumLeft, mLineNumWidth;

// Line number font/colors
Fl_Font linenumber_font_
Fl_Fontsize linenumber_size_
Fl_Color linenumber_fgcolor_
Fl_Color linenumberbgcolor_
Fl_Align linenumber_align_
const char * linenumber_format_
;

/*
file
Fl_Text_Editor widget . */

Fl_Text_Editor(int X, int Y, int W, int H, const char * l = 0);
Fl_Text_Editor::~Fl_Text_Editor() { remove_all_key_bindings(); }
int handle(int e) FL_OVERRIDE;
void insert_mode(int b) { insert_mode_ = b; }

1357 Fl_Text_Editor.H
int insert_mode() { return insert_mode_; }
void tab_nav(int val);
int tab_nav() const;
void add_key_binding(int key, int state, Key_Func f, Key_Binding** list);
void add_key_binding(int key, int state, Key_Func f)
{ add_key_binding(key, state, f, &key_bindings); }
void remove_key_binding(int key, int state, Key_Binding** list);
void remove_key_binding(int key, int state)
{ remove_key_binding(key, state, &key_bindings); }
void remove_all_key_bindings(Key_Binding** list);
void remove_all_key_bindings() { remove_all_key_bindings(&key_bindings); }
void add_default_keyBindings(Key_Binding** list);
void add_default_keyBindings(Key_Binding* list) const;
void add_default_keyBindings(int key, int state) const
{ add_default_key_bindings(Key_Binding* list); }
void default_key_function(Key_Func f) { default_key_function_ = f; }

// functions for the built in default bindings
static int kf_default(int c, Fl_Text_Editor* e);
static int kf_ignore(int c, Fl_Text_Editor* e);
static int kf_backspace(int c, Fl_Text_Editor* e);
static int kf_enter(int c, Fl_Text_Editor* e);
static int kf_move(int c, Fl_Text_Editor* e);
static int kf_shift_move(int c, Fl_Text_Editor* e);
static int kf_e_s_move(int c, Fl_Text_Editor* e);
static int kf_meta_move(int c, Fl_Text_Editor* e);
static int kf_meta_move(int c, Fl_Text_Editor* e);
static int kf_home(int, Fl_Text_Editor* e);
static int kf_end(int c, Fl_Text_Editor* e);
static int kf_left(int c, Fl_Text_Editor* e);
static int kf_up(int c, Fl_Text_Editor* e);
static int kf_right(int c, Fl_Text_Editor* e);
static int kf_down(int c, Fl_Text_Editor* e);
static int kf_page_up(int c, Fl_Text_Editor* e);
static int kf_page_down(int c, Fl_Text_Editor* e);
static int kf_insert(int c, Fl_Text_Editor* e);
static int kf_delete(int c, Fl_Text_Editor* e);
static int kf_copy(int c, Fl_Text_Editor* e);
static int kf_cut(int c, Fl_Text_Editor* e);
static int kf_paste(int c, Fl_Text_Editor* e);
static int kf_select_all(int c, Fl_Text_Editor* e);
static int kf_undo(int c, Fl_Text_Editor* e);
static int kf_redo(int c, Fl_Text_Editor* e);

protected:
int handle_key();
void maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_CHANGED);

#ifndef FL_DOXYGEN
int insert_mode_; // Key_Binding* key_bindings;
#endif

#ifndef FL_DOXYGEN
Key_Func default_key_function_; // new
#endif

３４．１５２ ＦＩＴＥＬＩＥ．Ｈ

// Tile header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2016 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
#ifndef FL_TILE_H
#define FL_TILE_H

 Generated by Doxygen
20 #include "Fl_Group.H"
21
22 /*
23 The Fl_Tile class lets you resize its children by dragging
24 the border between them.
25 */
26
27 class FL_EXPORT Fl_Tile : public Fl_Group {
28 public:
29   int handle(int event) FL_OVERRIDE;
30   Fl_Tile(int X, int Y, int W, int H, const char *L = 0);
31   void resize(int X, int Y, int W, int H) FL_OVERRIDE;
32   void move_intersection(int oldx, int oldy, int newx, int newy);
33   FL_DEPRECATED("in 1.4.0 - use move_intersection(p) instead",
34                 void position(int oldx, int oldy, int newx, int newy)) { return move_intersection(oldx, oldy, newx, newy); }
35   void position(int x, int y) { Fl_Group::position(x, y); }
36 };
37
38 #endif

34.153 Fl_Tiled_Image.H

// Tiled image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2015 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// https://www.fltk.org/bugs.php
//
// File
// Fl_Tiled_Image widget.

#ifndef Fl_Tiled_Image_H
#define Fl_Tiled_Image_H

#include "Fl_Image.H"

class FL_EXPORT Fl_Tiled_Image : public Fl_Image {
protected:
  Fl_Image *image_; // The image that is tiled
  int alloc_image_; // Did we allocate this image?
public:
  Fl_Tiled_Image(Fl_Image *i, int W = 0, int H = 0);
  virtual ~Fl_Tiled_Image();
  Fl_Image *copy(int W, int H) const FL_OVERRIDE;
  Fl_Image *copy() const {
    return Fl_Image::copy();
  }
  void color_average(Fl_Color c, float i) FL_OVERRIDE;
  void desaturate() FL_OVERRIDE;
  void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) FL_OVERRIDE;
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  Fl_Image *image() { return image_; }
};
#endif // !Fl_Tiled_Image_H

34.154 Fl_Timer.H

/*
 * Timer header file for the Fast Light Tool Kit (FLTK).
 */

// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// Generated by Doxygen
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 /*! Fl_Timer widget. */
18 #ifndef Fl_Timer_H
19 #define Fl_Timer_H
20 #endif
21 #ifndef Fl_Widget_H
22 #include "Fl_Widget.H"
23 #endif
24
25 // values for type():
26 #define FL_NORMAL_TIMER 0
27 #define FL_VALUE_TIMER 1
28 #define FL_HIDDEN_TIMER 2
29
30 class FL_EXPORT Fl_Timer : public Fl_Widget {
31 static void stepcb(void *);
32 void step();
33 char on, direction_;
34 double delay, total;
35 long lastsec,lastusec;
36 protected:
37 void draw() FL_OVERRIDE;
38 public:
39 int handle(int) FL_OVERRIDE;
40 Fl_Timer(uchar t,int x,int y,int w,int h, const char *l);
41 ~Fl_Timer();
42 void value(double);
43 double value()const {return delay>0.0?delay:0.0;}
44 char direction()const {return direction_;
45 void direction(char d) {direction_ = d;}
46 char suspended()const {return !on;}
47 void suspended(char d);
48 }
49 #endif

34.155 Fl_Toggle_Button.H

1 //
2 // Toggle button header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //
17 /*! Fl_Toggle_Button widget. */
18 #ifndef Fl_Toggle_Button_H
19 #define Fl_Toggle_Button_H
20 #define Fl_Toggle/Button_H
21 #include "Fl_Button.H"
22 #include "Fl_Widget.H"
23 #include "Fl_Box.H"
24
25 class FL_EXPORT Fl_Toggle_Button : public Fl_Button {
26 public:
27 Fl_Toggle_Button(int X,int Y,int W,int H,const char *=0);
28 }
29 #endif

34.156 Fl_Toggle_Light_Button.H

1 //
2 // Toggle light button header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
16 //
17 /*! Fl_Toggle_Light_Button widget. */
18 #ifndef Fl_Toggle_Light_Button_H
19 #define Fl_Toggle_Light_Button_H
20 #define Fl_Toggle_Light/Button_H
21 #include "Fl_Button.H"
22 #include "Fl_Widget.H"
23 #include "Fl_Box.H"
24 #include "Fl_Toggle_Button.H"
25
26 class FL_EXPORT Fl_Toggle_Light_Button : public Fl_Button {
27 public:
28 Fl_Toggle_Light_Button(int X,int Y,int W,int H,const char *=0);
29 }
30 #endif

Generated by Doxygen
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 // provided for back-compatibility only
17
18 ifndef Fl_Toggle_Light_Button
19 #include "Fl_Light_Button.H"
20 #define Fl_Toggle_Light_Button Fl_Light_Button
21 #endif
22

34.157 Fl_Toggle_Round_Button.H

1 //
2 // Toggle round button header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 // provided for back-compatibility only
17
18 ifndef Fl_Toggle_Round_Button
19 #include "Fl_Round_Button.H"
20 #define Fl_Toggle_Round_Button Fl_Round_Button
21 #endif
22

34.158 Fl_Tooltip.H

1 //
2 // Tooltip header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2011 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 //\file
18 Fl_Tooltip widget . */
19
20 ifndef Fl_Tooltip_H
21 #define Fl_Tooltip_H
22 #endif
23
24 #include <FL/Fl.H>
25 #include <FL/Fl_Widget.H>
26
27 class FL_EXPORT Fl_Tooltip {
28 friend class Fl_TooltipBox;
29 public:
30 static float delay() { return delay_; }
31 static void delay(float f) { delay_ = f; }
32 static float hidedelay() { return hidedelay_; }
33 static void hidedelay(float f) { hidedelay_ = f; }
34 static float hoverdelay() { return hoverdelay_; }
35 static void hoverdelay(float f) { hoverdelay_ = f; }
36
Generated by Doxygen
static int enabled() { return Fl::option(Fl::OPTION_SHOW_TOOLTIPS); }
static void enable(int b = 1) { Fl::option(Fl::OPTION_SHOW_TOOLTIPS, (b!=0)); }
static void disable() { enable(0); }
static void (*enter)(Fl_Widget* w);
static void enter_area(Fl_Widget * w, int X, int Y, int W, int H, const char* tip);
static void (*exit)(Fl_Widget *w);
static Fl_Widget * current() {return widget_;}
static void current(Fl_Widget *);

static Fl_Font font() { return font_; }
static void font(Fl_Font i) { font_ = i; }
static Fl_Fontsize size() { return (size_ == -1 ? FL_NORMAL_SIZE : size_); }
static void size(Fl_Fontsize s) { size_ = s; }
static Fl_Color color() { return color_; }
static void color(Fl_Color c) { color_ = c; }
static Fl_Color textcolor() { return textcolor_; }
static void textcolor(Fl_Color c) { textcolor_ = c; }
static int margin_width() { return margin_width_; }
static void margin_width(int v) { margin_width_ = v; }
static int margin_height() { return margin_height_; }
static void margin_height(int v) { margin_height_ = v; }
static float delay_; 
static float hidedelay_; 
static float hoverdelay_; 
static Fl_Color color_; 
static Fl_Color textcolor_; 
static Fl_Font font_; 
static Fl_Fontsize size_; 
static Fl_Widget* widget_; 
static int margin_width_; 
static int margin_height_; 
static int wrap_width_; 
static int wrap_width(int v) { wrap_width_ = v; }
static float wrap_width(); 
static Fl_Window* current_window(void);
static Fl_Window* current_window();
static const int draw_symbols_; // 1 = draw @-symbols in tooltips, 0 = no

private:
friend void Fl_Widget::tooltip(const char *);
friend void Fl_Widget::copy_tooltip(const char *);
static void enter_(Fl_Widget * w);
static void exit_(Fl_Widget *w);
static void set_enter_exit_once_();

private:
static Fl_Font font_;
static Fl_Fontsize size_;
static Fl_Color color_;
static Fl_Color textcolor_;
static Fl_Font font_; 
static Fl_Fontsize size_; 
static Fl_Widget * widget_; 
static int margin_width_; 
static int margin_height_; 
static int wrap_width_; 
static int wrap_width(int v) { wrap_width_ = v; }
static Fl_Color textcolor_; 
static Fl_Font font_; 
static Fl_Fontsize size_; 
static Fl_Widget* widget_; 
static int margin_width_; 
static int margin_height_; 
static const int draw_symbols_; // 1 = draw @-symbols in tooltips, 0 = no

34.159 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 = FL_REASON_UNKNOWN , FL_TREE_REASON_SELECTED = FL_REASON_SELECTED ,
  FL_TREE_REASON_DESELECTED = FL_REASON_DESELECTED , FL_TREE_REASON_RESELECTED ,
  FL_REASON_RESELECTED ,
34.159.1 Detailed Description

This file contains the definitions of the Fl_Tree class.

34.159.2 Enumeration Type Documentation

34.159.2.1 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>

34.160 Fl_Tree.H

Go to the documentation of this file.

//
// fl_Tree.h
//
// This file is part of the Fl_Tree widget for FLTK
// Copyright (C) 2009-2010 by Greg Ercolano.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// http://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
// http://www.fltk.org/bugs.php
//
//
//
// Fl_Tree
enum Fl_Tree_Reason {
    FL_TREE_REASON_NONE = FL_REASON_UNKNOWN,
    FL_TREE_REASON_SELECTED = FL_REASON_SELECTED,
    FL_TREE_REASON_DESELECTED = FL_REASON_DESELECTED,
    FL_TREE_REASON_RESELECTED = FL_REASON_RESELECTED,
    FL_TREE_REASON_OPENED = FL_REASON_OPENED,
    FL_TREE_REASON_CLOSED = FL_REASON_CLOSED,
};
FL_TREE_REASON_DRAGGED = FL_REASON_DRAGGED

class FL_EXPORT Fl_Tree : public Fl_Group {
friend class Fl_Tree_Item;
Fl_Tree_Item *_root; // can be null!
Fl_Tree_Item *_item_focus; // item that has focus box
Fl_Tree_Item *_callback_item; // item invoked during callback (can be NULL)
Fl_Tree_Reason _callback_reason; // reason for the callback
Fl_Tree_Prefs _prefs; // all the tree's settings
int _scrollbar_size; // size of scrollbar trough
Fl_Tree_Item *_lastselect; // last selected item
char _lastpushed; // FL_PUSH occurred on: 0=nothing, 1=open/close,
2=usericon, 3=label
void fix_scrollbar_order();
protected:
Fl_Scrollbar *_vscroll;
Fl_Scrollbar *_hscroll;
int _tox,_toy,_tow,_toh;
int _tix,_tiy,_tiw,_tih;

int _tree_w;
int _tree_h;
void item_clicked(Fl_Tree_Item * val);
void do_callback_for_item(Fl_Tree_Item * item, Fl_Tree_Reason reason);

// next_visible_item() and extend_selection() moved to `public` in ABI 1.3.3
// undocumented draw_tree() dropped -- draw() does all the work now
// draw() has to be protected per FLTK convention (was public in 1.3.x)
void draw() FL_OVERRIDE;

public:
Fl_Tree(int X, int Y, int W, int H, const char *L=0);
~Fl_Tree();
int handle(int e) FL_OVERRIDE;
void show_self();
void resize(int,int,int,int) FL_OVERRIDE;

// root methods
void root_label(const char *new_label);
Fl_Tree_Item * root();
void root(Fl_Tree_Item *newitem);
const Fl_Tree_Prefs& prefs()const { return _prefs; }

// Item creation/removal methods
Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem=0);
Fl_Tree_Item * add(Fl_Tree_Item *parent_item, const char *name);
Fl_Tree_Item *insert_above(Fl_Tree_Item *above, const char *name);
Fl_Tree_Item * insert(Fl_Tree_Item *item, const char *name, int pos);
int remove(Fl_Tree_Item *item);
void clear();
void clear_children(Fl_Tree_Item *item);

// Item lookup methods
Fl_Tree_Item *find_item(const char *path);
const Fl_Tree_Item *find_item(const char *path) const;
int item_pathname(char *pathname, int pathnamelen, const Fl_Tree_Item *item) const;
Fl_Tree_Item * find_clicked(int yonly=0) const;
Fl_Tree_Item * find_clicked(int yonly=0);
Fl_Tree_Item *item_clicked();
Fl_Tree_Item *first();
Fl_Tree_Item *first_visible(); // deprecated in ABI 10303
Fl_Tree_Item *first_visible_item();
Fl_Tree_Item *next(Fl_Tree_Item *item=0);
Fl_Tree_Item *prev(Fl_Tree_Item *item=0);
Fl_Tree_Item *last();
Fl_Tree_Item *last_visible(); // deprecated in ABI 10303
Fl_Tree_Item *last_visible_item();
Fl_Tree_Item *next_visible_item(Fl_Tree_Item *item=0, int dir=FL_Down, bool visible=false);
Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item=0, int dir=FL_Down);
int get_selected_items(Fl_Tree_Item_Array &items);

// Item open/close methods
int open(Fl_Tree_Item *item, int docallback=1);
int open(const char *path, int docallback=1);
void open_toggle(Fl_Tree_Item *item, int docallback=1);
int close(Fl_Tree_Item *item, int docallback=1);
int close(const char *path, int docallback=1);
int is_open(Fl_Tree_Item *item) const;
int is_open(const char *path) const;
int is_close(Fl_Tree_Item *item) const;
int is_close(const char *path) const;
34.160 Fl_Tree.H

```
379 // Item selection methods
380 int select(Fl_Tree_Item *item, int docallback=1);
381 int select(const char *path, int docallback=1);
382 void select_toggle(Fl_Tree_Item *item, int docallback=1);
383 int deselect(Fl_Tree_Item *item, int docallback=1);
384 int deselect(const char *path, int docallback=1);
385 void select_toggle(Fl_Tree_Item *item, int docallback=1);
386 int deselect(Fl_Tree_Item *item, int docallback=1);
387 int select_all(Fl_Tree_Item *item=0, int docallback=1);
388 int extend_selection_dir(Fl_Tree_Item *from,
389 Fl_Tree_Item *to,
390 int dir,
391 int val,
392 bool visible);
393 int extend_selection(Fl_Tree_Item *from,
394 Fl_Tree_Item *to,
395 int val=1,
396 bool visible=false);
397 void set_item_focus(Fl_Tree_Item *item);
398 Fl_Tree_Item *get_item_focus() const;
399 int is_selected(Fl_Tree_Item *item) const;
400 int is_selected(const char *path);
401 // Item attribute related methods
402 Fl_Font item_labelfont() const;
403 void item_labelfont(Fl_Font val);
404 Fl_Fontitem_labelsize() const;
405 void item_labelsize(Fl_Fontsize val);
406 Fl_Color item_labelfgcolor(void) const;
407 void item_labelfgcolor(Fl_Color val);
408 Fl_Color item_labelbgcolor(void) const;
409 void item_labelbgcolor(Fl_Color val);
410 Fl_Color connectorcolor() const;
411 void connectorcolor(Fl_Color val);
412 int marginleft() const;
413 void marginleft(int val);
414 int margintop() const;
415 void margintop(int val);
416 int marginbottom() const;
417 void marginbottom(int val);
418 int linespacing() const;
419 void linespacing(int val);
420 int openchild_marginbottom() const;
421 void openchild_marginbottom(int val);
422 int usericonmarginleft() const;
423 void usericonmarginleft(int val);
424 int labellmarginleft() const;
425 void labellmarginleft(int val);
426 int widgetmarginleft() const;
427 void widgetmarginleft(int val);
428 int connectorwidth() const;
429 void connectorwidth(int val);
430 Fl_Image * usericon() const;
431 void usericon(Fl_Image *val);
432 Fl_Image * openicon() const;
433 void openicon(Fl_Image *val);
434 Fl_Image * closeicon() const;
435 void closeicon(Fl_Image *val);
436 int showcollapse() const;
437 void showcollapse(int val);
438 int showroot() const;
439 void showroot(int val);
440 Fl_Tree_Connector connectorstyle() const;
441 void connectorstyle(Fl_Tree_Connector val);
442 Fl_Tree_Sort sortorder() const;
443 void sortorder(Fl_Tree_Sort val);
444 Fl_Boxtype selectbox() const;
445 void selectbox(Fl_Boxtype val);
446 Fl_Tree_Item_Reselect_Mode item_reselect_mode() const;
447 void item_reselect_mode(Fl_Tree_Item_Reselect_Mode mode);
448 void item_reselect_mode(int mode);
449 void calc_dimensions();
450 void calc_tree();
451 void recalc_tree();
452 int displayed(Fl_Tree_Item *item);
453 void show_item(Fl_Tree_Item *item, int int yoff);
454 void show_item(Fl_Tree_Item *item);
455 void show_item_top(Fl_Tree_Item *item);
456 void show_item_middle(Fl_Tree_Item *item);
457 void show_item_bottom(Fl_Tree_Item *item);
458 void display(Fl_Tree_Item *item);
459 int vposition() const;
```
470    void vposition(int pos);
471    int  hposition() const;
472    void hposition(int pos);
473
474    int is_scrollbar(Fl_Widget *w);
475    int scrollbar_size() const;
476    void scrollbar_size(int size);
477    int is_vscroll_visible() const;
478    int is_hscroll_visible() const;
479
481   // callback related
483    void callback_item(Fl_Tree_Item *item);
484    Fl_Tree_Item *callback_item();
485    void callback_reason(Fl_Tree_Reason reason);
486    Fl_Tree_Reason callback_reason() const;
487
489    void load(class Fl_Preferences&);
490    }
491
492   #endif /*FL_TREE_H*/

34.161 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
  Tree widget item.

34.161.1 Detailed Description

This file contains the definitions for Fl_Tree_Item.

34.162 Fl_Tree_Item.H

Go to the documentation of this file.
1 //
2
3 #ifndef FL_TREE_ITEM_H
4 #define FL_TREE_ITEM_H
5
6 #include <FL/Fl.H>
7 #include <FL/Fl_Widget.H>
8 #include <FL/Fl_Image.H>
9 #include <FL/fl_draw.H>
10
11 #include <FL/Fl_Tree_Item_Array.H>
12 #include <FL/Fl_Tree_Prefs.H>
13
15 // FL/Fl_Tree_Item.H
17 // Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
19 // Copyright (C) 2009-2010 by Greg Ercolano.
20 //
21 // This library is free software. Distribution and use rights are outlined in
22 // the file *COPYING* which should have been included with this file. If this
23 // file is missing or damaged, see the license at:
24 //
25 //  https://www.fltk.org/COPYING.php
26 //
27 // Please see the following page on how to report bugs and issues:
28 //
29 //  https://www.fltk.org/bugs.php
30 //
34.162 Fl_Tree_Item.H

31
36
64 class Fl_Tree;
65 class FL_EXPORT Fl_Tree_Item {
66
Fl_Tree
// parent tree
*_tree;
67
const char
// label (memory managed)
*_label;
68
Fl_Font
_labelfont;
// label’s font face
69
Fl_Fontsize
_labelsize;
// label’s font size
70
Fl_Color
_labelfgcolor;
// label’s fg color
71
Fl_Color
_labelbgcolor;
// label’s bg color (0xffffffff is ’transparent’)
73
enum Fl_Tree_Item_Flags {
74
OPEN
= 1«0,
75
VISIBLE
= 1«1,
76
ACTIVE
= 1«2,
77
SELECTED
= 1«3
78
};
79
unsigned short _flags;
// misc flags
80
int
_xywh[4];
// xywh of this widget (if visible)
81
int
_collapse_xywh[4];
// xywh of collapse icon (if visible)
82
int
_label_xywh[4];
// xywh of label
83
Fl_Widget
// item’s label widget (optional)
*_widget;
84
Fl_Image
// item’s user-specific icon (optional)
*_usericon;
85
Fl_Image
// deactivated usericon
*_userdeicon;
86
Fl_Tree_Item_Array
_children;
// array of child items
87
Fl_Tree_Item
// parent item (=0 if root)
*_parent;
88
void
// user data that can be associated with an item
*_userdata;
89
Fl_Tree_Item
// previous sibling (same level)
*_prev_sibling;
90
Fl_Tree_Item
// next sibling (same level)
*_next_sibling;
91
// Protected methods
92 protected:
93
void _Init(const Fl_Tree_Prefs &prefs, Fl_Tree *tree);
94
void show_widgets();
95
void hide_widgets();
96
void draw_vertical_connector(int x, int y1, int y2, const Fl_Tree_Prefs &prefs);
97
void draw_horizontal_connector(int x1, int x2, int y, const Fl_Tree_Prefs &prefs);
98
void recalc_tree();
99
int calc_item_height(const Fl_Tree_Prefs &prefs) const;
100
Fl_Color drawfgcolor() const;
101
Fl_Color drawbgcolor() const;
102
103 public:
104
Fl_Tree_Item(const Fl_Tree_Prefs &prefs);
// CTOR -- backwards compatible
105
Fl_Tree_Item(Fl_Tree *tree);
// CTOR -- ABI 1.3.3+
106
virtual ~Fl_Tree_Item();
// DTOR -- ABI 1.3.3+
107
Fl_Tree_Item(const Fl_Tree_Item *o);
// COPY CTOR
109
int x()const { return(_xywh[0]); }
111
int y()const { return(_xywh[1]); }
114
int w()const { return(_xywh[2]); }
116
int h()const { return(_xywh[3]); }
119
int label_x()const { return(_label_xywh[0]); }
122
int label_y()const { return(_label_xywh[1]); }
126
int label_w()const { return(_label_xywh[2]); }
129
int label_h()const { return(_label_xywh[3]); }
130
virtual int draw_item_content(int render);
131
void draw(int X, int &Y, int W, Fl_Tree_Item *itemfocus,
132
int &tree_item_xmax, int lastchild=1, int render=1);
133
void show_self(const char *indent = "") const;
134
void label(const char *val);
135
const char *label() const;
136
138
inline void user_data( void* data ) { _userdata = data; }
139
141
inline void* user_data()const { return _userdata; }
142
144
void labelfont(Fl_Font val) {
145
_labelfont = val;
146
recalc_tree();
// may change tree geometry
147
}
149
Fl_Font labelfont()const {
150
return(_labelfont);
151
}
153
void labelsize(Fl_Fontsize val) {
154
_labelsize = val;
155
recalc_tree();
// may change tree geometry
156
}
158
Fl_Fontsize labelsize()const {
159
return(_labelsize);
160
}
162
void labelfgcolor(Fl_Color val) {
163
_labelfgcolor = val;
164
}
166
Fl_Color labelfgcolor()const {
167
return(_labelfgcolor);
168
}
170
void labelcolor(Fl_Color val) {
171
labelfgcolor(val);
172
}

Generated by Doxygen

1367


void labelbgcolor(Fl_Color val) {
    _labelbgcolor = val;
}
Fl_Color labelbgcolor() const {
    return(_labelbgcolor);
}

Fl_Widget *widget() const {
    return(_widget);
}

int children() const {
    return(_children.total());
}
Fl_Tree_Item *child(int index) {
    return(_children[index]);
}
const Fl_Tree_Item *child(int t) const; 
int has_children() const {
    return(children());
}

int find_child(const char *name); 
int find_child(Fl_Tree_Item *item); 
int remove_child(Fl_Tree_Item *item); 
int remove_child(const char *new_label); 
void clear_children(); 
void swap_children(int ax, int bx); 

Fl_Tree_Item *add(const Fl_Tree_PREFS &prefs, const char *new_label, Fl_Tree_Item *newitem); 
Fl_Tree_Item *add(const Fl_Tree_PREFS &prefs, const char *new_label); 
Fl_Tree_Item *add(const Fl_Tree_PREFS &prefs, char **arr, Fl_Tree_Item *newitem); 
Fl_Tree_Item *add(const Fl_Tree_PREFS &prefs, char **arr); 
Fl_Tree_Item *replace(Fl_Tree_Item *new_item); 
Fl_Tree_Item *replace_child(Fl_Tree_Item *olditem, Fl_Tree_Item *newitem); 
Fl_Tree_Item *insert(const Fl_Tree_PREFS &prefs, const char *new_label, int pos=0); 
Fl_Tree_Item *insert_above(const Fl_Tree_PREFS &prefs, const char *new_label); 
Fl_Tree_Item *deparent(int index); 
int reparent(Fl_Tree_Item *newchild, int index); 
int move(int to, int from); 
int move(Fl_Tree_Item *item, int op=0, int pos=0); 
int move_above(Fl_Tree_Item *item); 
int move_below(Fl_Tree_Item *item); 
int move_into(Fl_Tree_Item *item, int pos=0); 

Fl_Tree_Item *parent() {
    return(_parent);
}
const Fl_Tree_Item *parent() const {
    return(_parent);
}

void parent(Fl_Tree_Item *val) {
    _parent = val;
}
const Fl_Tree_PREFS prefs() const;
const Fl_Tree *tree() const {
    return(_tree);
}

Fl_Tree *tree() { 
    return(_tree);
}
284 } 
286 // State
288 void open();
289 void close();
291 int is_open() const {
292 return is_flag(OPEN);
293 } 
295 int is_close() const {
296 return is_flag(OPEN)?0:1;
297 } 
299 void open_toggle() {
300 is_open()?close():open(); // handles calling recalc_tree()
301 } 
305 void select(int val=1) {
306 set_flag(SELECTED, val);
307 } 
309 void select_toggle() {
310 if ( is_selected() ) 
311 deselect(); // deselect if selected
312 } else {
313 select(); // select if deselected
314 
315 } 
320 int select_all() {
321 int count = 0;
322 if ( ! is_selected() ) {
323 select();
324 ++count;
325 }
326 for ( int t=0; t<children(); t++ ) {
327 count += child(t)->select_all();
328 }
329 return(count);
330 } 
332 void deselect() {
333 set_flag(SELECTED, 0);
334 } 
339 int deselect_all() {
340 int count = 0;
341 if ( is_selected() ) {
342 deselect();
343 ++count;
344 }
345 for ( int t=0; t<children(); t++ ) {
346 count += child(t)->deselect_all();
347 }
348 return(count);
349 } 
351 char is_selected() const {
352 return is_flag(SELECTED);
353 } 
359 void activate(int val=1) {
360 set_flag(ACTIVE, val);
361 if ( _widget && val != (int)_widget->active() ) {
362 if ( val ) 
363 _widget->activate();
364 } else {
365 _widget->deactivate();
366 }
367 _widget->redraw();
368 } 
370 void deactivate() {
371 activate(0);
372 } 
376 void deactivate() {
377 activate(0);
378 } 
380 char is_activated() const {
381 return is_flag(ACTIVE);
382 } 
384 char is_active() const {
385 return is_activated();
386 } 
388 int visible() const {
389 return is_flag(VISIBLE);
390 } 
392 int is_visible() const {
393 return is_flag(VISIBLE);
394 } 
395 int visible_r() const;
396 406 void usericon(Fl_Image * val) {
407 usericon = val;
408 recalc_tree(); // may change tree geometry
409 } 
411 Fl_Image *usericon() const {
412 return(usericon);
413 } 
440 void userdeicon(Fl_Image * val) {
# Fl_Tree_Item_Array.H File Reference

This file defines a class that manages an array of `Fl_Tree_Item` pointers.

```cpp
#include <FL/Fl.H>
#include "Fl_Export.H"
```

## Classes

- `class Fl_Tree_Item_Array`

  Manages an array of `Fl_Tree_Item` pointers.

## Detailed Description

This file defines a class that manages an array of `Fl_Tree_Item` pointers.

## Fl_Tree_Item_Array.H

Go to the documentation of this file.

```cpp
1 //
2 #ifndef _FL_TREE_ITEM_ARRAY_H
3 #define _FL_TREE_ITEM_ARRAY_H
4 #include <FL/Fl.H>
5 #ifndef FL_TREE_ITEM_ARRAY_H
6 #include "Fl_Export.H"
7 #ifndef FL_TREE_EXPORT
8 #define FL_TREE_EXPORT Fl_Tree_Item; // forward decl must precede first doxygen comment block
9 #endif
10 // FL/Fl_Tree_Item_Array.H
11 //
12 // Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
13 // Copyright (C) 2009-2010 by Greg Ercolano.
14 //
15 // This library is free software. Distribution and use rights are outlined in
16 // the file "COPYING" which should have been included with this file. If this
17 // file is missing or damaged, see the license at:
18 //
19 // https://www.fltk.org/COPYING.php
20 //
21 // Generated by Doxygen
```
This file contains the definitions for Fl_Tree's preferences.
#include <FL/Fl.H>

Classes

- class Fl_Tree_Prefs
  
  Tree widget's preferences.

Typedefs

- typedef void() Fl_Tree_Item_Draw_Callback(Fl_Tree_Item *, void *)

Enumerations

- enum Fl_Tree_Connector { FL_TREE_CONNECTOR_NONE =0 , FL_TREE_CONNECTOR_DOTTED =1 , FL_TREE_CONNECTOR_SOLID =2 }

  Defines the style of connection lines between items.

- enum Fl_Tree_Item_Draw_Mode { FL_TREE_ITEM_DRAW_DEFAULT =0 , FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET =1 , FL_TREE_ITEM_HEIGHT_FROM_WIDGET =2 }

  Bit flags that control how item's labels and widget(s) are drawn in the tree via item_draw_mode().
• enum Fl_Tree_Item_Reselect_Mode { FL_TREE_SELECTABLE_ONCE =0 , FL_TREE_SELECTABLE_ALWAYS }

Defines the ways an item can be (re) selected via item_reselect_mode().

• enum Fl_Tree_Select { FL_TREE_SELECT_NONE =0 , FL_TREE_SELECT_SINGLE =1 , FL_TREE_SELECT_MULTI =2 , FL_TREE_SELECT_SINGLE_DRAGGABLE =3 }

Tree selection style.

• enum Fl_Tree_Sort { FL_TREE_SORT_NONE =0 , FL_TREE_SORT_ASCENDING =1 , FL_TREE_SORT_DESCENDING =2 }

Sort order options for items added to the tree.

34.165.1 Detailed Description

This file contains the definitions for Fl_Tree's preferences.

Fl_Tree_Prefs

: ..: : :

Fl_Tree : Fl_Tree_Item

34.165.2 Enumeration Type Documentation

34.165.2.1 Fl_Tree_Connector

denum Fl_Tree_Connector

Defines the style of connection lines between items.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_CONNECTOR_NONE</td>
<td>Use no lines connecting items.</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_DOTTED</td>
<td>Use dotted lines connecting items (default)</td>
</tr>
<tr>
<td>FL_TREE_CONNECTOR_SOLID</td>
<td>Use solid lines connecting items.</td>
</tr>
</tbody>
</table>

34.165.2.2 Fl_Tree_Item_Draw_Mode

denum 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>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_ITEM_DRAW_DEFAULT</td>
<td>If widget() defined, draw in place of label, and widget() tracks item height (default)</td>
</tr>
<tr>
<td>FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET</td>
<td>If widget() defined, include label to the left of the widget.</td>
</tr>
<tr>
<td>FL_TREE_ITEM_HEIGHT_FROM_WIDGET</td>
<td>If widget() defined, widget()'s height controls item's height.</td>
</tr>
</tbody>
</table>

34.165.2.3 Fl_Tree_Item_Reselect_Mode

denum Fl_Tree_Item_Reselect_Mode

Defines the ways an item can be (re) selected via item_reselect_mode().

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECTABLE_ONCE</td>
<td>Item can only be selected once (default)</td>
</tr>
</tbody>
</table>
## 34.166 Fl_Tree_Prefs.H

Go to the documentation of this file.

```
1 //
2 #ifndef FL_TREE_PREFS_H
3 #define FL_TREE_PREFS_H
4 #include <FL/Fl.H> // needed for ABI version features (via Enumerations.H)
6 // FL/Fl_Tree_Prefs.H
8 // Fl_Tree_Prefs -- This file is part of the Fl_Tree widget for FLTK
9 // Copyright (C) 2009-2010 by Greg Ercolano.
11 // This library is free software. Distribution and use rights are outlined in
12 // the file "COPYING" which should have been included with this file. If this
13 // file is missing or damaged, see the license at:
15 // https://www.fltk.org/COPYING.php
17 // Please see the following page on how to report bugs and issues:
19 // https://www.fltk.org/bugs.php
21 enum Fl_Tree_Sort {
23     FL_TREE_SORT_NONE=0,
25     FL_TREE_SORT_ASCENDING=1,
```

#### 34.165.2.4 Fl_Tree_Select

```
enum Fl_Tree_Select
Tree selection style.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SELECT_NONE</td>
<td>Nothing selected when items are clicked.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE</td>
<td>Single item selected when item is clicked (default)</td>
</tr>
<tr>
<td>FL_TREE_SELECT_MULTI</td>
<td>Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.</td>
</tr>
<tr>
<td>FL_TREE_SELECT_SINGLE_DRAGGABLE</td>
<td>Single items may be selected, and they may be reordered by mouse drag.</td>
</tr>
</tbody>
</table>

#### 34.165.2.5 Fl_Tree_Sort

```
enum Fl_Tree_Sort
Sort order options for items added to the tree.
```

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_TREE_SORT_NONE</td>
<td>No sorting; items are added in the order defined (default).</td>
</tr>
<tr>
<td>FL_TREE_SORT_ASCENDING</td>
<td>Add items in ascending sort order.</td>
</tr>
<tr>
<td>FL_TREE_SORT_DESCENDING</td>
<td>Add items in descending sort order.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
```
enum Fl_Tree_Connector {
    FL_TREE_CONNECTOR_NONE = 0,
    FL_TREE_CONNECTOR_DOTTED = 1,
    FL_TREE_CONNECTOR_SOLID = 2
};

enum Fl_Tree_Select {
    FL_TREE_SELECT_NONE = 0,
    FL_TREE_SELECT_SINGLE = 1,
    FL_TREE_SELECT_MULTI = 2,
    FL_TREE_SELECT_SINGLE_DRAGGABLE = 3
};

enum Fl_Tree_Item_Reselect_Mode {
    FL_TREE_SELECTABLE_ONCE = 0,
    FL_TREE_SELECTABLE_ALWAYS
};

enum Fl_Tree_Item_Draw_Mode {
    FL_TREE_ITEM_DRAW_DEFAULT = 0,
    FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET = 1,
    FL_TREE_ITEM_HEIGHT_FROM_WIDGET = 2
};

public:

Fl_Tree_Prefs();

Fl_Tree_Prefs();
```

inline Fl_Color labelbgcolor() const { return(item_labelbgcolor()); }
inline void labelbgcolor(Fl_Color val) { item_labelbgcolor(val); }

// Margins
inline int marginleft() const {
    return(_marginleft);
}
inline void marginleft(int val) {
    _marginleft = val;
}
inline int margintop() const {
    return(_margintop);
}
inline void margintop(int val) {
    _margintop = val;
}
inline int marginbottom() const {
    return(_marginbottom);
}
inline void marginbottom(int val) {
    _marginbottom = val;
}
inline int openchild_marginbottom() const {
    return(_openchild_marginbottom);
}
inline void openchild_marginbottom(int val) {
    _openchild_marginbottom = val;
}
inline int usericonmarginleft() const {
    return(_usericonmarginleft);
}
inline void usericonmarginleft(int val) {
    _usericonmarginleft = val;
}
inline int labelmarginleft() const {
    return(_labelmarginleft);
}
inline void labelmarginleft(int val) {
    _labelmarginleft = val;
}
inline int widgetmarginleft() const {
    return(_widgetmarginleft);
}
inline void widgetmarginleft(int val) {
    _widgetmarginleft = val;
}
inline int linespacing() const {
    return(_linespacing);
}
inline void linespacing(int val) {
    _linespacing = val;
}

// Colors and Styles
inline Fl_Color connectorcolor() const {
    return(_connectorcolor);
}
inline void connectorcolor(Fl_Color val) {
    _connectorcolor = val;
}
inline Fl_Tree_Connector connectorstyle() const {
    return(_connectorstyle);
}
inline void connectorstyle(Fl_Tree_Connector val) {
    _connectorstyle = val;
}
inline void connectorstyle(int val) {
    _connectorstyle = Fl_Tree_Connector(val);
}
inline int connectorwidth() const {
    return(_connectorwidth);
}
inline void connectorwidth(int val) {
    _connectorwidth = val;
}

// Icons
inline Fl_Image *openicon() const {
    return(_openimage);
}
void openicon(Fl_IMAGE *val);
inline Fl_Image *closeicon() const {
    return(_closeimage);
}
void closeicon(Fl_IMAGE *val);
inline Fl_Image *usericon() const {
    return(_userimage);
inline void usericon(Fl_Image *val) {
    _userimage = val;
    // Update deactivated version of icon..
    if ( _userdeimage ) delete _userdeimage;
    if ( _userimage ) {
        _userdeimage = _userimage->copy();
        _userimage->inactive();
    } else {
        _userdeimage = 0;
    }
}

inline Fl_Image *opendeicon() const {
    return _opendeimage;
}

inline Fl_Image *closedeicon() const {
    return _closedeimage;
}

inline Fl_Image *userdeicon() const {
    return _userdeimage;
}

// Options
inline char showcollapse() const {
    return _showcollapse;
}

inline void showcollapse(int val) {
    _showcollapse = val;
}

inline Fl_Tree_Sort sortorder() const {
    return _sortorder;
}

inline void sortorder(Fl_Tree_Sort val) {
    _sortorder = val;
}

inline Fl_Boxtype selectbox() const {
    return _selectbox;
}

inline void selectbox(Fl_Boxtype val) {
    _selectbox = val;
}

inline int showroot() const {
    return _showroot;
}

inline void showroot(int val) {
    _showroot = char(val);
}

inline Fl_Tree_Select selectmode() const {
    return _selectmode;
}

inline void selectmode(Fl_Tree_Select val) {
    _selectmode = val;
}

inline Fl_Tree_Item_Reselect_Mode item_reselect_mode() const {
    return _itemreselectmode;
}

void item_reselect_mode(Fl_Tree_Item_Reselect_Mode mode) {
    _itemreselectmode = mode;
}

inline Fl_Tree_Item_Draw_Mode item_draw_mode() const {
    return _itemdrawmode;
}

inline void item_draw_mode(Fl_Tree_Item_Draw_Mode val) {
    _itemdrawmode = val;
}

void item_draw_callback(Fl_Tree_Item_Draw_Callback *cb, void *data=0) {
    _itemdrawcallback = cb;
    _itemdrawuserdata = data;
}

void *item_draw_user_data() const {
    return _itemdrawuserdata;
}

void do_item_draw_callback(Fl_Tree_Item *o) const {
    _itemdrawcallback(o, _itemdrawuserdata);
}

};

#endif /*FL_TREE_PREFS_H*/
This file contains simple "C"-style type definitions.

```c
#include "fl_attr.h"
```

## Typedefs

### Miscellaneous

- `typedef unsigned int Fl_Shortcut`  
  16-bit Unicode character + 8-bit indicator for keyboard flags.
- `typedef unsigned char uchar`  
  `unsigned char`
- `typedef unsigned long ulong`  
  `unsigned long`

### Detailed Description

This file contains simple "C"-style type definitions.

### Typedef Documentation

#### Fl_Shortcut

```c
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.
#define FL_TYPES_H

#include "fl_attr.h"

/* group: Miscellaneous */

typedef unsigned char uchar;
typedef unsigned long ulong;

/* group: Miscellaneous */

typedef unsigned int Fl_Shortcut;

#endif

34.169   fl_utf8.h File Reference

header for Unicode and UTF-8 character handling

#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <sys/stat.h>

Functions

• int fl_access (const char *f, int mode)
  Cross-platform function to test a files access() with a UTF-8 encoded name or value.

• int fl_chdir (const char *path)
  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.

• int fl_chmod (const char *f, int mode)
  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.

• int fl_close_fd (int fd)
  Cross-platform function to close a file descriptor.

• int fl_execvp (const char *file, char **const argv)
  FILE * fl_fopen (const char *f, const char *mode)
  Cross-platform function to open files with a UTF-8 encoded name.

• char * fl_getcwd (char *buf, int len)
  Cross-platform function to get the current working directory as a UTF-8 encoded value.

• char * fl_getenv (const char *v)
  Cross-platform function to get environment variables with a UTF-8 encoded name or value.

• char * fl_make_path (const char *path)
  Cross-platform function to recursively create a path in the file system.

• void fl_make_path_for_file (const char *path)
  Cross-platform function to create a path for the file in the file system.

• int fl_mkdir (const char *f, int mode)
  Cross-platform function to create a directory with a UTF-8 encoded name.

• unsigned int fl_nonspacing (unsigned int ucs)
  Returns true if the Unicode character ucs is non-spacing.

• int fl_open (const char *fname, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.

• int fl_open_ext (const char *fname, int binary, int oflags,...)
  Cross-platform function to open files with a UTF-8 encoded name.

• int fl_putenv (const char *var)
  Cross-platform function to write environment variables with a UTF-8 encoded name or value.

• int fl_rename (const char *f, const char *n)
  Cross-platform function to rename a filesystem object using UTF-8 encoded names.

• int fl_rmdir (const char *f)
Cross-platform function to remove a directory with a UTF-8 encoded name.

- int fl_stat (const char *f, struct stat *b)
  Cross-platform function to stat() a file using a UTF-8 encoded name or value.

- int fl_system (const char *cmd)
  Cross-platform function to run a system command with a UTF-8 encoded string.

- int fl_toupper (unsigned int ucs)
  Returns the Unicode upper case value of ucs.

- unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)
  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.

- int fl_unlink (const char *fname)
  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.

- char * fl_utf2mbcs (const char *s)
  Converts UTF-8 string s to a local multi-byte character string.

- const char * fl_utf8back (const char *p, const char *start, const char *end)
  Move p backward until it points to the start of a UTF-8 character.

- int fl_utf8bytes (unsigned ucs)
  Return the number of bytes needed to encode the given UCS4 character in UTF-8.

- unsigned fl_utf8decode (const char *p, const char *end, int *len)
  Decode a single UTF-8 encoded character starting at p.

- int fl_utf8encode (unsigned ucs, char *buf)
  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.

- unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)
  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.

- unsigned fl_utf8fromchar (char *dst, unsigned dstlen, char *src, unsigned srclen)
  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.

- unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)
  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.

- const char * fl_utf8fwd (const char *p, const char *start, const char *end)
  Move p forward until it points to the start of a UTF-8 character.

- int fl_utf8len (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.

- int fl_utf8len1 (char c)
  Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.

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

- int fl_utf8strlen (const char *text, int len)
  Return the length in bytes of a UTF-8 string.

- int fl_utf8test (const char *src, unsigned srclen)
  Examines the first srclen bytes in src and returns a verdict on whether it is UTF-8 or not.

- unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).

- unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 1-byte characters.

- unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)
  Convert a UTF-8 sequence into an array of 16-bit characters.

- unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
  Converts a UTF-8 string into a wide character string.

- int fl_utf_nb_char (const char *buf, int len)
Returns the number of Unicode chars in the UTF-8 string.

- int fl_utf_strcasecmp (const char *s1, const char *s2)
  UTF-8 aware strcasecmp - converts to Unicode and tests.
- int fl_utf_strncasecmp (const char *s1, const char *s2, int n)
  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
- int fl_utf_tolower (const unsigned char *str, int len, char *buf)
  Converts the string str to its lower case equivalent into buf.
- int fl_utf_toupper (const unsigned char *str, int len, char *buf)
  Converts the string str to its upper case equivalent into buf.
- int fl_wcwidth (const char *src)
  extended wrapper around fl_wcwidth(unsigned int ucs) function.
- int fl_wcwidth_ (unsigned int ucs)
  Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

34.169.1 Detailed Description

header for Unicode and UTF-8 character handling

34.170 fl_utf8.h

Go to the documentation of this file.
34.170 fl_utf8.h 1381

76 /* F2: Encode a Unicode value into a UTF-8 sequence, return the number of bytes used */
77 FLEXPORT int fl_utf8encode(unsigned ucs, char* buf);
78
79 /* F2: Move forward to the next valid UTF-8 sequence start between start and end */
80 FLEXPORT const char* fl_utf8fwd(const char* p, const char* start, const char* end);
81
82 /* F2: Move backward to the previous valid UTF-8 sequence start */
83 FLEXPORT const char* fl_utf8back(const char* p, const char* start, const char* end);
84
85 /* XX: Convert a single 32-bit Unicode value into UTF16 */
86 FLEXPORT unsigned fl_ucs_to_Utf16(const unsigned ucs, unsigned short* dst, const unsigned dstlen);
87
88 /* F2: Convert a UTF-8 string into UTF16 - makes UTF16 on win32, *UCS4* elsewhere */
89 FLEXPORT unsigned fl_utf8toUtf16(const char* src, unsigned srclen, unsigned short* dst, unsigned dstlen);
90
91 /* F2: Convert a wide character string to UTF-8 - takes in UTF16 on win32, *UCS4* elsewhere */
92 FLEXPORT unsigned fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen);
93
94 /* F2: Convert a UTF-8 string into ASCII, eliding untranslatable glyphs */
95 FLEXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen);
96
97 /* F2: Convert 8859-1 string to UTF-8 */
98 FLEXPORT unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen);
99
100 /* F2: Returns true if the current O/S locale is UTF-8 */
101 FLEXPORT int fl_utf8locale(void);
102
103 /* XX: Examine the first len characters of src, to determine if the input text is UTF-8 or not */
104 FLEXPORT int fl_utf8test(const char *src, unsigned len);
105
106 /* XX: return width of "raw" ucs character in columns. */
107 FLEXPORT int fl_wcwidth_(unsigned int ucs);
108
109 /* XX: return width of UTF-8 character string in columns. */
110 FLEXPORT int fl_wcwidth(const char *src);
111
112 /* OD: Return true if the character is non-spacing */
113 FLEXPORT unsigned int fl_nonspacing(unsigned int ucs);
114
115 /* XX: Convert UTF-8 to a local multi-byte encoding - mainly for win32 */
116 FLEXPORT unsigned fl_utf8to_mb(const char *src, unsigned srclen, char *dst, unsigned dstlen);
117
118 /* OD: utf8 from mbcs */
119 FLEXPORT int fl_utf2mbcs(const char *src);
120
121 /* XX: Convert a local multi-byte encoding to UTF-8 - mainly for win32 */
122 FLEXPORT unsigned fl_utf8from_mb(char *dst, unsigned dstlen, const char *src, unsigned srclen);
123
124 /*****************************************************************************/
125 #ifdef _WIN32
126 /* these two Windows-only functions are kept for API compatibility */
127 #define WIN32
128
129 /* OD: Attempt to convert a string in the current locale to UTF-8 */
130 FLEXPORT char *fl_locale_to_utf8(const char *s, int len, unsigned int codepage);
131#endif /*_WIN32 */
132
133 /*****************************************************************************/
134 /* OD: UTF-8 aware strncasecmp - converts to lower case Unicode and tests */
135 FLEXPORT int fl_utf_strncasecmp(const char *s1, const char *s2, int n);
136
137 /* OD: convert the Unicode lower case value of ucs */
138 FLEXPORT int fl_tolower(unsigned int ucs);
139
140 /* OD: return the Unicode upper case value of ucs */
141 FLEXPORT int fl_toupper(unsigned int ucs);
142
143 /* OD: converts the UTF-8 string to the lower case equivalent */
144 FLEXPORT int fl_utf_tolower(const unsigned char *str, int len, char *buf);
145
146 Generated by Doxygen
/* OD: converts the UTF-8 string to the upper case equivalent */
FL_EXPORT int fl_utf_toupper(const unsigned char *str, int len, char *buf);
/* OD: Portable UTF-8 aware chmod wrapper */
FL_EXPORT int fl_chmod(const char *f, int mode);
/* OD: Portable UTF-8 aware access wrapper */
FL_EXPORT int fl_access(const char *f, int mode);
/* OD: Portable UTF-8 aware stat wrapper */
FL_EXPORT int fl_stat(const char *path, struct stat *buffer);
/* OD: Portable UTF-8 aware getcwd wrapper */
FL_EXPORT char *fl_getcwd(char *buf, int len);
/* Portable UTF-8 aware chdir wrapper */
FL_EXPORT int fl_chdir(const char *path);
/* OD: Portable UTF-8 aware fopen wrapper */
FL_EXPORT FILE *fl_fopen(const char *f, const char *mode);
/* OD: Portable UTF-8 aware system wrapper */
FL_EXPORT int fl_system(const char* f);
/* OD: Portable UTF-8 aware execvp wrapper */
FL_EXPORT int fl_execvp(const char *file, char *const *argv);
/* OD: Portable UTF-8 aware open wrapper */
FL_EXPORT int fl_open(const char *fname, int oflags, ...);
FL_EXPORT int fl_open_ext(const char *fname, int binary, int oflags, ...);
/* Portable wrapper around unix-style close() function */
FL_EXPORT int fl_close_fd(int fd);
/* OD: Portable UTF-8 aware unlink wrapper */
FL_EXPORT int fl_unlink(const char *fname);
/* OD: Portable UTF-8 aware rmdir wrapper */
FL_EXPORT int fl_rmdir(const char *f);
/* OD: Portable UTF-8 aware rename wrapper */
FL_EXPORT int fl_rename(const char* f, const char *t);
/* OD: Given a full pathname, this will create the directory path needed to hold the file named */
FL_EXPORT void fl_make_path_for_file( const char *path );
/* OD: recursively create a path in the file system */
FL_EXPORT char fl_make_path( const char *path );

34.171 Fl_Valuator.H

// Valuator header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:

Generated by Doxygen
Fl_Valuator widget.

```
#ifndef Fl_Valuator_H
#define Fl_Valuator_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// shared type() values for classes that work in both directions:
#define FL_VERTICAL 0
#define FL_HORIZONTAL 1

class FL_EXPORT Fl_Valuator : public Fl_Widget {
    double value_, previous_value_; // truncates to this range *after* rounding
    double min, max; // rounds to multiples of A/B, or no rounding if A is zero

protected:
    int horizontal()const {return type()& FL_HORIZONTAL;}
    Fl_Valuator(int X, int Y, int W, int H, const char * L);

double previous_value()const {return previous_value_;
    void handle_push() {previous_value_ = value_;}
    double softclamp(double);
    void handle_drag(double newvalue);
    void handle_release(); // use drag() value
    virtual void value_damage(); // cause damage() due to value() changing
    void set_value(double v) {value_ = v;

public:
    int horizontal()const {return type()& FL_HORIZONTAL;}
    Fl_Valuator(int X, int Y, int W, int H, const char * L);

double minimum()const {return min;}
void minimum(double a) {min = a;}
double maximum()const {return max;}
void maximum(double a) {max = a;}
void range(double a, double b) {min = a; max = b;}
void step(int a) {A = a; B = 1;}
void step(double a, int b) {A = a; B = b;}
void step(double s);
    double step()const {return A/B;}
    void precision(int digits);

    double value()const {return value_;}
    int value(double);

    virtual int format(char *);
    double round(double); // round to nearest multiple of step
    double clamp(double); // keep in range
    double increment(double, int); // add n*step to value

};
```

#endif
```
```cpp
#include "Fl_Valuator.H"
#include "Fl_Input.H"

class FL_EXPORT Fl_Value_Input : public Fl_Valuator {
  public:
    // This is the encapsulated Fl_input attribute to which
    // this class delegates the value font, color and shortcut
    Fl_Input input;
  private:
    char soft_;
  static void input_cb(Fl_Widget *, void *);
  void value_damage() FL_OVERRIDE; // cause damage() due to value() changing
  public:
  int handle(int) FL_OVERRIDE;
  protected:
  void draw() FL_OVERRIDE;
  public:
  void resize(int, int, int, int) FL_OVERRIDE;
  Fl_Value_Input(int x, int y, int w, int h, const char *l=0);
  ~Fl_Value_Input();
  void soft(char s) {soft_ = s;}
  char soft() const {return soft_;
  int shortcut() const {return input.shortcut();}
  void shortcut(int s) {input.shortcut(s);
  Fl_Font textfont() const {return input.textfont();}
  void textfont(Fl_Font s) {input.textfont(s);
  Fl_Fontsize textsize() const {return input.textsize();}
  void textsize(Fl_Fontsize s) {input.textsize(s);
  Fl_Color textcolor() const {return input.textcolor();}
  void textcolor(Fl_Color n) {input.textcolor(n);
  Fl_Color cursor_color() const {return input.cursor_color();}
  void cursor_color(Fl_Color n) {input.cursor_color(n);

34.173 Fl_Value_Output.H

// Value output header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

class FL_EXPORT Fl_Value_Output : public Fl_Valuator {
  Fl_Font textfont_; 
  Fl_Fontsize textsize_; 
  uchar soft_; 
  Fl_Color textcolor_; 
  protected:
  void draw() FL_OVERRIDE;
  public:
  int handle(int) FL_OVERRIDE;
  Fl_Value_Output(int x, int y, int w, int h, const char *l=0);
  void soft(uchar s) {soft_ = s;}
  uchar soft() const {return soft_;}
  Fl_Font textfont() const {return textfont_;}
  void textfont(Fl_Font s) {textfont_ = s;}

Generated by Doxygen
34.174 Fl_Value_Slider.H

1 //
2 // Value Slider header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 /*
17  * Fl_Value_Slider widget. */
18 */
19 #ifndef Fl_Value_Slider_H
20 #define Fl_Value_Slider_H
21 #include "Fl_Slider.H"
22 #include "Fl_Font.H"
23
24 class FL_EXPORT Fl_Value_Slider : public Fl_Slider {
25 Fl_Font textfont_;   // Text font
26 Fl_Fontsize textsize_; // Text size
27 Fl_Color textcolor_;   // Text color
28 short value_width_;   // Value width
29 short value_height_;   // Value height
30
31 protected:
32 void draw() FL_OVERRIDE;
33 int handle(int) FL_OVERRIDE;
34 Fl_Value_Slider();
35 Fl_Value_Slider(int x, int y, int w, int h, const char *l = 0);
36
37 public:
38 int textfont()const { return textfont_; }
39 void textfont(Fl_Font s) { textfont_ = s; }
40 Fl_Fontsize textsize()const { return textsize_; }
41 void textsize(Fl_Fontsize s) { textsize_ = s; }
42 Fl_Color textcolor()const { return textcolor_; }
43 void textcolor(Fl_Color s) { textcolor_ = s; }
44
45 void value_width(int s) {
46 if (s > w() - 10) s = w() - 10;
47 if (s < 10) s = 10;
48 value_width_ = (short)s;
49 }
50 int value_width()const { return (value_width_); }
51
52 void value_height(int s) {
53 if (s > h() - 10) s = h() - 10;
54 if (s < 10) s = 10;
55 value_height_ = (short)s;
56 }
57 int value_height()const { return (value_height_); }
58 #endif

34.175  Fl_Widget.H File Reference

Fl_Widget and Fl_Label classes.
#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.

34.175.1 Detailed Description

Fl_Widget and Fl_Label classes.

34.175.2 Macro Definition Documentation

34.175.2.1 FL_RESERVED_TYPE

#define FL_RESERVED_TYPE 100
Reserved type numbers (necessary for my cheapo RTTI) start here.
Grep the header files for "RESERVED_TYPE" to find the next available number.

34.176  Fl_Widget.H

Go to the documentation of this file.
1 //
2 // Widget header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2023 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //  https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //  https://www.fltk.org/bugs.php
15 //
Generated by Doxygen
#ifndef Fl_Widget_H
#define Fl_Widget_H

#include "Fl.H"

class Fl_Widget;
class Fl_Window;
class Fl_Group;
class Fl_Image;

typedef void (Fl_Callback *)(Fl_Widget*, void*);
typedef Fl_Callback* Fl_Callback_; // needed for BORLAND

typedef void (Fl_Callback0)(Fl_Widget*);
typedef void (Fl_Callback1)(Fl_Widget*, long);

struct FL_EXPORT Fl_Label {
    const char * value;
    Fl_Image * image;
    Fl_Image * deimage;
    Fl_Font font;
    Fl_Fontsize size;
    Fl_Color color;
    Fl_Align align_;  
    uchar type;
public:
    void draw(int,int,int,int, Fl_Align) const ;
    void measure(int &w, int &h) const ;
};

class FL_EXPORT Fl_Widget {  
friend class Fl_Group;

private:
    Fl_Group * parent_;  
    Fl_Callback * callback_;  
    void * user_data_;  
    int x_,y_,w_,h_;  
    unsigned int flags_;  
    Fl_Label label_;  
    Fl_Font size ;  
    uchar type_;  
    uchar damage_;  
    uchar box_;  
    uchar when_;  
    const char *tooltip_;  

public:
    Fl_Widget(const Fl_Widget &);  
    Fl_Widget& operator=(const Fl_Widget &);  

protected:
    Fl_Widget(int x, int y, int w, int h, const char *label=0L);  

    void * x(int v) {x_ = v;}
    void * y(int v) {y_ = v;}
    void * w(int v) {w_ = v;}
    void * h(int v) {h_ = v;}
    unsigned int flags()const {return flags_;
    void set_flag(unsigned int c) {flags_ |= c;
    void clear_flag(unsigned int c) {flags_ &= ~c;
    unsigned int flags()const {return flags_;
    void set_flag(unsigned int c) {flags_ |= c;
    void clear_flag(unsigned int c) {flags_ &= ~c;
    void draw(int,int,int,int, Fl_Align) const ;
    void measure(int &w, int &h) const ;

    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,
        OVERFLOW = 1«7,
        VISIBLE_FOCUS = 1«8,
        COPIED_LABEL = 1«9,
        CLIP_CHILDREN = 1«10,
        MENU_WINDOW = 1«11,
        TOOLKIT_WINDOW = 1«12,
        MODAL = 1«13,
        NO_OVERLAY = 1«14,
        GROUP_RELATIVE = 1«15,
        COPIED_TOOLKIT = 1«16,
        FULLSCREEN = 1«17,
        MAC_USE_ACCESSIBLE_MENU = 1«18,
        NEEDS_KEYBOARD = 1«19,
        IMAGE_BOUND = 1«20,
        needs_keyboard = 1«21,
};

Generated by Doxygen
DEIMAGE_BOUND = 1«22,
// Note to devs: add new FLTK core flags above this line (up to 1«28).
// Three more flags, reserved for user code
USERFLAG3 = 1«29,
USERFLAG2 = 1«30,
USERFLAG1 = 1«31
;
void draw_box() const;
void draw_box(Fl_Boxtype t, Fl_Color c) const;
void draw_box(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const;
void draw_backdrop() const;
void draw_focus() const {
  draw_focus(box(), x(), y(), w(), h(), color());
}
void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H, Fl_Color bg) const;
void draw_label() const;
void draw_label(int, int, int, int) const;
public:
  virtual ~Fl_Widget();
  virtual void draw() = 0;
  virtual int handle(int event);
  int is_label_copied() const { return ((flags_ & Fl::SEQUENCE) ? 1 : 0); }
  void needs_keyboard(bool needs) {
    if (needs) set_flag(Fl::NEEDS_KEYBOARD);
    else clear_flag(Fl::NEEDS_KEYBOARD);
  }
  bool needs_keyboard() const {
    return (flags_ & Fl::NEEDS_KEYBOARD);
  }
  Fl_Group* parent() const { return parent_; }
  uchar type() const { return type_; }
  void type(uchar t) { type_ = t; }
  int x() const { return x_; }
  int y() const { return y_; }
  int w() const { return w_; }
  int h() const { return h_; }
  virtual void resize(int x, int y, int w, int h);
  int damage_resize(int, int, int, int);
  void position(int X, int Y) { resize(X, Y, w_, h_); }
  void size(int W, int H) { resize(X_, Y_, W, H); }
  Fl_Align align() const { return label_.align_; }
  void align(Fl_Align alignment) { label_.align_ = alignment; }
  Fl_Boxtype box() const { return box_; }
  void box(Fl_Boxtype new_box) { box_ = new_box; }
  Fl_Color color() const { return color_; }
  void color(Fl_Color bg) { color_ = bg; }
  Fl_Color selection_color() const { return color2_; }
  void selection_color(Fl_Color a) { color2_ = a; }
Generated by Doxygen
void color(Fl_Color bg, Fl_Color sel) {color_ = bg; color2_ = sel;}

const char * label() const {return label_.value;}

void label(const char * text);

void copy_label(const char *new_label);

void label(Fl_Labeltype a, const char * b) {label_.type = a; label_.value = b;}

Fl_Labeltype labeltype() const {return (Fl_Labeltype)label_.type;}

void labeltype(Fl_Labeltype a) {label_.type = a;}

Fl_Color labelcolor() const {return label_.color;}

void labelcolor(Fl_Color c) {label_.color = c;}

Fl_Font labelfont() const {return label_.font;}

void labelfont(Fl_Font f) {label_.font = f;}

Fl_Fontsize labelsize() const {return label_.size;}

void labelsize(Fl_Fontsize pix) {label_.size = pix;}

Fl_Image * image() {return label_.image;}

const Fl_Image * image() const {return label_.image;}

void deimage(Fl_Image * img);

void deimage(Fl_Image & img);

void bind_deimage(Fl_Image * img);

int deimage_bound() const {return ((flags_ & DEIMAGE_BOUND) ? 1 : 0);}

void bind_deimage(int f) {if (f) set_flag(DEIMAGE_BOUND); else clear_flag(DEIMAGE_BOUND);}

const char * tooltip() const {return tooltip_;

void tooltip(const char *text); // see Fl_Tooltip

void copy_tooltip(const char *text); // see Fl_Tooltip

Fl_Callback_p callback() const {return callback_;}

void callback(Fl_Callback * cb, void * p) {callback_ = cb; user_data_ = p;}

void callback(Fl_Callback * cb) {callback_ = cb;}

void callback(Fl_Callback0 * cb) {
    callback_ = (Fl_Callback*)(fl_intptr_t)cb;
}

void callback(Fl_Callback1 * cb, long p = 0) {
    callback_ = (Fl_Callback*)(fl_intptr_t)cb;
    user_data_ = (void*) (fl_intptr_t)p;
}

long argument() const {return (long)(fl_intptr_t)user_data_;

void argument(long v) {user_data_ = (void*) (fl_intptr_t)v;

Fl_When when() const {return (Fl_When) when_;

void when(uchar i) {when_ = i;}

Generated by Doxygen
unsigned int visible() const { return !(flags_ & INVISIBLE); }

int visible_r() const;

virtual void show();

virtual void hide();

void set_visible() { flags_ &= ~INVISIBLE; }

void clear_visible() { flags_ |= INVISIBLE; }

unsigned int active() const { return !(flags_ & INACTIVE); }

int active_r() const;

virtual void activate();

virtual void deactivate();

unsigned int output() const { return (flags_ & OUTPUT); }

void set_output() { flags_ |= OUTPUT; }

void clear_output() { flags_ &= ~OUTPUT; }

unsigned int takesevents() const { return !(flags_ & (INACTIVE | INVISIBLE | OUTPUT)); }

unsigned int changed() const { return flags_ & CHANGED; }

void set_changed() { flags_ |= CHANGED; }

void clear_changed() { flags_ &= ~CHANGED; }

void clear_active() { flags_ |= INACTIVE; }

void set_active() { flags_ &= ~INACTIVE; }

int take_focus();

void set_visible_focus() { flags_ |= VISIBLE_FOCUS; }

void clear_visible_focus() { flags_ &= ~VISIBLE_FOCUS; }

void visible_focus(int v) { if (v) set_visible_focus(); else clear_visible_focus(); }

unsigned int visible_focus() const { return flags_ & VISIBLE_FOCUS; }

static void default_callback(Fl_Widget *widget, void *data);

void do_callback(Fl_Callback_Reason reason = FL_REASON_UNKNOWN) { do_callback(this, user_data_, reason); }

void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason = FL_REASON_UNKNOWN) {
  do_callback(widget, (void *)(fl_intptr_t)arg, reason);
}

void do_callback(Fl_Widget *widget, void *arg = 0, Fl_Callback_Reason reason = FL_REASON_UNKNOWN);

/* Internal use only. */
int test_shortcut();

/* Internal use only. */
static unsigned int labelShortcut(const char *t);

/* Internal use only. */
static int test_shortcut(const char*, const bool require_alt = false);

void _set_fullscreen() { flags_ |= FULLSCREEN; }

void _clear_fullscreen() { flags_ &= ~FULLSCREEN; }

int contains(const Fl_Widget *w) const;

int inside(const Fl_Widget *wgt) const { return wt->contains(this) == 0; }

void redraw();

void redraw_label();

uchar damage() const { return damage_; }

void clear_damage(uchar c = 0) { damage_ = c; }

void damage(uchar c);

void damage(uchar c, int x, int y, int w, int h);

void draw_label(int, int, int, int, Fl_Align) const;
virtual Fl_Group* as_group() const { return NULL; }
virtual Fl_Window* const as_window() const { return NULL; }
virtual Fl_Group* as_group() { return NULL; }
virtual Fl_Window* const as_window() const { return NULL; }
virtual class Fl_GL_Window* as_gl_window() const { return NULL; }
virtual class Fl_GL_Window* as_gl_window() { return NULL; }
virtual Fl_Color color2() const { return (Fl_Color)color2_; }
virtual Fl_Color* color2() { return &color2_; }
void color2(unsigned a) { color2_ = a; }
int shortcut_label() const { return flags_ & SHORTCUT_LABEL; }
};

#define FL_RESERVED_TYPE 100

#endif /* Fl_Widget_Surface_h */
#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

34.178.1 Detailed Description

Fl_Window widget.

34.179 Fl_Window.H

Go to the documentation of this file.

```cpp
1 //
2 // Window header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //   https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //   https://www.fltk.org/bugs.php
15 //
16 ifndef Fl_Window_H
17 #define Fl_Window_H
18
19 #include <FL/Fl.H>
20 #include <FL/Fl_Group.H>
21 #include <FL/Fl_Bitmap.H>
22
23 #define FL_WINDOW 0xF0
24 #define FL_DOUBLE_WINDOW 0xF1
25
26 class Fl_X;
27 class Fl_Window_Driver;
28 class Fl_RGB_Image;
29 class Fl_Double_Window;
30
31 class Fl_EXPORT Fl_Window : public Fl_Group {  // TODO: it would make sense to merge the use of Fl_X and Fl_Window_Driver, maybe simply by
32 private:
33     static char *default_xclass_;
34     static char show_iconic_; // 1 means create next window in iconic form
35     int no_fullscreen_x;
36     int no_fullscreen_y;
37     int no_fullscreen_w;
38     int no_fullscreen_h;
39     int fullscreen_screen_top;
40     int fullscreen_screen_bottom;
41     int fullscreen_screen_left;
42     int fullscreen_screen_right;
43 }

Generated by Doxygen
const char * iconlabel_;  // TODO: deriving Fl_Window_Driver from Fl_X. However, there are a lot of historic kludges
Fl_X *flx_; // points at the system-specific stuff, but exists only after the window is mapped
Fl_Window_Driver *pWindowDriver; // points at the system-specific stuff at window creation time

// private size_range stuff:
int minw_, minh_, maxw_, maxh;
uchar size_range_set_; // true (1) if size_range() has been set or calculated

// cursor stuff
Fl_Cursor cursor_default;
void _Fl_Window(); // constructor innards

// unimplemented copy ctor and assignment operator
Fl_Window(const Fl_Window&);
Fl_Window& operator=(const Fl_Window&);

protected:

static Fl_Window *current_;  // TODO: for some platforms around Fl_X.
virtual void draw();
void draw() FL_OVERRIDE;

virtual void flush();

void force_position(int force) {
  if (force) set_flag(FORCE_POSITION);
  else clear_flag(FORCE_POSITION);
}

int force_position()const { return ((flags() & FORCE_POSITION)?1:0); }

void free_icons();

void default_size_range(); // calculate size_range() if not set explicitly
int is_resizable(); // calculate size_range() and return whether this is resizable

public:

Fl_Window(int w, int h, const char *title = 0);
Fl_Window(int x, int y, int w, int h, const char *title = 0);
Fl_Window(int x, int y, int w, int h, const char *title = 0);
virtual ~Fl_Window();

int handle(int) FL_OVERRIDE;

void resize(int X,int Y,int W,int H) FL_OVERRIDE;
void resize(int X, int Y, int W, int H) FL_OVERRIDE;

void border(int b);
void border(int b);
void clear_border() {set_flag(NOBORDER);}
unsigned int border()const {return !(flags() & NOBORDER);}  // TODO: for some platforms around Fl_X.

void set_override() {set_flag(NOBORDER|OVERRIDE);}  // TODO: for some platforms around Fl_X.
unsigned int override() const { return flags()&OVERRIDE; }

void set_menu_window() {set_flag(MENU_WINDOW);  // TODO: for some platforms around Fl_X.
unsigned int menu_window()const {return flags() & MENU_WINDOW; }}  // TODO: for some platforms around Fl_X.

void hotspot(int x, int y, int offscreen = 0);
void hotspot(const Fl_Widget *, int offscreen = 0);
void hotspot(const Fl_Widget& p, int offscreen = 0) {hotspot(&p,offscreen);}

void free_position() {clear_flag(FORCE_POSITION);}

void size_range(int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0);
const char * label()const {return Fl_Widget::label();}
const char * iconlabel()const {return iconlabel_;

void label(const char * label, const char* iconlabel); // platform dependent

static void default_xclass(const char *);
static const char *default_xclass();
const char * xclass() const;
void xclass(const char * c);
365  static void default_icon(const Fl_RGB_Image *);
366  static void default_icons(const Fl_RGB_Image [], int);
367  void icon(const Fl_RGB_Image *);
368  void icons(const Fl_RGB_Image [], int);
370
371  #if defined(_WIN32) || defined(FL_DOXYGEN)
372  typedef struct HICON__ * HICON;
374  static void default_icon(HICON big_icon, HICON small_icon);
376  #endif // defined(_WIN32) || defined(FL_DOXYGEN)
377
378  / * for legacy compatibility */
379  const void * icon() const;
380  void icon(const void * ic);
381
389  int shown() {return flx_ != 0;}
390  void show() FL_OVERRIDE;
391  void hide() FL_OVERRIDE;
392
395  static void default_icons(HICON big_icon, HICON small_icon);
396
404  void icons(HICON big_icon, HICON small_icon);
406  #endif // defined(_WIN32) || defined(FL_DOXYGEN)
407
408  / * for legacy compatibility */
409  const void * icon() const;
410  void icon(const void * ic);
411
394  int x_root() const;
395  int y_root() const;
396
417  void wait_forExpose();
398  void fullscreen();
399  void fullscreen_off();
400  void fullscreen_off(int X, int Y, int W, int H);
401  unsigned int fullscreen_active() const { return flags() & FULLSCREEN; }
402  void fullscreen_screens(int top, int bottom, int left, int right);
403  void iconize();
404
391  static void default_callback(Fl_Window *, void* v);
405
417  void fullscreen();
418  void fullscreen_off();
419  void fullscreen_off(int X, int Y, int W, int H);
420  unsigned int fullscreen_active() const { return flags() & FULLSCREEN; }
421  void fullscreen_screens(int top, int bottom, int left, int right);
422  void iconize();
423
428  #endif
430
431  static Fl_Window *current();
432  void make_current();
433
442  void cursor(Fl_Cursor);
443  void cursor(const Fl_RGB_Image *, int, int);
444  void default_cursor(Fl_Cursor);
445
452  / * for legacy compatibility */
453  void cursor(Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE);
454  void default_cursor(Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE);
455
461  static void default_callback(Fl_Window *, void* v);
462
468  int decorated_w() const;
469
478  virtual class Fl_Double_Window *as_double_window() {return 0L;}
479
484  virtual class Fl_Overlay_Window *as_overlay_window() {return 0L;}
485
492  virtual class Fl_Double_Window *as_double_window() {return 0L;}
493
499  int screen_num();
500  void screen_num(int screen_num);
501  static bool is_a_rescale();
502  fl_uintptr_t os_id();
503
508  #endif
509
34.180 Fl_Wizard.H
1 //
2 // Fl_Wizard widget definitions.
3 //
4 // Copyright 1999-2010 by Easy Software Products.
5 // Copyright 2011-2020 by Bill Spitzak and others.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // file COPYING which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
34.181 Fl_XBM_Image.H

10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:
14 //
15 // https://www.fltk.org/bugs.php
16 //
17 18 /* \file
19 Fl_Wizard widget. */
20
21 // Include necessary header files...
22 //
23 #ifndef _Fl_Wizard_H_
24 #define _Fl_Wizard_H_
25
26 #include <FL/Fl_Group.H>
27
28 class FL_EXPORT Fl_Wizard : public Fl_Group {
29
30 Fl_Widget *value_
31
32 protected:
33
34 void draw() FL_OVERRIDE;
35
36 public:
37
38 Fl_Wizard(int, int, int, int, const char * = 0);
39
40 void next();
41 void prev();
42 Fl_Widget *value();
43 void value(Fl_Widget *);
44
45 #endif // !_Fl_Wizard_H_

34.181 Fl_XBM_Image.H

1 //
2 // XBM image header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 17 /* \file
18 Fl_XBM/Image class. */
19
20 ifndef Fl_XBM_Image_H
21 define Fl_XBM_Image_H
22 # include "Fl_Bitmap.H"
23
24 class FL_EXPORT Fl_XBM_Image : public Fl_Bitmap {
25
26 public:
27
28 Fl_XBM/Image(const char* filename);
29
30 #endif // Fl_XBM/Image_H

34.182 Fl_XPM_Image.H

1 //
2 // XPM image header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
Fl_XPM_Image class . */

#ifndef Fl_XPM_Image_H
#define Fl_XPM_Image_H
#include "Fl_Pixmap.H"

class FL_EXPORT Fl_XPM_Image : public Fl_Pixmap {

public:

Fl_XPM_Image(const char * filename);

};

#endif // !Fl_XPM_Image
#define FL_BOTTOM_BCOL FL_DARK2 // 40
#define FL_RIGHT_BCOL FL_DARK3 // 36
#define FL_INACTIVE FL_INACTIVE_COLOR
#define FL_INACTIVE_COL FL_INACTIVE_COLOR
#define FL_FREE_COL1 FL_FREE_COLOR
#define FL_FREE_COL2 ((Fl_Color)(FL_FREE_COLOR+1))
#define FL_FREE_COL3 ((Fl_Color)(FL_FREE_COLOR+2))
#define FL_FREE_COL4 ((Fl_Color)(FL_FREE_COLOR+3))
#define FL_FREE_COL5 ((Fl_Color)(FL_FREE_COLOR+4))
#define FL_FREE_COL6 ((Fl_Color)(FL_FREE_COLOR+5))
#define FL_FREE_COL7 ((Fl_Color)(FL_FREE_COLOR+6))
#define FL_FREE_COL8 ((Fl_Color)(FL_FREE_COLOR+7))
#define FL_FREE_COL9 ((Fl_Color)(FL_FREE_COLOR+8))
#define FL_FREE_COL10 ((Fl_Color)(FL_FREE_COLOR+9))
#define FL_FREE_COL11 ((Fl_Color)(FL_FREE_COLOR+10))
#define FL_FREE_COL12 ((Fl_Color)(FL_FREE_COLOR+11))
#define FL_FREE_COL13 ((Fl_Color)(FL_FREE_COLOR+12))
#define FL_FREE_COL14 ((Fl_Color)(FL_FREE_COLOR+13))
#define FL_FREE_COL15 ((Fl_Color)(FL_FREE_COLOR+14))
#define FL_FREE_COL16 ((Fl_Color)(FL_FREE_COLOR+15))
#define FL_TOMATO ((Fl_Color)(131))
#define FL_INDIANRED ((Fl_Color)(164))
#define FL_SLATEBLUE ((Fl_Color)(195))
#define FL_DARKGOLD ((Fl_Color)(84))
#define FL_PALEGREEN ((Fl_Color)(157))
#define FL_ORCHID ((Fl_Color)(203))
#define FL_DARKCYAN ((Fl_Color)(189))
#define FL_DARKTOMATO ((Fl_Color)(113))
#define FL_WHEAT ((Fl_Color)(174))
#define FL_ALIGN_BESIDE FL_ALIGN_INSIDE
#define FL_PUP_TOGGLE 2 // FL_MENU_TOGGLE
#define FL_PUP_INACTIVE 1 // FL_MENU_INACTIVE
#define FL_NO_FRAME FL_NO_BOX
#define FL_ROUNDED3D_UPBOX FL_ROUND_UP_BOX
#define FL_ROUNDED3D_DOWNBOX FL_ROUND_DOWN_BOX
#define FL_OVAL3D_UPBOX FL_ROUND_UP_BOX
#define FL_OVAL3D_DOWNBOX FL_ROUND_DOWN_BOX
#define FL_MBUTTON1 1
#define FL_LEFTMOUSE 1
#define FL_MBUTTON2 2
#define FL_MBUTTON3 3
#define FL_RIGHTMOUSE 3
#define FL_MBUTTON4 4
#define FL_MBUTTON5 5
#define FL_INVALID_STYLE 255
#define FL_NORMAL_STYLE FL_HELVETICA
#define FL_BOLD_STYLE FL_HELVETICA_BOLD
#define FL_ITALIC_STYLE FL_HELVETICA_ITALIC
#define FL_BOLDITALIC_STYLE FL_HELVETICA_BOLD_ITALIC
#define FL_FIXED_STYLE FL_COURIER
#define FL_FIXEDBOLD_STYLE FL_COURIER_BOLD
#define FL_FIXEDITALIC_STYLE FL_COURIER_ITALIC
#define FL_FIXEDBOLDITALIC_STYLE FL_COURIER_BOLD_ITALIC
#define FL_TIMES_STYLE FL_TIMES
#define FL_TIMESBOLD_STYLE FL_TIMES_BOLD
#define FL_TIMESITALIC_STYLE FL_TIMES_ITALIC
#define FL_TIMESBOLDITALIC_STYLE FL_TIMES_BOLD_ITALIC
#define FL_SHADOW_STYLE (FL_SHADOW_LABEL«8)
#define FL_ENGRAVED_STYLE (FL_ENGRAVED_LABEL«8)
#define FL_EMBOSSED_STYLE (FL_EMBOSSED_LABEL«0)
#define FL_TINY_SIZE 8
#define FL_TINY_FONT FL_TINY
#define FL_LARGE_FONT FL_LARGE
#define FL_HUGE_FONT FL_HUGE
#define FL_MEDIUM_FONT FL_MEDIUM
#define FL_SMALL_FONT FL_SMALL
#define FL_NORMAL_FONT FL_NORMAL
#define FL_MEDIUM_FONT FL_MEDIUM
#define FL_LARGE_FONT FL_LARGE
#define FL_NORMAL_FONT FL_NORMAL
#define FL_NORMAL_FONT FL_NORMAL
#define FL_RETURN_END_CHANGED FL_WHEN_RELEASE
#define FL_RETURN_CHANGED FL_WHEN_CHANGED
#define FL_RETURN_END FL_WHEN_RELEASE_ALWAYS
#define FL_RETURN_ALWAYS (FL_WHEN_CHANGED | FL_WHEN_NOT_CHANGED)

#define FL_BOUND_WIDTH 3

typedef int FL_Coord;
typedef int FL_COLOR;

// fltk interaction:

#define FL_CMD_OPT void
extern FL_EXPORT void fl_initialize(int*, char**[], const char*, FL_CMD_OPT*, int);
inline void fl_finish() {}

typedef void (*FL_IO_CALLBACK) (FL_SOCKET, void*);
inline void fl_add_io_callback(int fd, short w, FL_IO_CALLBACK cb, void* v) {
    Fl::add_fd(fd, w, cb, v);
}
inline void fl_remove_io_callback(int fd, short, FL_IO_CALLBACK) {
    Fl::remove_fd(fd); // removes all the callbacks!
}

// type of callback is different and no "id" number is returned:
inline void fl_add_timeout(long msec, void (*cb)(void*), void* v) {
    Fl::add_timeout(msec *.001, cb, v);
}
inline void fl_remove_timeout(int) {}

// type of callback is different!
inline void fl_set_idle_callback(void (*cb)()) {Fl::set_idle(cb);}

FL_EXPORT Fl_Widget* fl_do_forms(void);
FL_EXPORT Fl_Widget* fl_check_forms();
inline Fl_Widget* fl_do_only_forms(void) {return fl_do_forms();}
inline Fl_Widget* fl_check_only_forms(void) {return fl_check_forms();}

// because of new redraw behavior, these are no-ops:
inline void fl_freeze_object(Fl_Widget*) {}
inline void fl_unfreeze_object(Fl_Widget*) {}
inline void fl_freeze_form(Fl_Window*) {}
inline void fl_unfreeze_form(Fl_Window*) {}
inline void fl_freeze_all_forms() {}
inline void fl_unfreeze_all_forms() {}
inline void fl_set_focus_object(Fl_Window*, Fl_Widget* o) {Fl::focus(o);}
inline void fl_reset_focus_object(Fl_Widget* o) {Fl::focus(o);}
#define fl_set_object_focus fl_set_focus_object

// void fl_set_form_atclose(Fl_Window*,int (*cb)(Fl_Window*,void*),void* v)
// void fl_set_atclose(int (*cb)(Fl_Window*,void*),void*)
// fl_set_form_atactivate/atdeactivate not implemented!

// Fl_Widget:

inline void fl_set_object_boxtype(Fl_Widget* o, Fl_Boxtype a) {o->box(a);}
inline void fl_set_object_lsize(Fl_Widget* o, int s) {o->labelsize(s);}

/* forms lib font indexes must be byte sized - extract correct byte from style word */
inline void fl_set_object_lstyle(Fl_Widget* o, int a) {
    Fl_Font font = Fl::font(o->check_forms());
    int aa = fl::labeltype(font);
    int a8 = aa & 0xff;
    o->labelfont((Fl_Font)(a & 0xff)); o->labeltype((Fl_Labeltype)(a >> 8));
}
inline void fl_set_object_lcolor(Fl_Widget* o, Fl_Color a) {o->labelcolor(a);}
define fl_set_object_lcolor fl_set_object_lcol

#define fl_set_object_lalign fl_set_object_align
inline void fl_set_object_lalign(Fl_Widget* o, Fl_Align a) {o->align(a);}
inline void fl_set_object_align(Fl_Widget* o, Fl_Widget* a) {o->align(a);}
inline void fl_set_object_label(Fl_Widget* o, const char* a) {o->label(a); o->redraw();}
inline void fl_set_object_position(Fl_Widget* o, int x, int y) {o->position(x,y);}
inline void fl_set_object_size(Fl_Widget* o, int w, int h) {o->size(w,h);}
inline void fl_set_object_position(Fl_Widget* o, int x, int y, int w, int h) {o->position(x,y,w,h);}

#define fl_set_object_geometry fl_set_object_size
inline void fl_set_object_geometry(Fl_Widget* o, int x, int y, int w, int h) {o->size(x,y,w,h);}

#define fl_set_object_lcolor fl_set_object_lcol
#define fl_set_object_lalign fl_set_object_lcol

inline void fl_set_object_color(Fl_Widget* o, Fl_Color a, Fl_Color b) {o->color(a,b);}
inline void fl_set_object_label(Fl_Widget* o, const char* a) {o->label(a); o->redraw();}
inline void fl_set_object_position(Fl_Widget* o, int x, int y) {o->position(x,y);}

// forms CB (Fl_Widget*, long)
inline void fl_set_object_callback(Fl_Widget* o, Forms_CB c, long a) {o->callback(c,a);}
#define fl_set_object_callback fl_set_object_scallback

#define fl_set_call_back fl_set_object_scallback
inline void fl_set_object_scallback(Fl_Widget* o) {o->do_callback();}
inline void fl_set_object_return(Fl_Widget* o, int v) {
    o->when(Fl::When(v) | FL_WHEN_RELEASE);
}
inline void fl_redraw_object(Fl_Widget* o) {o->redraw();}
inline void fl_show_object(Fl_Widget* o) {o->show();}
inline void fl_hide_object(Fl_Widget* o) {o->hide();}
inline void fl_free_object(Fl_Widget* o) { delete o;}
inline void fl_delete_object(Fl_Widget* o) {o->parent()->remove(o);}
inline void fl_activate_object(Fl_Widget* o) {o->activate();}
inline void fl_deactivate_object(Fl_Widget* o) {o->deactivate();}
inline void fl_add_object(Fl_Window* f, Fl_Widget* x) {f->add(x);}
inline void fl_insert_object(Fl_Widget* o, Fl_Widget* b) {b->parent()->insert(*o,b);}

inline Fl_Window* FL_ObjWin(Fl_Widget* o) {return o->window();}

// things that appeared in the demos a lot that I don’t emulate, but
// I did not want to edit out of all the demos...
inline int fl_get_border_width() {return 3;}
inline void fl_set_border_width(int) {}
inline void fl_set_object_dblbuffer(Fl_Widget*, int) {}
inline void fl_set_form_dblbuffer(Fl_Window*, int) {}

// Fl_Window:
inline Fl_Window* fl_bgn_form(Fl_Boxtype b, int w, int h) {
  Fl_Window * g = new Fl_Window(w,h,0);
  g->box(b);
  return g;
}

inline Fl_Window* fl_bgn_group() {return new Fl_Group(0,0,0,0,0);}
inline void fl_end_group() {Fl_Group::current()->forms_end();}

#define resizebox _ddfdesign_kludge()

inline void fl_scale_form(Fl_Window* f, double x, double y) {
  f->resizable(f); f->size((int)f->w() * x, int(f->h() * y));
}

inline void fl_set_form_position(Fl_Window* f, int x, int y) {
  f->position(x,y);
}

inline void fl_set_form_size(Fl_Window* f, int w, int h) {
  f->size(w,h);
}

#define fl_set_initial_placement fl_set_form_geometry
inline void fl_adjust_form_size(Fl_Window*) {}

FL_EXPORT void fl_show_form(Fl_Window* f, int p, int b, const char* n);

enum { // “p” argument values:
  FL_PLACE_FREE = 0, // make resizable
  FL_PLACE_MOUSE = 1, // mouse centered on form
  FL_PLACE_CENTER = 2, // center of the screen
  FL_PLACE_FIXED = 4, // fixed position, resizable
  FL_PLACE_SIZE = 8, // fixed size, normal fltk behavior
  FL_PLACE_GEOMETRY = 16, // fixed size and position
  FL_PLACE_ASPECT = 32, // keep aspect ratio (ignored)
  FL_PLACE_FULLSCREEN = 64, // fill screen
  FL_PLACE_HOTSPOT = 128, // enables hotspot
  FL_PLACE_ICONIC = 256, // iconic (ignored)
  FL_FREE_SIZE = (1«14), // force resizable
  FL_FIX_SIZE = (1«15) // force off resizable
};
#define FL_PLACE_FREE_CENTER (FL_PLACE_CENTER|FL_FREE_SIZE)

#define FL_PLACE_CENTERFREE (FL_PLACE_CENTER|FL_FREE_SIZE)

enum { // “b” argument values:
  FL_NOBORDER = 0,
  FL_FULLSCREEN,
  FL_TRANSIENT
};

FL_EXPORT void fl_show_form_window(Fl_Window*) {}
inline void fl_set_form_callback(Fl_Window* f, Forms_FormCB c) {f->callback(c);}  
#define fl_set_form_callback fl_set_form_callback

inline void fl_init() {}

FL_EXPORT void fl_set_graphics_mode(int,int);

inline int fl_form_is_visible(Fl_Window* f) {return f->visible();}
inline int fl_mouse_button() {return Fl::event_button();}
#define fl_mousebutton fl_mouse_button

#define fl_free free
#define fl_malloc malloc
#define fl_calloc calloc
#define fl_realloc realloc

// Drawing functions. Only usable inside an Fl_Free object?
inline void fl_drw_box(Fl_Boxtype b,int x,int y,int w,int h, Fl_Color bgc,int=3) {
  fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_frame(Fl_Boxtype b,int x,int y,int w,int h, Fl_Color bgc,int=3) {
  fl_draw_box(b,x,y,w,h,bgc);
}
inline void fl_drw_text(Fl_Align align, int x, int y, int w, int h,
  Fl_Color fgcolor, int size, Fl_Font style,
  const char * s) {
  fl_font(style,size);
  fl_color(fgcolor);
  fl_draw(s,x,y,w,h,align);
}
inline void fl_drw_text_beside(Fl_Align align, int x, int y, int w, int h,
  Fl_Color fgcolor, int size, Fl_Font style,
  const char * s) {
  fl_font(style,size);
  fl_color(fgcolor);
  fl_draw(s,x,y,w,h,align);
}

inline void fl_set_font_name(Fl_Font n,const char* s) {Fl::set_font(n,s);}

inline void fl_mapcolor(Fl_Color c, uchar r, uchar g, uchar b) {Fl::set_color(c,r,g,b);}

#define fl_set_clipping(x,y,w,h) fl_push_clip(x,y,w,h)
#define fl_unset_clipping() fl_pop_clip()

// Forms classes:
inline Fl_Widget* fl_add_new(Fl_Widget* p) {return p;}
inline Fl_Widget* fl_add_new(uchar t,Fl_Widget* p) {p->type(t); return p;}

#define forms_constructor(type,name)

inline type* name(uchar t,int x,int y,int w,int h,const char* l) {
  return (type*)(fl_add_new(t, new type(x,y,w,h,l)));
}
#define forms_constructort(type,name)

inline type* name(uchar t,int x,int y,int w,int h) {
  return (type*)(fl_add_new(new type(t,x,y,w,h,l)));
}
#define forms_constructorb(type,name)

inline type* name(Fl_Boxtype t,int x,int y,int w,int h,const char* l) {
  return (type*)(fl_add_new(new type(t,x,y,w,h,l)));
}

#include "Fl_FormsBitmap.H"
#define FL_NORMAL_BITMAP FL_NO_BOX

inline void fl_set_bitmap_data(Fl_Widget* o, int w, int h, const uchar* b) {
  ((Fl_FormsBitmap *)o)->set(w,h,b);
}

#include "Fl_FormsPixmap.H"
#define FL_NORMAL_PIXMAP FL_NO_BOX

inline void fl_set_pixmap_data(Fl_Widget* o, char*const* b) {
  ((Fl_FormsPixmap *)o)->set(b);
}

#include "Fl_Box.H"
#include "Fl_Browser.H"

inline void fl_clear_browser(Fl_Widget* o) {
  ((Fl_Browser*)o)->clear();
}
inline void fl_add_browser_line(Fl_Widget* o, const char* s) {
  ((Fl_Browser*)o)->add(s);
}

inline void fl_addto_browser(Fl_Widget* o, const char* s) {
  ((Fl_Browser*)o)->add(s);
} /* should also scroll to bottom */

//inline void fl_addto_browser_chars(Fl_Widget*, const char*)
#define fl_append_browser fl_addto_browser_chars

inline void fl_insert_browser_line(Fl_Widget* o, int n, const char* s) {
  ((Fl_Browser*)o)->insert(n,s);
}

inline void fl_delete_browser_line(Fl_Widget* o, int n) {
  ((Fl_Browser*)o)->remove(n);
}

inline char* fl_get_browser_line(Fl_Widget* o, int n) {
  return ((Fl_Browser*)o)->text(n);
}

inline int fl_load_browser(Fl_Widget* o, const char* f) {
  return ((Fl_Browser*)o)->load(f);
}

inline void fl_select_browser_line(Fl_Widget* o, int n) {
  ((Fl_Browser*)o)->select(n,1);
}

inline void fl_deselect_browser_line(Fl_Widget* o, int n) {
  ((Fl_Browser*)o)->select(n,0);
}

inline void fl_deselect_browser(Fl_Widget* o) {
  ((Fl_Browser*)o)->deselect();
}

inline int fl_isselected_browser_line(Fl_Widget* o, int n) {
  return ((Fl_Browser*)o)->selected(n);
}

inline int fl_get_browser_topline(Fl_Widget* o, int n) {
  return ((Fl_Browser*)o)->topline(n);
}

inline int fl_get_browser_maxline(Fl_Widget* o) {
  return ((Fl_Browser*)o)->size();
}

inline int fl_get_browser_dimension(Fl_Widget* o, int n, int i) {
  return ((Fl_Browser*)o)->displayed(n,i);
}

# include "Fl_Button.H"

#define FL_NORMAL_BUTTON 0
#define FL_TOUCH_BUTTON 4
#define FL_INOUT_BUTTON 5
#define FL_RETURN_BUTTON 6
#define FL_HIDDEN_RET_BUTTON 7
#define FL_PUSH_BUTTON FL_TOGGLE_BUTTON
#define FL_MENU_BUTTON 9

inline Fl_Button* fl_add_button(uchar t, int x, int y, int w, int h, const char* l) {
  return (new Fl_Button(t, x, y, w, h, l))->value();
}

inline void fl_set_button_data(Fl_Widget* o, int a, int b, uchar* c) {
  (new Fl_Bitmap(c, a, b))->label(o); // does not delete old Fl_Bitmap!
}

inline Fl_Button* fl_add_bitmapbutton(int t, int x, int y, int w, int h, const char* l) {
  return (new Fl_Button(t, x, y, w, h, l))->value();
}

inline Fl_Button* fl_add_pixmapbutton(int t, int x, int y, int w, int h, const char* l) {
  return (new Fl_Button(t, x, y, w, h, l))->value();
}

#include "Fl_Light_Button.H"

forms_constructor(Fl_Light_Button, fl_add_lightbutton)

#include "Fl_Round_Button.H"

forms_constructor(Fl_Round_Button, fl_add_roundbutton)
forms_constructor(Fl_Round_Button, fl_add_round3dbutton)

#include "Fl_Check_Button.H"

forms_constructor(Fl_Check_Button, fl_add_checkbutton)

inline Fl_Widget* fl_add_bitmapbutton(int t, int x, int y, int w, int h, const char* l) {
  return (new Fl_Button(t, x, y, w, h, l))->value();
}

inline Fl_Widget* fl_add_bitmapbutton_data(int t, char* data) {
  return (new Fl_Button(t, x, y, w, h, l))->value();
}

Generated by Doxygen
inline void fl_add_button(Fl_Widget *o, const char *const *c) {
    return o;
}

inline void fl_set_pixmapbutton_data(Fl_Widget *o, const char *const *c) {
    (new Fl_Pixmap(c))->label(o); // does not delete old Fl_Pixmap!
}

#include "Fl_Chart.H"

forms_constructor(Fl_Chart, fl_add_chart)

inline void fl_clear_chart(Fl_Widget *o) {
    ((Fl_Chart *)o)->clear();
}
inline void fl_add_chart_value(Fl_Widget *o, double v, const char *s, uchar c) {
    ((Fl_Chart *)o)->add(v, s, c);
}
inline void fl_insert_chart_value(Fl_Widget *o, int i, double v, const char *s, uchar c) {
    ((Fl_Chart *)o)->insert(i, v, s, c);
}
inline void fl_replace_chart_value(Fl_Widget *o, int i, double v, const char *s, uchar c) {
    ((Fl_Chart *)o)->replace(i, v, s, c);
}
inline void fl_set_chart_bounds(Fl_Widget *o, double a, double b) {
    ((Fl_Chart *)o)->bounds(a, b);
}
inline void fl_set_chart_maxnumb(Fl_Widget *o, int v) {
    ((Fl_Chart *)o)->maxsize(v);
}
inline void fl_set_chart_autosize(Fl_Widget *o, int v) {
    ((Fl_Chart *)o)->autosize(v);
}
inline void fl_set_chart_lstyle(Fl_Widget *o, Fl_Font v) {
    ((Fl_Chart *)o)->textfont(v);
}
inline void fl_set_chart_lsize(Fl_Widget *o, int v) {
    ((Fl_Chart *)o)->textsize(v);
}
inline void fl_set_chart_lcolor(Fl_Widget *o, Fl_Color v) {
    ((Fl_Chart *)o)->textcolor(v);
}
#define fl_set_chart_lcol fl_set_chart_lcolor

#include "Fl_Choice.H"

#define FL_NORMAL_CHOICE 0
#define FL_NORMAL_CHOICE2 0
#define FL_DROPLIST_CHOICE 0

forms_constructor(Fl_Choice, fl_add_choice)

inline void fl_clear_choice(Fl_Widget *o) {
    ((Fl_Choice *)o)->clear();
}
inline void fl_addto_choice(Fl_Widget *o, const char *s) {
    ((Fl_Choice *)o)->add(s);
}
inline void fl_replace_choice(Fl_Widget *o, int i, const char *s) {
    ((Fl_Choice *)o)->replace(i-1, s);
}
inline void fl_delete_choice(Fl_Widget *o, int i) {
    ((Fl_Choice *)o)->remove(i-1);
}
inline void fl_set_choice(Fl_Widget *o, int i) {
    ((Fl_Choice *)o)->value(i-1);
}
inline int fl_get_choice(Fl_Widget *o) {
    return ((Fl_Choice *)o)->value()+1;
}
inline const char* fl_get_choice_item_text(Fl_Widget *, int);
inline int fl_get_choice_maxitems(Fl_Widget *);
inline void fl_set_choice_fontsize(Fl_Widget *o, int x) {
    ((Fl_Choice *)o)->textsize(x);
}
inline void fl_set_choice_fontstyle(Fl_Widget *o, Fl_Font x) {
    ((Fl_Choice *)o)->textfont(x);
}
#include "Fl_Clock.H"

forms_constructor(Fl_Clock, fl_add_clock)

inline void fl_get_clock(Fl_Widget *o, int *h, int *m, int *s) {
    *h = ((Fl_Clock *)o)->hour();
    *m = ((Fl_Clock *)o)->minute();
    *s = ((Fl_Clock *)o)->second();
}
#include "Fl_Counter.H"

forms_constructor(Fl_Counter, fl_add_counter)

inline void fl_set_counter_value(Fl_Widget *o, double v) {
    ((Fl_Counter *)o)->value(v);
}
inline void fl_set_counter_bounds(Fl_Widget *o, double a, double b) {
    ((Fl_Counter *)o)->bounds(a, b);
}
inline void fl_set_counter_step(Fl_Widget *o, double a, double b) {
    ((Fl_Counter *)o)->step(a, b);
}
inline void fl_set_counter_precision(Fl_Widget *o, int v) {
    ((Fl_Counter *)o)->precision(v);
}
inline void fl_set_counter_return(Fl_Widget *o, int v) {
    ((Fl_Counter *)o)->when((Fl_When)(v|FL_WHEN_RELEASE));
}
inline double fl_get_counter_value(Fl_Widget *o) {
    return ((Fl_Counter *)o)->value();
}
inline float fl_get_counter_minimum(Fl_Widget *o);
//inline void fl_set_counter_filter(Fl_Widget*, const char* (*)(Fl_Widget*, double, int));
// Cursor stuff cannot be emulated because it uses X stuff
inline void fl_set_counter_filter(Fl_Widget* w, Fl_Cursor c) {w->cursor(c);}
#define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
#define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT

#include "Fl_Dial.H"

#define FL_DIAL_COL1 FL_GRAY
#define FL_DIAL_COL2 37

forms_constructor(Fl_Dial, fl_add_dial)
inline void fl_set_dial_value(Fl_Widget* o, double v) {
  ((Fl_Dial *)o)->value(v);
}
inline double fl_get_dial_value(Fl_Widget* o) {
  return ((Fl_Dial *)o)->value();
}
inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
  ((Fl_Dial *)o)->bounds(a, b);
}
inline void fl_get_dial_bounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Dial*)o)->minimum());
  *b = float(((Fl_Dial*)o)->maximum());
}

#define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
#define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT

#include "Fl_Free.H"
inline Fl_Free*
fl_add_free(int t, double x, double y, double w, double h, const char* l, FL_HANDLEPTR hdl) {
  return (Fl_Free *)(fl_add_new(new Fl_Free(t, int(x), int(y), int(w), int(h), l, hdl)));
}

#include "fl_ask.H"
#include "fl_show_colormap.H"
inline int fl_show_question(const char* c, int = 0) {return fl_choice("%s", fl_no, fl_yes, 0L, c);}
FL_EXPORT void fl_show_message(const char *, const char *, const char *);
inline int fl_show_choices(const char* c, int n, const char* b1, const char* b2, const char * b3, int) {
  return fl_show_choice(0, c, 0, n, b1, b2, b3);
}
#include "filename.H"
#include "Fl_File_Chooser.H"
inline int do_matching(char* a, const char* b) {return fl_filename_match(a, b);}
FL_EXPORT char* fl_show_file_selector(const char* message, const char* dir, char* deflt = 0);
FL_EXPORT char* fl_get_directory();
# FL_EXPORT char* fl_get_pattern();
# FL_EXPORT char* fl_get_filename();

#include "Fl_Input.H"
forms_constructor(Fl_Input, fl_add_input)
inline void fl_set_input(Fl_Widget* o, const char* v) {
    ((Fl_Input *)o)->value(v);
}
inline void fl_set_input_return(Fl_Widget* o, int x) {
    ((Fl_Input *)o)->when((Fl_When)(x | FL_WHEN_RELEASE));
}
inline void fl_set_input_color(Fl_Widget* o, Fl_Color a, Fl_Color b) {
    ((Fl_Input *)o)->textcolor(a);
    ((Fl_Input *)o)->cursor_color(b);
}
inline void fl_set_input_scroll(Fl_Widget*, int);
inline void fl_set_input_cursorpos(Fl_Widget* o, int x, int /*y*/);
inline void fl_set_input_selected(Fl_Widget*, int);
inline void fl_set_input_selected_range(Fl_Widget*, int, int);
inline void fl_set_input_vscrollbarsize(Fl_Widget*, int, int);
inline void fl_set_input_topline(Fl_Widget*, int);
inline void fl_set_input_offset(Fl_Widget*, int);
inline void fl_set_input_maxchars(Fl_Widget*, int);
inline void fl_set_input_format(Fl_Widget*, int, int);
inline void fl_set_input_hscrollbar(Fl_Widget*, int);
inline void fl_set_input_vscrollbar(Fl_Widget*, int);
inline int fl_get_input_topline(Fl_Widget*);
inline int fl_get_input_screenlines(Fl_Widget*);
inline int fl_get_input_cursorpos(Fl_Widget*, int*, int*);
inline const char* fl_get_input(Fl_Widget* o) {return ((Fl_Input*)o)->value();}

#include "Fl_Menu_Button.H"
types are not implemented, they all act like FL_PUSH_MENU:
define FL_TOUCH_MENU 0
define FL_PUSH_MENU 1
define FL_PULLDOWN_MENU 2
forms_constructor(Fl_Menu_Button, fl_add_menu)
inline void fl_clear_menu(Fl_Widget* o) {
    ((Fl_Menu_Button *)o)->clear();
}
inline void fl_set_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button *)o)->clear(); ((Fl_Menu_Button*)o)->add(s);
}
inline void fl_addto_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button *)o)->add(s);
}
inline void fl_replace_menu_item(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button *)o)->replace(i-1,s);
}
inline void fl_delete_menu_item(Fl_Widget* o, int i) {
    ((Fl_Menu_Button*)o)->remove(i-1);
}
inline void fl_set_menu_item_shortcut(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button*)o)->shortcut(i-1,fl_old_shortcut(s));
}
inline void fl_set_menu_item_mode(Fl_Widget* o, int i, long x) {
    ((Fl_Menu_Button*)o)->mode(i-1,(int)x);
}
inline const char* fl_get_menu_item_text(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button*)o)->text(i);
}
inline int fl_get_menu_maxitems(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->size();
}
inline int fl_get_menu_item_mode(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button*)o)->mode(i);
}
inline const char* fl_get_menu_text(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->text();
}

#include "Fl_Positioner.H"
define FL_NORMAL_POSITIONER 0
forms_constructor(Fl_Positioner, fl_add_positioner)
inline void fl_set_positioner_xvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner *)o)->xvalue(v);
}
inline void fl_set_positioner_xvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->xvalue();
}
inline void fl_set_positioner_xbounds(Fl_Widget* o, double a, double b) {
    ((Fl_Positioner*)o)->xbounds(a,b);
}
inline void fl_set_positioner_xbounds(Fl_Widget* o, float a, float b) {
    a = float(((Fl_Positioner*)o)->xminimum());
    b = float(((Fl_Positioner*)o)->xmaximum());
}
inline void fl_set_positioner_yvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->yvalue(v);
}
inline void fl_set_positioner_yvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->yvalue();
}
inline void fl_set_positioner_ybounds(Fl_Widget* o, double a, double b) {
    return ((Fl_Positioner*)o)->ybounds();
}
This file defines wrapper functions for OpenGL in FLTK.

```c
#include "Enumerations.H"
#include <GL/gl.h>
```

This file defines wrapper functions for OpenGL in FLTK.

```c
#include "Enumerations.H"
#include <GL/gl.h>
```
Functions

- **void gl_color (Fl_Color i)**
  
  Sets the current OpenGL color to an FLTK color.

- **void gl_color (int c)**
  
  back compatibility

- **int gl_descent ()**
  
  Returns the current font's descent.

- **void gl_draw (const char * )**
  
  Draws a nul-terminated string in the current font at the current position.

- **void gl_draw (const char *, float x, float y)**
  
  Draws a nul-terminated string in the current font at the given position.

- **void gl_draw (const char *, int n)**
  
  Draws an array of n characters of the string in the current font at the current position.

- **void gl_draw (const char *, int n, float x, float y)**
  
  Draws n characters of the string in the current font at the given position.

- **void gl_draw (const char *, int n, int x, int y)**
  
  Draws n characters of the string in the current font at the given position.

- **void gl_draw (const char *, int x, int y)**
  
  Draws a nul-terminated string in the current font at the given position.

- **void gl_draw (const char *, int x, int y, int w, int h, Fl_Align)**
  
  Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X.

- **void gl_draw_image (const uchar *, int x, int y, int w, int h, int d=3, int ld=0)**

- **void gl_finish ()**
  
  Releases an OpenGL context.

- **void gl_font (int fontid, int size)**
  
  Sets the current OpenGL font to the same font as calling fl_font().

- **int gl_height ()**
  
  Returns the current font's height.

- **void gl_measure (const char *, int &x, int &y)**
  
  Measure how wide and tall the string will be when drawn by the gl_draw() function.

- **void gl_rect (int x, int y, int w, int h)**
  
  Outlines the given rectangle with the current color.

- **void gl_rectf (int x, int y, int w, int h)**
  
  Fills the given rectangle with the current color.

- **void gl_start ()**
  
  Creates an OpenGL context.

- **int gl_texture_pile_height ()**
  
  Returns the current maximum height of the pile of pre-computed string textures.

- **void gl_texture_pile_height (int max)**
  
  Changes the maximum height of the pile of pre-computed string textures.

- **void gl_texture_reset ()**
  
  To call after GL operations that may invalidate textures used to draw text in GL scenes (e.g., switch between FL_← DOUBLE / FL_SINGLE modes).

- **double gl_width (const char *)**
  
  Returns the width of the string in the current font.

- **double gl_width (const char *, int n)**
  
  Returns the width of n characters of the string in the current font.

- **double gl_width (uchar)**
  
  Returns the width of the character in the current font.
34.184.1 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)

34.184.2 Function Documentation

34.184.2.1 gl_color()

void gl_color (Fl_Color i)
Sets the current OpenGL color to an FLTK color.
For color-index modes it will use fl_xpixel(c), which is only right if the window uses the default colormap!

34.184.2.2 gl_draw() [1/7]

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)

34.184.2.3 gl_draw() [2/7]

void gl_draw (const char * str, float x, float y )
Draws a nul-terminated string in the current font at the given position.
See also
   gl_texture_pile_height(int)

34.184.2.4 gl_draw() [3/7]

void gl_draw (const char * str, int n )
Draws an array of n characters of the string in the current font at the current position.
See also
   gl_texture_pile_height(int)
34.184.2.5  **gl_draw() [4/7]**

```c
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)`

34.184.2.6  **gl_draw() [5/7]**

```c
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)`

34.184.2.7  **gl_draw() [6/7]**

```c
void gl_draw (const char * str, int x, int y )
```

Draws a nul-terminated string in the current font at the given position.

See also

- `gl_texture_pile_height(int)`

34.184.2.8  **gl_draw() [7/7]**

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

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

34.184.2.9  **gl_font()**

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

Sets the current OpenGL font to the same font as calling `fl_font()`.
See also

Fl::draw_GL_text_with_textures(int val)

### 34.184.2.10 gl_rect()}

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

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

### 34.184.2.11 gl_rectf()}

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

Fills the given rectangle with the current color.

See also

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

### 34.184.2.12 gl_texture_pile_height() [1/2]

```c
int gl_texture_pile_height (  
    void )
```

Returns the current maximum height of the pile of pre-computed string textures.
The default value is 100

See also

```c
Fl::draw_GL_text_with_textures(int)
```

### 34.184.2.13 gl_texture_pile_height() [2/2]

```c
void gl_texture_pile_height (  
    int max )
```

Changes the maximum height of the pile of pre-computed string textures.
Strings that are often re-displayed can be processed much faster if this pile is set high enough to hold all of them.

Parameters

| max | Maximum height of the texture pile |

See also

```c
Fl::draw_GL_text_with_textures(int)
```

### 34.185 gl.h

Go to the documentation of this file.
// OpenGL header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2018 by Bill Spitzak and others.

// You must include this instead of GL/gl.h to get the Microsoft
// APIENTRY stuff included (from <windows.h>) prior to the OpenGL
// header files.

// This file also provides "missing" OpenGL functions, and
// gl_start() and gl_finish() to allow OpenGL to be used in any window

// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php

// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef FL_gl_H
#define FL_gl_H

#include "Enumerations.H" // for color names
#ifdef _WIN32
#include <windows.h>
#endif
#ifndef APIENTRY
#ifdef __CYGWIN__
#define APIENTRY __attribute__ ((__stdcall__))
#else
#define APIENTRY
#endif
#endif

#ifdef __APPLE__ // PORTME: OpenGL path abstraction
#ifndef GL_SILENCE_DEPRECATION
#define GL_SILENCE_DEPRECATION 1
#endif
#ifndef __gl3_h__ // make sure OpenGL/gl3.h was not included before
#include <OpenGL/gl.h>
#endif
#else
#include <GL/gl.h>
#endif // __APPLE__ // PORTME: OpenGL Path abstraction

FL_EXPORT void gl_start();
FL_EXPORT void gl_finish();

FL_EXPORT void gl_color(Fl_Color i);
inline void gl_color(int c) {gl_color((Fl_Color)c);};

FL_EXPORT void gl_rect(int x,int y,int w,int h);
FL_EXPORT void gl_rectf(int x,int y,int w,int h);

FL_EXPORT void gl_font(int fontid, int size);
FL_EXPORT int gl_height();
FL_EXPORT int gl_descent();
FL_EXPORT double gl_width(const char *);
FL_EXPORT double gl_width(const char *, int n);
FL_EXPORT double gl_width(uchar);

FL_EXPORT void gl_draw_image(const uchar *, int x,int y,int w,int h, int d=3, int ld=0);

#endif // !FL_gl_H

34.186  gl2opengl.h

1 /*
2 gl.h
3 */
If you include this, you might be able to port old GL programs. There are also much better emulators available on the net.

```c
8 *
9 #ifndef _FL_gl2opengl_h_
10 #define _FL_gl2opengl_h_
11
12 #include <FL/gl.h>
13 #include "gl_draw.H"
14
15 inline void clear() {glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
16 #define RGBcolor(r,g,b) glColor3ub(r,g,b)
17 #define bgnline() glBegin(GL_LINE_STRIP)
18 #define bgnpolygon() glBegin(GL_POLYGON)
19 #define bgnclosedline() glBegin(GL_LINE_LOOP)
20 #define endline() glEnd()
21 #define endpolygon() glEnd()
22 #define endclosedline() glEnd()
23 #define v2f(v) glVertex2fv(v)
24 #define v2s(v) glVertex2sv(v)
25 #define cmov(x,y,z) glRasterPos3f(x,y,z)
26 #define charstr(s) gl_draw(s)
27 #define fmprstr(s) gl_draw(s)
28 typedef float Matrix[4][4];
29 inline void pushmatrix() {glPushMatrix();
30 inline void popmatrix() {glPopMatrix();
31 inline void multmatrix(Matrix m) {glMultMatrixf((float *)m);
32 inline void color(int n) {glIndexi(n);
33 inline void rectf(int x,int y,int r,int t) {gl_rectf(x,y,r-x,t-y)}
34 inline void rects(int x,int y,int r,int t) {gl_regs(x,y,r-x,t-y)}
35 inline void rectfs(int x,int y,int r,int t) {gl_rectf(x,y,r-x,t-y)}
36 inline void rectfi(int x,int y,int r,int t) {gl_rectf(x,y,r-x,t-y)}
37 inline void recti(int x,int y,int r,int t) {gl_rectf(x,y,r-x,t-y)}
38 inline void rectfs(int x,int y,int r,int t) {gl_regs(x,y,r-x,t-y)}
39 #endif /* _FL_gl2opengl_h_ */
```

---

### gl_draw.H

1 //
2 // OpenGL header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 //
17 #include "gl.h"
18
19 extern FL_EXPORT void gl_remove_displaylist_fonts();

---

### glu.h

1 //
2 // Glu header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // You must include this instead of GL/gl.h to get the Microsoft
7 // APIENTRY stuff included (from <windows.h>) prior to the OpenGL
8 // header files.
9 //
10 // This file also provides "missing" OpenGL functions, and
11 // gl_start() and gl_finish() to allow OpenGL to be used in any window
12 //
13 // This library is free software. Distribution and use rights are outlined in
14 // the file "COPYING" which should have been included with this file. If this
15 // file is missing or damaged, see the license at:
16 //
17 // https://www.fltk.org/COPYING.php
18 //
19 // Please see the following page on how to report bugs and issues:
ifndef FL_glu_H
#define FL_glu_H

#include "Enumerations.H" // for color names
#ifdef _WIN32
#include <windows.h>
#endif
#ifndef APIENTRY
#ifdef __CYGWIN__
#define APIENTRY __attribute__ ((__stdcall__))
#else
#define APIENTRY
#endif
#endif
#endif

#ifdef __APPLE__ // PORTME: OpenGL Path abstraction
#include <OpenGL/glu.h>
#else
#include <GL/glu.h>
#endif
#endif // !FL_glu_H

 glut.H

GLUT emulation header file for the Fast Light Tool Kit (FLTK).

Copyright 1998-2023 by Bill Spitzak and others.

This library is free software. Distribution and use rights are outlined in
the file "COPYING" which should have been included with this file. If this
file is missing or damaged, see the license at:

https://www.fltk.org/COPYING.php

Please see the following page on how to report bugs and issues:

https://www.fltk.org/bugs.php

Emulation of GLUT using fltk.

GLUT is Copyright (c) Mark J. Kilgard, 1994, 1995, 1996:
"This program is freely distributable without licensing fees and is
provided without guarantee or warranty expressed or implied. This
program is not in the public domain."

Although I have copied the GLUT API, none of my code is based on
any GLUT implementation details and is therefore covered by the LGPL.

Commented out lines indicate parts of GLUT that are not emulated.

ifndef _FL_glut_H_
define _FL_glut_H_
#include "gl.h"
#include "Fl.H"
#include "Fl_Gl_Window.H"

class FL_EXPORT Fl_Glut_Window : public Fl_Gl_Window {
    void _init();
    int mouse_down;
protected:
    void draw() FL_OVERRIDE;
    void draw_overlay() FL_OVERRIDE;
    int handle(int) FL_OVERRIDE;
public: // so the inline functions work
    int number;
    int menu[3];
    void make_current();
    void (*display)();
    void (*overlaydisplay)();
    void (*reshape)(int w, int h);
    void (*keyboard)(uchar, int x, int y);
    void (*mouse)(int b, int state, int x, int y);
    void (*motion)(int x, int y);
    void (*passivemotion)(int x, int y);
    void (*entry)(int);
}
61. void (*visibility)(int);
62. void (*special)(int, int x, int y);
63. Fl_Glut_Window(int w, int h, const char *t=0);
64. Fl_Glut_Window(int x, int y, int w, int h, const char *t=0);
65. ~Fl_Glut_Window();
66. }
67. 
68. extern FL_EXPORT Fl_Glut_Window *glut_window; // the current window
69. extern FL_EXPORT int glut_menu; // the current menu
70. 
71. // function pointers that are not per-window:
72. extern FL_EXPORT void (*glut_idle_function)();
73. extern FL_EXPORT void (*glut_menustate_function)(int);
74. extern FL_EXPORT void (*glut_menustatus_function)(int,int,int);
75. 
76. //# define GLUT_API_VERSION This does not match any version of GLUT exactly...
77. 
78. FL_EXPORT void glutInit(int *argcp, char **argv); // creates first window
79. 
80. FL_EXPORT void glutInitDisplayMode(unsigned int mode);
81. // the FL_ symbols have the same value as the GLUT ones:
82. # define GLUT_RGB FL_RGB
83. # define GLUT_RGBA FL_RGB
84. # define GLUT_INDEX FL_INDEX
85. # define GLUT_SINGLE FL_SINGLE
86. # define GLUT_DOUBLE FL_DOUBLE
87. # define GLUT_ACCUM FL_ACCUM
88. # define GLUT_ALPHA FL_ALPHA
89. # define GLUT_DEPTH FL_DEPTH
90. # define GLUT_STENCIL FL_STENCIL
91. # define GLUT_MULTISAMPLE FL_MULTISAMPLE
92. # define GLUT_STEREO FL_STENCIL
93. // define GLUT_LUMINANCE 512
94. 
95. FL_EXPORT void glutInitWindowPosition(int x, int y);
96. 
97. FL_EXPORT void glutInitWindowSize(int w, int h);
98. 
99. FL_EXPORT void glutMainLoop();
100. 
101. FL_EXPORT void glutCreateWindow(char *title);
102. FL_EXPORT void glutCreateWindow(const char *title);
103. 
104. FL_EXPORT int glutCreateSubWindow(int win, int x, int y, int width, int height);
105. 
106. FL_EXPORT void glutDestroyWindow(int win);
107. 
108. inline int glutGetWindow() {return glut_window->number;}
109. 
110. inline void glutPostRedisplay() {glut_window->redraw();}
111. 
112. FL_EXPORT void glutPostWindowRedisplay(int win);
113. 
114. FL_EXPORT void glutSwapBuffers();
115. 
116. inline int glutGetWindow() {return glut_window->number;}
117. 
118. FL_EXPORT void glutSetWindowTitle(char *t); (glut_window->label(t));
119. 
120. inline void glutSetWindow(char *t) {glut_window->label(t);}
121. 
122. inline void glutIconSetWindowTitle(char *t) {glut_window->iconlabel(t);}
123. 
124. inline void glutSetPositionWindow(int x, int y) {glut_window->position(x,y);}
125. 
126. inline void glutSetShapeWindow(int w, int h) {glut_window->size(w,h);}
127. 
128. inline void glutPopWindow() {glut_window->show();}
129. 
130. inline void glutPushWindow() { /* do nothing */}
131. 
132. inline void glutIconifyWindow() {glut_window->iconize();}
133. 
134. inline void glutShowWindow() {glut_window->show();}
135. 
136. inline void glutHideWindow() {glut_window->hide();}
137. 
138. inline void glutFullScreen() {glut_window->fullscreen();}
139. 
140. inline void glutSetCursor(Fl_Cursor cursor) {glut_window->cursor(cursor);}
141. // notice that the numeric values are different than glut:
142. # define GLUT_CURSOR_RIGHT_ARROW ((Fl_Cursor)2)
143. # define GLUT_CURSOR_LEFT_ARROW ((Fl_Cursor)67)
144. # define GLUT_CURSOR_INFO FL_CURSOR_INFO
145. # define GLUT_CURSOR_DESTROY FL_CURSOR_DESTROY
146. # define GLUT_CURSOR.AttributeSet FL_CURSOR_HHELP
147. # define GLUT_CURSOR_CYCLE ((Fl_Cursor)26)
148. # define GLUT_CURSOR_SPRAY ((Fl_Cursor)63)
# define GLUT_CURSOR_WAIT FL_CURSOR_WAIT
# define GLUT_CURSOR_TEXT FL_CURSOR_INSERT
# define GLUT_CURSOR_CROSSHAIR FL_CURSOR_CROSS
# define GLUT_CURSOR_UP_DOWN FL_CURSOR_NS
# define GLUT_CURSOR_LEFT_RIGHT FL_CURSOR_WE
# define GLUT_CURSOR_TOP_SIDE FL_CURSOR_N
# define GLUT_CURSOR_BOTTOM_SIDE FL_CURSOR_S
# define GLUT_CURSOR_LEFT_SIDE FL_CURSOR_W
# define GLUT_CURSOR_RIGHT_SIDE FL_CURSOR_E
# define GLUT_CURSOR_TOP_LEFT_CORNER FL_CURSOR_NW
# define GLUT_CURSOR_TOP_RIGHT_CORNER FL_CURSOR_NE
# define GLUT_CURSOR_BOTTOM_RIGHT_CORNER FL_CURSOR_SE
# define GLUT_CURSOR_BOTTOM_LEFT_CORNER FL_CURSOR_SW
# define GLUT_CURSOR_INHERIT FL_CURSOR_DEFAULT
# define GLUT_CURSOR_NONE FL_CURSOR_NONE
# define GLUT_CURSOR_FULL_CROSSHAIR FL_CURSOR_CROSS

inline void glutWarpPointer(int, int) { /* do nothing */ }

inline void glutEstablishOverlay() {glut_window->make_overlay_current();}

inline void glutRemoveOverlay() {glut_window->hide_overlay();}

inline void glutUseLayer(GLenum layer) {
    layer ? glut_window->make_overlay_current() : glut_window->make_current();
}

enum {GLUT_NORMAL, GLUT_OVERLAY};

inline void glutPostOverlayRedisplay() {glut_window->redraw_overlay();}

inline void glutShowOverlay() {glut_window->redraw_overlay();}

inline void glutHideOverlay() {glut_window->hide_overlay();}

FL_EXPORT int glutCreateMenu(void (*)(int));

FL_EXPORT void glutDestroyMenu(int menu);

inline int glutGetMenu() {return glut_menu;}

inline void glutSetMenu(int m) {glut_menu = m;}

FL_EXPORT void glutAddMenuEntry(const char *label, int value);

FL_EXPORT void glutAddSubMenu(char *label, int submenu);

FL_EXPORT void glutChangeToMenuEntry(int item, char *labela, int value);

FL_EXPORT void glutChangeToSubMenu(int item, char *label, int submenu);

FL_EXPORT void glutRemoveMenuItem(int item);

inline void glutAttachMenu(int b) {glut_window->menu[b] = glut_menu;}

inline void glutDetachMenu(int b) {glut_window->menu[b] = 0;}

inline void glutDisplayFunc(void (*f)()) {glut_window->display = f;}

inline void glutReshapeFunc(void (*f)(int w, int h)) {glut_window->reshape = f;}

inline void glutKeyboardFunc(void (*f)(uchar key, int x, int y)) {
    glut_window->keyboard = f;
}

inline void glutMouseFunc(void (*f)(int b, int state, int x, int y)) {
    glut_window->mouse = f;
}

#define GLUT_LEFT_BUTTON 0
#define GLUT_MIDDLE_BUTTON 1
#define GLUT_RIGHT_BUTTON 2
#define GLUT_DOWN 0
#define GLUT_UP 1

inline void glutMotionFunc(void (*f)(int x, int y)) {glut_window->motion = f;}

inline void glutPassiveMotionFunc(void (*f)(int x, int y)) {
    glut_window->passivemotion = f;
}

inline void glutEntryFunc(void (*f)(int s)) {glut_window->entry = f;}

enum {GLUT_LEFT, GLUT_ENTERED};

inline void glutVisibilityFunc(void (*f)(int s)) {glut_window->visibility = f;}

enum {GLUT_NOT_VISIBLE, GLUT_VISIBLE};

FL_EXPORT void glutIdleFunc(void (*f)());

inline void glutTimerFunc(unsigned int msec, void (*f)(int), int value) {
    Fl::add_timeout(msec *.001, (void (*)(void *))f, (void *)(fl_intptr_t)value);
}
inline void glutMenuStateFunc(void (*f)(int state)) {
    glut_menu_state_function = f;
}
inline void glutMenuStatusFunc(void (*f)(int status, int x, int y)) {
    glut_menu_status_function = f;
}
enum {GLUT_MENU_NOT_IN_USE, GLUT_MENU_IN_USE};
inline void glutSpecialFunc(void (*f)(int key, int x, int y)) {
    glut_window->special = f;
}
define GLUT_KEY_F1 1
#define GLUT_KEY_F2 2
#define GLUT_KEY_F3 3
#define GLUT_KEY_F4 4
#define GLUT_KEY_F5 5
#define GLUT_KEY_F6 6
#define GLUT_KEY_F7 7
#define GLUT_KEY_F8 8
#define GLUT_KEY_F9 9
#define GLUT_KEY_F10 10
#define GLUT_KEY_F11 11
#define GLUT_KEY_F12 12
#define GLUT_KEY_LEFT FL_Left
#define GLUT_KEY_UP FL_Up
#define GLUT_KEY_RIGHT FL_Right
#define GLUT_KEY_DOWN FL_Down
#define GLUT_KEY_PAGE_UP FL_Page_Up
#define GLUT_KEY_PAGE_DOWN FL_Page_Down
#define GLUT_KEY_HOME FL_Home
#define GLUT_KEY_END FL_End
#define GLUT_KEY_INSERT FL_Insert
//inline void glutSpaceballMotionFunc(void (*)(int x, int y, int z));
//inline void glutSpaceballRotateFunc(void (*)(int x, int y, int z));
//inline void glutSpaceballButtonFunc(void (*)(int button, int state));
//inline void glutButtonBoxFunc(void (*)(int button, int state));
//inline void glutDialsFunc(void (*)(int dial, int value));
//inline void glutTabletMotionFunc(void (*)(int x, int y));
//inline void glutTabletButtonFunc(void (*)(int button, int state, int x, int y));
inline void glutOverlayDisplayFunc(void (*f)()) {
    glut_window->overlaydisplay = f;
}
//inline void glutWindowStatusFunc(void (*)(int state));
//enum {GLUT_HIDDEN, GLUT_FULLY_RETAINED, GLUT_PARTIALLY_RETAINED,
//    GLUT_FULLY_COVERED};
//inline void glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);
//inline GLfloat glutGetColor(int ndx, int component);
#define GLUT_RED 0
#define GLUT_GREEN 1
#define GLUT_BLUE 2
//inline void glutCopyColormap(int win);
// Warning: values are changed from GLUT!
// Also relies on the GL_ symbols having values greater than 100
FL_EXPORT int glutGet(GLFW_type);
enum {
    GLUT_RETURN_ZERO = 0,
    GLUT_WINDOW_X,
    GLUT_WINDOW_Y,
    GLUT_WINDOW_WIDTH,
    GLUT_WINDOW_HEIGHT,
    GLUT_WINDOW_PARENT,
    GLUT_SCREEN_WIDTH,
    GLUT_SCREEN_HEIGHT,
    GLUT_MENU_NUM_ITEMS,
    GLUT_DISPLAY_MODE_POSSIBLE,
    GLUT_INIT_WINDOW_X,
    GLUT_INIT_WINDOW_Y,
    GLUT_INIT_WINDOW_WIDTH,
    GLUT_INIT_WINDOW_HEIGHT,
    GLUT_INIT_DISPLAY_MODE,
    GLUT_WINDOW_BUFFER_SIZE,
    GLUT_VERSION,
    GLUT_WINDOW_NUM_CHILDREN,
    GLUT_WINDOW_CURSOR,
    GLUT_SCREEN_WIDTH_MM,
    GLUT_SCREEN_HEIGHT_MM,
# define GLUT_WINDOW_STENCIL_SIZE GL_STENCIL_BITS
# define GLUT_WINDOW_DEPTH_SIZE GL_DEPTH_BITS
# define GLUT_WINDOW_RED_SIZE GL_RED_BITS
# define GLUT_WINDOW_GREEN_SIZE GL_GREEN_BITS
# define GLUT_WINDOW_BLUE_SIZE GL_BLUE_BITS
# define GLUT_WINDOW_ALPHA_SIZE GL_ALPHA_BITS
# define GLUT_WINDOW_ACCUM_RED_SIZE GL_ACCUM_RED_BITS
# define GLUT_WINDOW_ACCUM_GREEN_SIZE GL_ACCUM_GREEN_BITS
# define GLUT_WINDOW_ACCUM_BLUE_SIZE GL_ACCUM_BLUE_BITS
# define GLUT_WINDOW_ACCUM_ALPHA_SIZE GL_ACCUM_ALPHA_BITS
# define GLUT_WINDOW_DOUBLEBUFFER GL_DOUBLEBUFFER
# define GLUT_WINDOW_RGBA GL_RGBA
# define GLUT_WINDOW_COLORMAP_SIZE GL_INDEX_BITS
#ifdef GL_SAMPLES_SGIS
# define GLUT_WINDOW_NUM_SAMPLES GL_SAMPLES_SGIS
#else
# define GLUT_WINDOW_NUM_SAMPLES GLUT_RETURN_ZERO
#endif
# define GLUT_WINDOW_STEREO GL_STEREO

# define GLUT_HAS_KEYBOARD 600
# define GLUT_HAS_MOUSE 601
# define GLUT_HAS_SPACEBALL 602
# define GLUT_HAS_DIAL_AND_BUTTON_BOX 603
# define GLUT_HAS_TABLET 604
# define GLUT_NUM_MOUSE_BUTTONS 605
# define GLUT_NUM_SPACEBALL_BUTTONS 606
# define GLUT_NUM_BUTTON_BOX_BUTTONS 607
# define GLUT_NUM_DIALS 608
# define GLUT_NUM_TABLET_BUTTONS 609

FL_EXPORT int glutDeviceGet(GLenum type);

// WARNING: these values are different than GLUT uses:
# define GLUT_ACTIVE_SHIFT FL_SHIFT
# define GLUT_ACTIVE_CTRL FL_CTRL
# define GLUT_ACTIVE_ALT FL_ALT
inline int glutGetModifiers() {return Fl::event_state() & (GLUT_ACTIVE_SHIFT | GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT);}

FL_EXPORT int glutLayerGet(GLenum);

extern "C"
{
typedef void (*GLUTproc)();
}

FL_EXPORT GLUTproc glutGetProcAddress(const char *procName);

//inline int glutVideoResizeGet(GLenum param);
#define GLUT_VIDEO_RESIZE_POSSIBLE 900
#define GLUT_VIDEO_RESIZE_IN_USE 901
#define GLUT_VIDEO_RESIZE_WIDTH_DELTA 902
#define GLUT_VIDEO_RESIZE_HEIGHT_DELTA 903
#define GLUT_VIDEO_RESIZE_WIDTH 904
#define GLUT_VIDEO_RESIZE_HEIGHT 905
#define GLUT_VIDEO_RESIZE_X 906
#define GLUT_VIDEO_RESIZE_Y 907
#define GLUT_VIDEO_RESIZE_WIDTH 908
#define GLUT_VIDEO_RESIZE_HEIGHT 909

//inline void glutSetupVideoResizing();
//inline void glutStopVideoResizing();

// Font argument must be a void* for compatibility, so...
struct Fl_Glut_Bitmap_Font {Fl_Font font; Fl_Fontsize size;};
FL_EXPORT struct Fl_Glut_Bitmap_Font {Fl_Font font; Fl_Fontsize size;};

extern FL_EXPORT struct Fl_Glut_Bitmap_Font glutBitmap9By15, glutBitmap8By13, glutBitmapTimesRoman10,
glutBitmapTimesRoman24, glutBitmapHelvetica10, glutBitmapHelvetica12,
glutBitmapHelvetica18;
#define GLUT_BITMAP_9_BY_15 (&glutBitmap9By15)
#define GLUT_BITMAP_8_BY_13 (&glutBitmap8By13)
#define GLUT_BITMAP_TIMES_ROMAN_10 (&glutBitmapTimesRoman10)
#define GLUT_BITMAP_TIMES_ROMAN_24 (&glutBitmapTimesRoman24)
#define GLUT_BITMAP_9_BY_15 (&glutBitmap9By15)
#define GLUT_BITMAP_8_BY_13 (&glutBitmap8By13)
#define GLUT_BITMAP_TIMES_ROMAN_10 (&glutBitmapTimesRoman10)
#define GLUT_BITMAP_TIMES_ROMAN_24 (&glutBitmapTimesRoman24)
FL_EXPORT void glutBitmapCharacter(void *font, int character);
FL_EXPORT int glutBitmapHeight(void *font);
FL_EXPORT int glutBitmapLength(void *font, const unsigned char *string);
FL_EXPORT int glutBitmapWidth(void *font, int character);
FL_EXPORT int glutExtensionSupported(char *name);

/* GLUT stroked font sub-API */
struct Fl_Glut_StrokeVertex {
    GLfloat X, Y;
};

struct Fl_Glut_StrokeStrip {
    int Number;
    const Fl_Glut_StrokeVertex* Vertices;
};

struct Fl_Glut_StrokeChar {
    GLfloat Right;
    int Number;
    const Fl_Glut_StrokeStrip* Strips;
};

struct Fl_Glut_StrokeFont {
    char * Name; // The source font name
    int Quantity; // Number of chars in font
    GLfloat Height; // Height of the characters
    const Fl_Glut_StrokeChar** Characters; // The characters mapping
};

extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeRoman;
extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeMonoRoman;

#define GLUT_STROKE_ROMAN (&glutStrokeRoman)
#define GLUT_STROKE_MONO_ROMAN (&glutStrokeMonoRoman)

FL_EXPORT void glutStrokeCharacter(void *font, int character);
FL_EXPORT GLfloat glutStrokeHeight(void *font);
FL_EXPORT int glutStrokeLength(void *font, const unsigned char *string);
FL_EXPORT int glutStrokeWidth(void *font, int character);

/* GLUT pre-built models sub-API */
FL_EXPORT void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutWireCube(GLdouble size);
FL_EXPORT void glutSolidCube(GLdouble size);
FL_EXPORT void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutWireDodecahedron();
FL_EXPORT void glutSolidDodecahedron();
FL_EXPORT void glutWireTeapot(GLdouble size);
FL_EXPORT void glutSolidTeapot(GLdouble size);
FL_EXPORT void glutWireOctahedron();
FL_EXPORT void glutSolidOctahedron();
FL_EXPORT void glutWireTetrahedron();
FL_EXPORT void glutSolidTetrahedron();
FL EXPORT void glutWireIcosahedron();
FL_EXPORT void glutSolidIcosahedron();
- CGContextRef fl_mac_gc ()
  
  Returns the macOS-specific graphics context for the current window.

- void fl_mac_set_about (Fl_Callback ∗ cb, void ∗ user_data, int shortcut=0)
  
  Attaches a callback to the "About myprog" item of the system application menu.

- FLWindow ∗ fl_mac_xid (const Fl_Window ∗ win)
  
  Returns the macOS-specific window reference corresponding to the given Fl_Window object.

Variables

- int fl_mac_os_version
  The version number of the running Mac OS X (e.g., 10.6.4, 10.13).

34.190.1 Detailed Description

Mac OS X-specific symbols.

34.191 mac.H

Go to the documentation of this file.
```c
#ifndef MAC_OS_X_VERSION_10_6
#define MAC_OS_X_VERSION_10_6 1060
#endif
#ifndef MAC_OS_X_VERSION_10_7
#define MAC_OS_X_VERSION_10_7 1070
#endif
#ifndef MAC_OS_X_VERSION_10_8
#define MAC_OS_X_VERSION_10_8 1080
#endif
#ifndef MAC_OS_X_VERSION_10_9
#define MAC_OS_X_VERSION_10_9 1090
#endif
#ifndef MAC_OS_X_VERSION_10_10
#define MAC_OS_X_VERSION_10_10 101000
#endif
#ifndef MAC_OS_X_VERSION_10_11
#define MAC_OS_X_VERSION_10_11 101100
#endif
#ifndef MAC_OS_X_VERSION_10_12
#define MAC_OS_X_VERSION_10_12 101200
#endif
#ifndef MAC_OS_X_VERSION_10_13
#define MAC_OS_X_VERSION_10_13 101300
#endif
#ifndef MAC_OS_X_VERSION_10_14
#define MAC_OS_X_VERSION_10_14 101400
#endif
#ifndef MAC_OS_X_VERSION_10_15
#define MAC_OS_X_VERSION_10_15 101500
#endif
#ifndef MAC_OS_X_VERSION_10_16
#define MAC_OS_X_VERSION_10_16 101600
#endif
#ifndef MAC_OS_VERSION_11_0
#define MAC_OS_VERSION_11_0 110000
#endif
#ifndef MAC_OS_VERSION_12_0
#define MAC_OS_VERSION_12_0 120000
#endif
#ifndef MAC_OS_VERSION_13_0
#define MAC_OS_VERSION_13_0 130000
#endif
#ifndef NSINTEGER_DEFINED // appears with 10.5 in NSObjCRuntime.h
if defined(__LP64__) && __LP64__
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif // NSINTEGER_DEFINED
#ifndef CGFLOAT_DEFINED // appears with 10.5 in CGBase.h
if defined(__LP64__) && __LP64__
typedef double CGFloat;
#else
typedef float CGFloat;
#endif // CGFLOAT_DEFINED
#else
typedef struct CGContext* CGContextRef;
#endif // FL_LIBRARY || FL_INTERNALS
extern CGContextRef fl_gc;
#endif // FL_DOXYGEN

void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut = 0);
extern CGContextRef fl_mac_gc();
extern FLWindow *fl_mac_xid(const Fl_Window *win);
extern Fl_Window *fl_mac_find(FLWindow *);
class Fl_Gl_Window;
```
34.192 math.h

---

34.193 names.h File Reference

This file defines arrays of human readable names for FLTK symbolic constants.

Variables

- const char *const fl_callback_reason_names []
This is an array of callback reason names you can use to convert font numbers into names.

- const char * const fl_eventnames []

This is an array of event names you can use to convert event numbers into names.

- const char * const fl_fontnames []

This is an array of font names you can use to convert font numbers into names.

### 34.193.1 Detailed Description

This file defines arrays of human readable names for FLTK symbolic constants.

#### 34.194 names.h

Go to the documentation of this file.
34.195 platform.H

1 //
2 // Platform header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2023 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 // These are FLTK symbols that are necessary or useful for calling
17 // platform specific functions. This file #include's certain platform
18 // specific system header files that are necessary to declare platform
19 // specific FLTK functions, for instance "Windows.h" under Windows.
20 //
21 // You should include this file if (and ONLY if) you need to call
22 // platform specific functions directly.
23 //
24 // See FLTK documentation: chapter "Operating System Issues" on when
25 // you need to #include <FL/platform.H>
26 //
27 #if !defined(FL_PLATFORM_H) && !defined(FL_DOXYGEN)
28 # define FL_PLATFORM_H
29 # include <FL/Fl_Export.H>
30 # include <FL/platform_types.h> // will bring in FL/fl_types.h
31 # include <FL/fl_config.h> // for uchar
32 class FL_Window;
33 # if !defined(_WIN32)
34 # include "win32.H"
35 # elif defined(FLTK_USE_MAYLAND)
36 # include "wayland.H"
37 # elif defined(FLTK_USE_X11)
38 # include "x11.H"
39 # elif defined(__APPLE__)
40 # include "mac.H"
41 # endif // !_WIN32
42
43 # endif // cross-platform declarations
44
45 # if defined(FL_LIBRARY) || defined(FL_INTERNALS)
46 # include <FL/FL_Window.H>
47 # class FL_EXPORT Fl_X {
48 public:
49 Fl_uintptr_t xid;
50 Fl_Window* w;
51
52 # endif // cross-platform declarations
53
54 # endif // platform.H
Definitions of platform-dependent types.

### Macros

- **#define FL_COMMAND opaque**
  
  An alias for `FL_CTRL` on Windows and X11, or `FL_META` on MacOS X.

- **#define FL_CONTROL opaque**

  An alias for `FL_META` on Windows and X11, or `FL_CTRL` on MacOS X.

### Typedefs

- typedef opaque `fl_intptr_t`

  An integral type large enough to store a pointer or a long value.

- typedef opaque `FL_Offscreen`

  Platform-specific value representing an offscreen drawing buffer.

- typedef struct opaque * `FL_Region`

  Pointer to a platform-specific structure representing a collection of rectangles.

- typedef opaque `FL_SOCKET`

  Socket or file descriptor

- typedef opaque `FL_Timestamp`

  Platform-specific point in time, used for delta time calculation.

- typedef opaque `fl_uintptr_t`

  An unsigned integral type large enough to store a pointer or an unsigned long value.

- typedef struct opaque * `GLContext`

  Pointer to a platform-specific structure representing the window's OpenGL rendering context.

### 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.
34.196.2 Typedef Documentation

34.196.2.1 flintptr_t

typedef opaque flintptr_t
An integral type large enough to store a pointer or a long value.
A pointer value can be safely cast to flintptr_t, and later cast back to its initial pointer type without change to the
pointer value. A variable of type flintptr_t can also store a long int value.

34.196.2.2 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

34.196.2.3 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 fCairoRegion *
- Windows: HRGN
- macOS: struct fCocoaRegion *

34.196.2.4 Fl_Timestamp

typedef opaque Fl_Timestamp
Platform-specific point in time, used for delta time calculation.

Note
This type may be a struct. sizeof(Fl_Timestamp) may be different on different platforms. Fl_Timestamp may
change with future ABI changes.

34.196.2.5 fluintptr_t

typedef opaque fluintptr_t
An unsigned integral type large enough to store a pointer or an unsigned long value.
A pointer value can be safely cast to fluintptr_t, and later cast back to its initial pointer type without change to the
pointer value. A variable of type fluintptr_t can also store an unsigned long int value.
34.196.2.6 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

34.197 platform_types.h

Go to the documentation of this file.

/*
 * Copyright 2016-2023 by Bill Spitzak and others.
 * This library is free software. Distribution and use rights are outlined in the file "COPYING" which should have been included with this file. If this file is missing or damaged, see the license at:
 * https://www.fltk.org/COPYING.php
 * Please see the following page on how to report bugs and issues:
 * https://www.fltk.org/bugs.php
 */

#ifndef Fl_Platform_Types_H
#define Fl_Platform_Types_H

#ifdef FL_DOXYGEN

typedef opaque fl_intptr_t;
typedef opaque fl_uintptr_t;

typedef struct opaque Fl_Offscreen;
typedef opaque Fl_Region;
typedef opaque FL_SOCKET;
typedef struct opaque *GLContext;
typedef opaque Fl_Timestamp;

#define FL_COMMAND opaque
#define FL_CONTROL opaque
#else /* FL_DOXYGEN */

#ifndef FL_PLATFORM_TYPES_H
#define FL_PLATFORM_TYPES_H

#include <FL/fl_config.h>

/* Platform-dependent types are defined here. These types must be defined by any platform:
 * FL_SOCKET, struct dirent, fl_intptr_t, fl_uintptr_t
 * NOTE: *FIXME* AlbrechtS 13 Apr 2016 (concerning FL_SOCKET)
 * The Fl::add_fd() API is partially inconsistent because some of the methods explicitly use 'int', but the callback typedefs use FL_SOCKET. With the definition of FL_SOCKET below we can have different data sizes and different signedness of socket numbers on *some* platforms.
 */
#endif
#endif /* FL_DOXYGEN */
#endif /* FL_PLATFORM_TYPES_H */

#include <FL/fl_config.h>

#endif

/* Platform-dependent types are defined here. These types must be defined by any platform:
 * FL_SOCKET, struct dirent, fl_intptr_t, fl_uintptr_t
 * NOTE: *FIXME* AlbrechtS 13 Apr 2016 (concerning FL_SOCKET)
 */

#ifndef _WIN64

#ifndef FL_SOCKET

#include <stddef.h> /* stdint.h not available before VS 2010 (1600) */
#endif
#endif

#define _WIN64

#include <stddef.h>

#endif

Generated by Doxygen
```c
typedef long fl_intptr_t;
typedef unsigned long fl_uintptr_t;

#define FL_COMMAND fl_command_modifier()
#define FL_CONTROL fl_control_modifier()

#include <FL/Fl_Export.H>

// This is currently the same for all platforms, but may change in the future
struct Fl_Timestamp_t {
  long sec;
  long usec;
};

typedef struct Fl_Timestamp_t Fl_Timestamp;

// Allows all hybrid combinations except WIN32 + X11 with MSVC
#else /* ! _WIN64 */
# define _WIN32 defined(_WIN32)
#endif

typedef void *GLContext;
typedef void *Fl_Region;
typedef fl_uintptr_t Fl_Offscreen;

/* Allows all hybrid combinations except WIN32 + X11 with MSVC */
#if defined(_WIN32) && !defined(__MINGW32__)
struct dirent {char d_name[1];};
#else
# include <dirent.h>
#endif

#if defined(_WIN64) && defined(_MSC_VER)
typedef unsigned __int64 FL_SOCKET; /* FIXME - FL_SOCKET (see above) */
#else
typedef int FL_SOCKET;
#endif

// This is currently the same for all platforms, but may change in the future
struct Fl_Timestamp_t {
  long sec;
  long usec;
};

typedef struct Fl_Timestamp_t Fl_Timestamp;

#define FL_PLATFORM_TYPES_H */
#endif /* FL_PLATFORM_TYPES_H */

// This is currently the same for all platforms, but may change in the future
struct Fl_Timestamp_t {
  long sec;
  long usec;
};

typedef struct Fl_Timestamp_t Fl_Timestamp;

#define FL_PLATFORM_TYPES_H */
#endif /* FL_PLATFORM_TYPES_H */

#define FL_DOXYGEN */
#endif /* FL_DOXYGEN */

#define FL_DOXYGEN */
#endif /* FL_DOXYGEN */

#define FL_DOXYGEN */
#endif /* FL_DOXYGEN */

```

### 34.198 wayland.H File Reference

Definitions of functions specific to the Wayland platform.

**Typedefs**

- typedef struct _cairo cairo_t
- typedef void *EGLContext

**Functions**

- struct wl_compositor *fl_wl_compositor()
  - Returns the wl_compositor of the current Wayland session.
- struct wl_display *fl_wl_display()
  - Returns the Wayland display in use.
- Fl_Window *fl_wl_find (struct wl_window *)
  - Returns the Fl_Window corresponding to a given the platform-specific window reference.
- cairo_t *fl_wl_gc()
  - Returns the cairo context associated to the current window or Fl_Image_Surface.
- EGLContext fl_wl_glcontext (GLContext rc)
  - Returns the EGLContext corresponding to the given GLContext.
- struct wl_surface *fl_wl_surface (struct wl_window *xid)
  - Returns the wl_surface associated to a shown window.
- struct wl_window *fl_wl_xid (const Fl_Window *win)
  - Returns a platform-specific reference associated to a shown window.
34.198.1 Detailed Description
Definitions of functions specific to the Wayland platform.

34.198.2 Function Documentation

34.198.2.1 fl_wl_compositor()

struct wl_compositor * fl_wl_compositor()

Returns the wl_compositor of the current Wayland session.
This allows, for example, to create a wl_surface with

struct wl_surface *my_wl_surface = wl_compositor_create_surface(fl_wl_compositor());

34.199 wayland.H

Go to the documentation of this file.

34.200 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)

Generated by Doxygen
Returns the Fl_Window corresponding to the given Windows-specific window reference.

- **HDC** `fl_win32_gc()`
  
  Returns the Windows-specific graphics context for the current window.

- **HGLRC** `fl_win32_glcontext(GLContext rc)`
  
  Returns the Windows-specific GL rendering context corresponding to the given GLContext.

- **HWND** `fl_win32_xid(const Fl_Window *win)`
  
  Returns the Windows-specific window reference corresponding to the given Fl_Window object.

### 34.201 win32.H

Go to the documentation of this file.

```c
#ifndef FL_PLATFORM_H
#error "Never use <FL/win32.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H

#include <windows.h>
typedef HWND Window;

typedef struct HGLRC_ *HGLRC;
extern FL_EXPORT HGLRC fl_win32_glcontext(GLContext rc);
extern FL_EXPORT HWND fl_win32_xid(const Fl_Window *win);
extern FL_EXPORT Fl_Window *fl_win32_find(HWND);
```

Generated by Doxygen
# x.H

1 //
2 // *Deprecated* platform header file for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2018 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 // IMPORTANT: This file is deprecated since FLTK 1.4.0. DO NOT include it.
17 // FL/x.H will be removed in a future FLTK release.
18 //
19 // Please include <FL/platform.H> instead if you really need it. See
20 // documentation in FL/platform.H to decide whether you need that file.
21 //
22 #if !defined(Fl_X_H) && !defined(FL_DOXYGEN)
23 # define Fl_X_H
24 # include <FL/platform.H>
25 # include <FL/platform.H>
26 #endif // !Fl_X_H

## 34.203 x11.H File Reference

Definitions of functions specific to the X11 platform.

### Functions

- `cairo_t * fl_cairo_gc ()`
  
  Returns the Cairo-specific currently active graphics context (OPTION_USE_CAIRO=On).

- `Display * fl_x11_display ()`
  
  Returns the X11 Display in use.

- `Fl_Window * fl_x11_find (Window xid)`
  
  Returns the Fl_Window corresponding to the given Window reference.

- `GC fl_x11_gc ()`
  
  Returns the X11-specific currently active graphics context.

- `Window fl_x11_xid (const Fl_Window *win)`
  
  Returns the Window reference for the given Fl_Window, or zero if not shown().

Generated by Doxygen
34.203.1 Detailed Description
Definitions of functions specific to the X11 platform.

34.203.2 Function Documentation

34.203.2.1 fl_x11_find()

Fl_Window * fl_x11_find (Window xid)

Returns the Fl_Window corresponding to the given Window reference.

34.203.2.2 fl_x11_gc()

GC fl_x11_gc ()

Returns the X11-specific currently active graphics context.

34.203.2.3 fl_x11_xid()

Window fl_x11_xid (const Fl_Window * win)

Returns the Window reference for the given Fl_Window, or zero if not shown().

34.204 x11.H

Go to the documentation of this file.

// X11 platform header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

// Do not directly include this file, instead use <FL/platform.H>. It will
// include this file if FLTK_USE_X11 is defined. This is to encourage
// portability of even the system-specific code...

// X11 platform header file for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

// Do not directly include this file, instead use <FL/platform.H>. It will
// include this file if FLTK_USE_X11 is defined. This is to encourage
// portability of even the system-specific code...

#ifndef FL_PLATFORM_H
#error "Never use <FL/x11.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H
#include <FL/Enumerations.H>

#ifdef _ABIN32 || defined(_ABI64) // fix for broken SGI Irix X .h files
#pragma set woff 3322

Generated by Doxygen
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#if defined(_ABIN32) || defined(_ABI64)
#pragma reset woff
#endif
#include <X11/Xatom.h>

typedef struct __GLXcontextRec *GLXContext;
extern GLXContext fl_x11_glcontext(GLContext rc);

// constant info about the X server connection:
extern FL_EXPORT Display *fl_display;
extern FL_EXPORT Display *fl_x11_display();
extern FL_EXPORT Window fl_x11_xid(const Fl_Window *win);
extern FL_EXPORT Fl_Window *fl_x11_find(Window);
extern FL_EXPORT int fl_screen;
extern FL_EXPORT XVisualInfo *fl_visual;
extern FL_EXPORT Colormap fl_colormap;

// drawing functions:
extern FL_EXPORT GC fl_gc;
#if FLTK_USE_CAIRO
typedef struct _cairo cairo_t;
extern FL_EXPORT cairo_t * fl_cairo_gc();
#endif
extern FL_EXPORT GC fl_x11_gc();
FL_EXPORT ulong fl_xpixel(Fl_Color i);
FL_EXPORT ulong fl_xpixel(uchar r, uchar g, uchar b);

// feed events into fltk:
FL_EXPORT int fl_handle(const XEvent&);

// you can use these in Fl::add_handler() to look at events:
extern FL_EXPORT const XEvent* fl_xevent;
extern FL_EXPORT ulong fl_event_time;

#if defined(FL_LIBRARY) || defined(FL_INTERNALS)
extern FL_EXPORT Window fl_message_window;
extern FL_EXPORT void *fl_xftfont;

// access to core fonts:
// This class provides a "smart pointer" that returns a pointer to an XFontStruct.
// The global variable fl_xfont can be called wherever a bitmap "core" font is
// needed, e.g. when rendering to a GL context under X11.
// With Xft X11 fonts, fl_xfont will return the current selected font.
// With XFT / X11 fonts, fl_xfont will attempt to return the bitmap "core" font most
// similar to (usually the same as) the current XFT font.
class FL_EXPORT Fl_XFont_On_Demand
{
public:
    Fl_XFont_On_Demand(XFontStruct * p = NULL) : ptr(p) { }
    Fl_XFont_On_Demand& operator=(const Fl_XFont_On_Demand& x)
    { ptr = x.ptr; return *this; }
    Fl_XFont_On_Demand& operator=(XFontStruct * p)
    { ptr = p; return *this; }
    XFontStruct* value();
    operator XFontStruct *() { return value(); }
    XFontStruct & operator *() { return *value(); }
    XFontStruct * operator->() { return value(); }
    bool operator==(const Fl_XFont_On_Demand& x) { return ptr == x.ptr; }
    bool operator!=(const Fl_XFont_On_Demand& x) { return ptr != x.ptr; }
private:
    XFontStruct *ptr;
};
#endif // FL_LIBRARY || FL_INTERNALS

extern FL_EXPORT char fl_override_redirect; // hack into Fl_X::make_xid();

47 #endif
48 #include <X11/Xlib.h>
49 #include <X11/Xutil.h>
50 #if defined(_ABIN32) || defined(_ABI64)
51 # pragma reset woff
52 #endif
53 #include <X11/Xatom.h>
54 typedef struct __GLXcontextRec *GLXContext;
55 extern GLXContext fl_x11_glcontext(GLContext rc);
56 // constant info about the X server connection:
57 extern FL_EXPORT Display *fl_display;
58 extern FL_EXPORT Display *fl_x11_display();
59 extern FL_EXPORT Window fl_x11_xid(const Fl_Window *win);
60 extern FL_EXPORT Fl_Window *fl_x11_find(Window);
61 extern FL_EXPORT int fl_screen;
62 extern FL_EXPORT XVisualInfo *fl_visual;
63 extern FL_EXPORT Colormap fl_colormap;
64 extern FL_EXPORT XVisualInfo *fl_visual;
65 extern FL_EXPORT Colormap fl_colormap;
66 // drawing functions:
67 extern FL_EXPORT GC fl_gc;
68 #if FLTK_USE_CAIRO
69 typedef struct _cairo cairo_t;
70 extern FL_EXPORT cairo_t * fl_cairo_gc();
71 #endif
72 #ifndef
73 extern FL_EXPORT GC fl_x11_gc();
74 FL_EXPORT ulong fl_xpixel(Fl_Color i);
75 FL_EXPORT ulong fl_xpixel(uchar r, uchar g, uchar b);
76 // feed events into fltk:
77 FL_EXPORT int fl_handle(const XEvent&);
78 // you can use these in Fl::add_handler() to look at events:
79 extern FL_EXPORT const XEvent* fl_xevent;
80 extern FL_EXPORT ulong fl_event_time;
81 #if defined(FL_LIBRARY) || defined(FL_INTERNALS)
82 extern FL_EXPORT Window fl_message_window;
83 extern FL_EXPORT void *fl_xftfont;
84 #endif // FL_LIBRARY || FL_INTERNALS
85 // access to core fonts:
86 // This class provides a "smart pointer" that returns a pointer to an XFontStruct.
87 // The global variable fl_xfont can be called wherever a bitmap "core" font is
88 // needed, e.g. when rendering to a GL context under X11.
89 // With Xft X11 fonts, fl_xfont will return the current selected font.
90 // With XFT / X11 fonts, fl_xfont will attempt to return the bitmap "core" font most
91 // similar to (usually the same as) the current XFT font.
92 class FL_EXPORT Fl_XFont_On_Demand
93 {
94 public:
95 FL_EXPORT Fl_XFont_On_Demand(XFontStruct * p = NULL) : ptr(p) { }
96 FL_EXPORT Fl_XFont_On_Demand& operator=(const Fl_XFont_On_Demand& x)
97 { ptr = x.ptr; return *this; }
98 FL_EXPORT Fl_XFont_On_Demand& operator=(XFontStruct * p)
99 { ptr = p; return *this; }
100 XFontStruct* value();
101 operator XFontStruct *() { return value(); }
102 XFontStruct & operator *() { return *value(); }
103 XFontStruct * operator->() { return value(); }
104 bool operator==(const Fl_XFont_On_Demand& x) { return ptr == x.ptr; }
105 bool operator!=(const Fl_XFont_On_Demand& x) { return ptr != x.ptr; }
106 private:
107 XFontStruct *ptr;
108 #if defined(FL_LIBRARY) || defined(FL_INTERNALS)
109 #endif // FL_LIBRARY || FL_INTERNALS
110 #if defined(FL_LIBRARY) || defined(FL_INTERNALS)
111 #endif // FL_LIBRARY || FL_INTERNALS
112 extern FL_EXPORT Fl_XFont_On_Demand fl_xfont;
113 extern FL_EXPORT char fl_override_redirect; // hack into Fl_X::make_xid();
114
115 #endif // FL_LIBRARY || FL_INTERNALS
116
117 #endif // FL_DOXYGEN

1 //
2 // OS X Core Graphics debugging help for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2010 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php

34.205 cgdebug.h
This file allows easier debugging of Mac OS X Core Graphics code. This file is normally not included into any FLTK builds, but since it has proven to be tremendously useful in debugging the FLTK port to "Quartz", I decided to add this file in case more bugs show up.

This header is activated by adding the following line to "config.h"

```
#include "src/cgdebug.h"
```

Running ".configure" will remove this line from "config.h".

When used erroneously, Core Graphics prints warnings to stderr. This is helpful, however it is not possible to associate a line number or source file with the warning message.

This header file outputs a trace of CG calls, interweaving them with CG warnings.

```
inline OSStatus dbgLocation(const char *, int line)
{
    fprintf(stderr, "%s:%d ", file, line);
    return 0;
}
```

```
inline OSStatus dbgEndl()
{
    fprintf(stderr, "\n");
    return 0;
}
```

```
inline void dbgCGContextClipToRect(CGContextRef a, CGRect b)
{
    CGContextClipToRect(a, b);  
    fprintf(stderr, "\n");
    dbgCGContextClipToRect(a, b);  
    fprintf(stderr, "\n");
}
```
inline void dbgCGContextFillRect(CGContextRef a, CGRect b) {
CGContextFillRect(a, b);
}

#define CGContextFillRect(a, b) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextFillRect(a, b);
fprintf(stderr, "\n");
}

inline OSSStatus dbgQDEndCGContext(CGrafPtr a, CGContextRef *b) {
return QDEndCGContext(a, b);
}

#define QDEndCGContext(a, b) (
dbgLocation(__FILE__, __LINE__) +
dbgQDEndCGContext(a, b) +
dbgEndl() )

inline void dbgClipCGContextToRegion(CGContextRef a, const Rect *b, RgnHandle c) {
ClipCGContextToRegion(a, b, c);
}

#define ClipCGContextToRegion(a, b, c) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgClipCGContextToRegion(a, b, c);
fprintf(stderr, "\n");
}

inline void dbgCGContextMoveToPoint(CGContextRef context, float x, float y) {
CGContextMoveToPoint(context, x, y);
}

#define CGContextMoveToPoint(a, b, c) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextMoveToPoint(a, b, c);
fprintf(stderr, "\n");
}

inline void dbgCGContextFillPath(CGContextRef context) {
CGContextFillPath(context);
}

#define CGContextFillPath(a) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextFillPath(a);
fprintf(stderr, "\n");
}

inline void dbgCGContextClosePath(CGContextRef context) {
CGContextClosePath(context);
}

#define CGContextClosePath(a) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextClosePath(a);
fprintf(stderr, "\n");
}

inline void dbgCGContextFlush(CGContextRef context) {
CGContextFlush(context);
}

#define CGContextFlush(a) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextFlush(a);
fprintf(stderr, "\n");
}

inline void dbgCGContextSaveGState(CGContextRef context) {
CGContextSaveGState(context);
}

#define CGContextSaveGState(a) {
fprintf(stderr, "%s:%d \n", __FILE__, __LINE__);
dbgCGContextSaveGState(a);
fprintf(stderr, "\n");
}

Generated by Doxygen
```c
185  dbgCGContext.SaveGState(a); \
186  fprintf(stderr, "n"); }
188  inline void dbgCGContext.RestoreGState(CGContextRef context)
189  {
190    CGContext.RestoreGState(context);
191  }
192
193  #define CGContext.RestoreGState(a) { \
194    fprintf(stderr, "%s:%d \
", __FILE__, __LINE__); \
195    dbgCGContext.RestoreGState(a); \n    fprintf(stderr, "n"); } }
197
198
199  }
200
\section{34.206 fastarrow.h}

1#define fastarrow_width 16
2#define fastarrow_height 16
3static const unsigned char fastarrow_bits[] = {
4  0x00, 0x00, 0x00, 0x07, 0xe0, 0x07, 0xfc, 0x03, 0xff, 0xff, 0xfc, 0x03,
5  0xe0, 0x07, 0x00, 0x07, 0xe0, 0x00, 0xe0, 0x07, 0xc0, 0x3f, 0xff, 0xff,
6  0xc0, 0x3f, 0xe0, 0x07, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

\section{34.207 Fl.cxx File Reference}

Implementation of the member functions of class \texttt{Fl}.

\begin{verbatim}
#include <FL/Fl.H>
#include <FL/platform.H>
#include "FL_Screen_Driver.H"
#include "FL_Window_Driver.H"
#include "FL_System_Driver.H"
#include "Fl_Timeout.h"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <ctype.h>
#include <stdlib.h>
#include "flstring.h"
\end{verbatim}

Macros

- \#define \texttt{FOREVER} 1e20

Functions

- \texttt{bool fl_clipboard.notify_empty} (void)
- \texttt{void fl_close_display} ()
- \texttt{const char \* fl_filename_name} (const char \*name)
  
  Gets the file name from a path.
- \texttt{Fl_Window \* fl_find} (Window xid)
  
  Returns the \texttt{Fl_Window} that corresponds to the given window reference, or \texttt{NULL} if not found.
- \texttt{void fl_fix_focus} ()
- \texttt{void fl_open_callback} (void(+cb)(const char \*))
  
  Register a function called for each file dropped onto an application icon.
- \texttt{void fl_open_display} ()
  
  Opens the display.
- \texttt{int fl_send_system_handlers} (void \*e)
- \texttt{void fl_throw_focus} (Fl_Widget \*o)
### Variables

- **bool fl_disable_wayland = true**
  Prevent the FLTK library from using its Wayland backend and forces it to use its X11 backend.

- **const char * fl_local_alt = Fl::system_driver()->alt_name()**
  string pointer used in shortcuts, you can change it to another language

- **const char * fl_local_ctrl = Fl::system_driver()->control_name()**
  string pointer used in shortcuts, you can change it to another language

- **int(* fl_local_grab)(int)**

- **const char * fl_local_meta = Fl::system_driver()->meta_name()**
  string pointer used in shortcuts, you can change it to another language

- **const char * fl_local_shift = Fl::system_driver()->shift_name()**
  string pointer used in shortcuts, you can change it to another language

- **Fl_Widget * fl_oldfocus**

- **Fl_Widget * fl_selection_requestor**

### 34.207.1 Detailed Description

Implementation of the member functions of class Fl.

### 34.207.2 Function Documentation

#### 34.207.2.1 fl_find()

```cpp
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.

#### 34.207.2.2 fl_open_display()

```cpp
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.

### 34.207.3 Variable Documentation

#### 34.207.3.1 fl_disable_wayland

```cpp
bool fl_disable_wayland = true
```

Prevent the FLTK library from using its Wayland backend and forces it to use its X11 backend.

Put this declaration somewhere in your code outside the body of any function:

```cpp
FL_EXPORT bool fl_disable_wayland = true;
```

This declaration makes sure source code developed for FLTK 1.3, including X11-specific code, will build and run with FLTK 1.4 and its Wayland platform with this single source code level change. This declaration has no effect on non-Wayland platforms. Don't put this declaration if you want the Wayland backend to be used when it's available.
34.208  fl_arc.cxx File Reference

Utility functions for drawing arcs and circles.
#include <FL/fl_draw.h>
#include <FL/math.h>

34.208.1 Detailed Description

Utility functions for drawing arcs and circles.

34.209  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.
- 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 &ret, int maxchar, 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 &ret, int maxchar, 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

**34.209.1 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()**
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.

### 34.210 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.*
- `void fl_recbound (int x, int y, int w, int h, Fl_Color bgcolor)`  
  *Draws a bounded rectangle with a given position, size and color.*
- `void fl_thin_down_box (int x, int y, int w, int h, Fl_Color c)"
Draws a box of type FL_THIN_DOWN_BOX.

- void \texttt{fl\_thin\_down\_frame} (int x, int y, int w, int h, Fl\_Color)
  Draws a frame of type FL_THIN_DOWN_FRAME.

- void \texttt{fl\_thin\_up\_box} (int x, int y, int w, int h, Fl\_Color)
  Draws a box of type FL_THIN_UP_BOX.

- void \texttt{fl\_thin\_up\_frame} (int x, int y, int w, int h, Fl\_Color)
  Draws a frame of type FL_THIN_UP_FRAME.

- void \texttt{fl\_up\_box} (int x, int y, int w, int h, Fl\_Color)
  Draws a box of type FL_UP_BOX.

- void \texttt{fl\_up\_frame} (int x, int y, int w, int h, Fl\_Color)
  Draws a frame of type FL_UP_FRAME.

### 34.210.1 Detailed Description

Drawing code for common box types.

### 34.210.2 Function Documentation

#### 34.210.2.1 \texttt{fl\_internal\_boxtype()}

```c
void fl_internal_boxtype ( 
    Fl_Boxtype t, 
    Fl_Box_Draw_F * f )
```

Sets the drawing function for a given box type.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>box type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>f</td>
<td>box drawing function</td>
</tr>
</tbody>
</table>

#### 34.210.2.2 \texttt{fl\_rectbound()}

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

Draws a bounded rectangle with a given position, size and color. Equivalent to drawing a box of type FL_BORDER_BOX.

### 34.211 \texttt{fl\_cmap.h}

---

Generated by Doxygen
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
0x00000000, // 0
0xff000000, // 1
0x00ff0000, // 2
0xffff0000, // 3
0x0000ff00, // 4
0xff00ff00, // 5
0x00ffff00, // 6
0xc671c600, // 7
0x71c67100, // 8
0x8e8e3800, // 9
0x7171c600, // 10
0x8e388e00, // 11
0x388e8e00, // 12
0x0000ff00, // 13
0xff00ff00, // 14
0x00ffff00, // 15
0x55555500, // 16
0xc671c600, // 17
0x71c67100, // 18
0x8e8e3800, // 19
0x7171c600, // 20
0x8e388e00, // 21
0x388e8e00, // 22
0x0000ff00, // 23
0xff00ff00, // 24
0x00ffff00, // 25
0x55555500, // 26
0xc671c600, // 27
0x71c67100, // 28
0x8e8e3800, // 29
0x7171c600, // 30
0x8e388e00, // 31
0x388e8e00, // 32
<table>
<thead>
<tr>
<th>Line</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>0xbf480000</td>
<td>82</td>
</tr>
<tr>
<td>102</td>
<td>0xbf6d0000</td>
<td>83</td>
</tr>
<tr>
<td>103</td>
<td>0xbf910000</td>
<td>84</td>
</tr>
<tr>
<td>104</td>
<td>0xbf960000</td>
<td>85</td>
</tr>
<tr>
<td>105</td>
<td>0xbfda0000</td>
<td>86</td>
</tr>
<tr>
<td>106</td>
<td>0xbff0000</td>
<td>87</td>
</tr>
<tr>
<td>107</td>
<td>0x10000000</td>
<td>88</td>
</tr>
<tr>
<td>108</td>
<td>0x10000000</td>
<td>89</td>
</tr>
<tr>
<td>109</td>
<td>0x10240000</td>
<td>90</td>
</tr>
<tr>
<td>110</td>
<td>0x10480000</td>
<td>91</td>
</tr>
<tr>
<td>111</td>
<td>0x10910000</td>
<td>92</td>
</tr>
<tr>
<td>112</td>
<td>0x10b60000</td>
<td>93</td>
</tr>
<tr>
<td>113</td>
<td>0x10da0000</td>
<td>94</td>
</tr>
<tr>
<td>114</td>
<td>0x10ff0000</td>
<td>95</td>
</tr>
<tr>
<td>115</td>
<td>0x20000000</td>
<td>96</td>
</tr>
<tr>
<td>116</td>
<td>0x20240000</td>
<td>97</td>
</tr>
<tr>
<td>117</td>
<td>0x20480000</td>
<td>98</td>
</tr>
<tr>
<td>118</td>
<td>0x206d0000</td>
<td>99</td>
</tr>
<tr>
<td>119</td>
<td>0x20910000</td>
<td>100</td>
</tr>
<tr>
<td>120</td>
<td>0x20b60000</td>
<td>101</td>
</tr>
<tr>
<td>121</td>
<td>0x20da0000</td>
<td>102</td>
</tr>
<tr>
<td>122</td>
<td>0x20ff0000</td>
<td>103</td>
</tr>
<tr>
<td>123</td>
<td>0x3f000000</td>
<td>104</td>
</tr>
<tr>
<td>124</td>
<td>0x3f240000</td>
<td>105</td>
</tr>
<tr>
<td>125</td>
<td>0x3f480000</td>
<td>106</td>
</tr>
<tr>
<td>126</td>
<td>0x3f6d0000</td>
<td>107</td>
</tr>
<tr>
<td>127</td>
<td>0x3f910000</td>
<td>108</td>
</tr>
<tr>
<td>128</td>
<td>0x3fb60000</td>
<td>109</td>
</tr>
<tr>
<td>129</td>
<td>0x3fda0000</td>
<td>110</td>
</tr>
<tr>
<td>130</td>
<td>0x3ff3f000</td>
<td>111</td>
</tr>
<tr>
<td>131</td>
<td>0x7f003f00</td>
<td>112</td>
</tr>
<tr>
<td>132</td>
<td>0x7f243f00</td>
<td>113</td>
</tr>
<tr>
<td>133</td>
<td>0x7f483f00</td>
<td>114</td>
</tr>
<tr>
<td>134</td>
<td>0x7f6d3f00</td>
<td>115</td>
</tr>
<tr>
<td>135</td>
<td>0x7f913f00</td>
<td>116</td>
</tr>
<tr>
<td>136</td>
<td>0x7fb63f00</td>
<td>117</td>
</tr>
<tr>
<td>137</td>
<td>0x7fda3f00</td>
<td>118</td>
</tr>
<tr>
<td>138</td>
<td>0x7fff3f00</td>
<td>119</td>
</tr>
<tr>
<td>139</td>
<td>0xbf003f00</td>
<td>120</td>
</tr>
<tr>
<td>140</td>
<td>0xbf243f00</td>
<td>121</td>
</tr>
<tr>
<td>141</td>
<td>0xbf483f00</td>
<td>122</td>
</tr>
<tr>
<td>142</td>
<td>0xbf6d3f00</td>
<td>123</td>
</tr>
<tr>
<td>143</td>
<td>0xbf913f00</td>
<td>124</td>
</tr>
<tr>
<td>144</td>
<td>0xbfb63f00</td>
<td>125</td>
</tr>
<tr>
<td>145</td>
<td>0xbfda3f00</td>
<td>126</td>
</tr>
<tr>
<td>146</td>
<td>0xbf3f3f00</td>
<td>127</td>
</tr>
<tr>
<td>147</td>
<td>0xbff003f00</td>
<td>128</td>
</tr>
<tr>
<td>148</td>
<td>0xbf243f00</td>
<td>129</td>
</tr>
<tr>
<td>149</td>
<td>0xbf483f00</td>
<td>130</td>
</tr>
<tr>
<td>150</td>
<td>0xbf6d3f00</td>
<td>131</td>
</tr>
<tr>
<td>151</td>
<td>0xbf913f00</td>
<td>132</td>
</tr>
<tr>
<td>152</td>
<td>0xbfb63f00</td>
<td>133</td>
</tr>
<tr>
<td>153</td>
<td>0xbfda3f00</td>
<td>134</td>
</tr>
<tr>
<td>154</td>
<td>0xbf3f3f00</td>
<td>135</td>
</tr>
<tr>
<td>155</td>
<td>0xbf000000</td>
<td>136</td>
</tr>
<tr>
<td>156</td>
<td>0xbf240000</td>
<td>137</td>
</tr>
<tr>
<td>157</td>
<td>0xbf480000</td>
<td>138</td>
</tr>
<tr>
<td>158</td>
<td>0xbf6d0000</td>
<td>139</td>
</tr>
<tr>
<td>159</td>
<td>0xbf910000</td>
<td>140</td>
</tr>
<tr>
<td>160</td>
<td>0xbf6d0000</td>
<td>141</td>
</tr>
<tr>
<td>161</td>
<td>0xbfda0000</td>
<td>142</td>
</tr>
<tr>
<td>162</td>
<td>0xbf3f0000</td>
<td>143</td>
</tr>
<tr>
<td>163</td>
<td>0xbf000000</td>
<td>144</td>
</tr>
<tr>
<td>164</td>
<td>0xbf240000</td>
<td>145</td>
</tr>
<tr>
<td>165</td>
<td>0xbf480000</td>
<td>146</td>
</tr>
<tr>
<td>166</td>
<td>0xbf6d0000</td>
<td>147</td>
</tr>
<tr>
<td>167</td>
<td>0xbf910000</td>
<td>148</td>
</tr>
<tr>
<td>168</td>
<td>0xbf3f0000</td>
<td>149</td>
</tr>
<tr>
<td>169</td>
<td>0xbfda0000</td>
<td>150</td>
</tr>
<tr>
<td>170</td>
<td>0xbf3f0000</td>
<td>151</td>
</tr>
<tr>
<td>171</td>
<td>0xbf000000</td>
<td>152</td>
</tr>
<tr>
<td>172</td>
<td>0xbf240000</td>
<td>153</td>
</tr>
<tr>
<td>173</td>
<td>0xbf480000</td>
<td>154</td>
</tr>
<tr>
<td>174</td>
<td>0xbf6d0000</td>
<td>155</td>
</tr>
<tr>
<td>175</td>
<td>0xbf910000</td>
<td>156</td>
</tr>
<tr>
<td>176</td>
<td>0xbf6d0000</td>
<td>157</td>
</tr>
<tr>
<td>177</td>
<td>0xbfda0000</td>
<td>158</td>
</tr>
<tr>
<td>178</td>
<td>0xbf3f0000</td>
<td>159</td>
</tr>
<tr>
<td>179</td>
<td>0xbf000000</td>
<td>160</td>
</tr>
<tr>
<td>180</td>
<td>0xbf240000</td>
<td>161</td>
</tr>
<tr>
<td>181</td>
<td>0xbf480000</td>
<td>162</td>
</tr>
<tr>
<td>182</td>
<td>0xbf6d0000</td>
<td>163</td>
</tr>
<tr>
<td>183</td>
<td>0xbf910000</td>
<td>164</td>
</tr>
<tr>
<td>184</td>
<td>0xbf6d0000</td>
<td>165</td>
</tr>
<tr>
<td>185</td>
<td>0xbfda0000</td>
<td>166</td>
</tr>
<tr>
<td>186</td>
<td>0xbf3f0000</td>
<td>167</td>
</tr>
<tr>
<td>187</td>
<td>0xbf000000</td>
<td>168</td>
</tr>
</tbody>
</table>
1442 File Documentation

188 0xff247f00, // 169
189 0xff487f00, // 170
190 0xff6d7f00, // 171
191 0xff917f00, // 172
192 0xffb67f00, // 173
193 0xffda7f00, // 174
194 0xffff7f00, // 175
195 0x0000bf00, // 176
196 0x0024bf00, // 177
197 0x0048bf00, // 178
198 0x006dbf00, // 179
199 0x0091bf00, // 180
200 0x00b6bf00, // 181
201 0x00dabf00, // 182
202 0x00ffbf00, // 183
203 0x3f00bf00, // 184
204 0x3f24bf00, // 185
205 0x3f48bf00, // 186
206 0x3f6dbf00, // 187
207 0x3f91bf00, // 188
208 0x3fb6bf00, // 189
209 0x3fdabf00, // 190
210 0x3fffbf00, // 191
211 0x7f00bf00, // 192
212 0x7f48bf00, // 193
213 0x7f6dbf00, // 194
214 0x7f91bf00, // 195
215 0x7fb6bf00, // 196
216 0x7fdabf00, // 197
217 0x7fffbf00, // 198
218 0xff00bf00, // 199
219 0xff24bf00, // 200
220 0xff48bf00, // 201
221 0xff6dbf00, // 202
222 0xff91bf00, // 203
223 0xffb6bf00, // 204
224 0xffdabf00, // 205
225 0xffffbf00, // 206
226 0x0000ff00, // 207
227 0x0024ff00, // 208
228 0x0048ff00, // 209
229 0x006dff00, // 210
230 0x0091ff00, // 211
231 0x00b6ff00, // 212
232 0x00daff00, // 213
233 0x00ffff00, // 214
234 0x3f00ff00, // 215
235 0x3f24ff00, // 216
236 0x3f48ff00, // 217
237 0x3f6dff00, // 218
238 0x3f91ff00, // 219
239 0x3fb6ff00, // 220
240 0x3fdaff00, // 221
241 0x3fffff00, // 222
242 0x7f00ff00, // 223
243 0x7f48ff00, // 224
244 0x7f6dff00, // 225
245 0x7f91ff00, // 226
246 0x7fb6ff00, // 227
247 0x7fdaff00, // 228
248 0x7fffbf00, // 229
249 0xff00ff00, // 230
250 0xff24ff00, // 231
251 0xff48ff00, // 232
252 0xff6dff00, // 233
253 0xff91ff00, // 234
254 0xffb6ff00, // 235
255 0xffdaff00, // 236
256 0xffff00, // 237
257 0x0000ff00, // 238
258 0x0024ff00, // 239
259 0x0048ff00, // 240
260 0x006dff00, // 241
261 0x0091ff00, // 242
262 0x00b6ff00, // 243
263 0x00daff00, // 244
264 0x00ffff00, // 245
265 0x3f00ff00, // 246
266 0x3f24ff00, // 247
267 0x3f48ff00, // 248
268 0x3f6dff00, // 249
269 0x3f91ff00, // 250
270 0x3fb6ff00, // 251
271 0x3fdaff00, // 252
272 0x3fffbf00, // 253
273 0xff00ff00, // 254
274 0xffffff00 // 255
34.212 fl_color.cxx File Reference

Color handling.
#include <FL/Fl.H>
#include <FL/Fl_Device.H>
#include <FL/Fl_Graphics_Driver.H>
#include "fl_cmap.h"

Functions

• Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)
  Returns the weighted average color between the two given colors.

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

Variables

• unsigned fl_cmap[256]

34.212.1 Detailed Description

Color handling.

34.212.2 Variable Documentation

34.212.2.1 fl_cmap

unsigned fl_cmap[256]
Initial value:
= {
    
}

34.213 Fl_compose.cxx File Reference

Utility functions to support text input.
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"

34.213.1 Detailed Description

Utility functions to support text input.

34.214 fl_contrast.cxx File Reference

Color contrast handling.
#include <FL/Fl.H>
#include <math.h>
Functions

- `Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, Fl_Fontsize fs, int context)`
  Returns a color that contrasts with the background color.
- `void fl_contrast_function (Fl_Contrast_Function *f)`
  Register a custom contrast function.
- `int fl_contrast_level ()`
  Get the contrast level (sensitivity) of the fl_contrast() method.
- `void fl_contrast_level (int level)`
  Set the contrast level (sensitivity) of the fl_contrast() method.
- `int fl_contrast_mode ()`
  Return the current contrast algorithm (mode).
- `void fl_contrast_mode (int mode)`
  Set the contrast algorithm (mode).
- `double fl_lightness (Fl_Color color)`
  Return the perceived lightness of a color.
- `double fl_luminance (Fl_Color color)`
  Return the raw / physical luminance of a color.
- `unsigned get_color (Fl_Color i)`

Variables

- `unsigned fl_cmap [256]`

34.214.1 Detailed Description
Color contrast handling.
Implementation of fl_contrast() and its variants.

34.215 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>

34.215.1 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!

34.216 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"

34.216.1 Detailed Description
Fl_Double_Window implementation.
34.217 _Fl_Gl_Choice.H_

1 //
2 // OpenGL definitions for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2018 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 //    https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 //    https://www.fltk.org/bugs.php
15 //
16 // Internal interface to set up OpenGL.
17 //
18 // A "Fl_Gl_Choice" is created from an OpenGL mode and holds information
19 // necessary to create a window (on X) and to create an OpenGL "context"
20 // (on both X and Win32).
21 //
22 // create_gl_context takes a window (necessary only on Win32) and an
23 // Fl_Gl_Choice and returns a new OpenGL context. All contexts share
24 // display lists with each other.
25 //
26 // On X another create_gl_context is provided to create it for any
27 // X visual.
28 //
29 // set_gl_context makes the given OpenGL context current and makes
30 // it draw into the passed window. It tracks the current one context
31 // to avoid calling the context switching code when the same context
32 // is used, though it is a mystery to me why the GLX/WGL libraries
33 // don't do this themselves...
34 //
35 // delete_gl_context destroy the context.
36 //
37 // This code is used by Fl_Gl_Window, gl_start(), and gl_visual()
38 //
39 #ifndef Fl_Gl_Choice_H
40 #define Fl_Gl_Choice_H
41
42 // Describes the platform-independent part of data needed to create a GLContext.
43 class Fl_Gl_Choice {
44   friend class Fl_Gl_Window_Driver;
45   int mode;
46   const int *alist;
47   Fl_Gl_Choice *next;
48 public:
49   Fl_Gl_Choice(int m, const int *alistp, Fl_Gl_Choice *n) : mode(m), alist(alistp), next(n) {}
50   ~Fl_Gl_Choice();
51   void set_gl_context(Fl_Gl_Window *window);
52   void delete_gl_context();
53 #endif // Fl_Gl_Choice_H

34.218 _Fl_Gl_Window_Driver.H_

1 //
2 // Definition of class Fl_Gl_Window_Driver, and of its platform-specific derived classes
3 // for the Fast Light Tool Kit (FLTK).
4 //
5 // Copyright 2016-2018 by Bill Spitzak and others.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 //    https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:
14 //
15 //    https://www.fltk.org/bugs.php
16 //
17 #ifndef Fl_Gl_Window_Driver_H
18 #define Fl_Gl_Window_Driver_H
19
20 #include <FL/Fl_Gl_Window.H>
21 #include <GL/gl.h>
22
23 class Fl_Gl_Choice;
24 class Fl_Gl_Window_Driver;
25 class Fl_Gl_Renderer;
26 class Fl_Gl_Border;
27 class Fl_Gl_Window;
28
29 #endif // Fl_Gl_Window_Driver_H
class Fl_Gl_Window_Driver {
protected:
  GLint current_prog;
  Fl_Gl_Window *pWindow;
public:
  static GLContext cached_context;
  static Fl_Window *cached_window;
  static int nContext;
  static GLContext *context_list;
  static Fl_Gl_Choice *first;
  static int copy;
  static float gl_scale;
  static GLContext gl_start_context;
  Fl_Gl_Choice *g() {return pWindow->g;}
  void g(Fl_Gl_Choice *c) {pWindow->g = c;}
  int mode() {return pWindow->mode_;
  void set_mode(int m) {pWindow->mode_ = m;}
  void alist(const int *l) {pWindow->alist = l;}
  virtual ~Fl_Gl_Window_Driver() {}
This internal (undocumented) class reads data chunks from a file or from memory in LSB-first byte order.

This class is used in Fl_GIF_Image and Fl_BMP_Image to avoid code duplication and may be extended to be used in similar cases. Future options might be to read data in MSB-first byte order or to add more methods.

*/
#endif FL_IMAGE_READER_H
#define FL_IMAGE_READER_H

#include <stdio.h>

class Fl_Image_Reader {
  public:
  // Create the reader.
  Fl_Image_Reader()
    : is_file_(0)
    , is_data_(0)
    , file_(0L)
    , data_(0L)
    , start_(0L)
    , end_(const unsigned char *)(-1L)
    , name_(0L)
    , error_(0) {}

  // Initialize the reader to access the file system, filename is copied and stored.
  int open(const char *filename);

  // Initialize the reader for memory access, name is copied and stored
  int open(const char *imagename, const unsigned char *data, const size_t datasize);
  // Deprecated, DO NOT USE!
  int open(const char *imagename, const unsigned char *data);

  // Close and destroy the reader
  ~Fl_Image_Reader();

  // Read a single byte from memory or a file
  unsigned char read_byte();

  // Read a 16-bit unsigned integer, LSB-first
  unsigned short read_word();

  // Read a 32-bit unsigned integer, LSB-first
  unsigned int read_dword();

  // Read a 32-bit signed integer, LSB-first
  int read_long() { return (int)read_dword(); }

  // Move the current read position to a byte offset from the beginning
  // of the file or the original start address in memory
  void seek(unsigned int n);

  // Get the current file or memory offset from the beginning
  // of the file or the original start address in memory
  long tell() const;

  // Get the current EOF or error status of the file or data block
  int error() const { return error_; }

  // return the name or filename for this reader
  const char *name() const { return name_; }

  // skip a given number of bytes
  void skip(unsigned int n) { seek((unsigned int)tell() + n); }
};

Generated by Doxygen
private:
87 // open() sets this if we read from a file
88 char is_file_;
89 // open() sets this if we read from memory
90 char is_data_;
91 // a pointer to the opened file
92 FILE *file_;  
93 // a pointer to the current byte in memory
94 const unsigned char *data_;  
95 // a pointer to the start of the image data
96 const unsigned char *start_;  
97 // a pointer to the end of image data if reading from memory, otherwise undefined
98 // note: currently (const unsigned char *){-1L} if end of memory is not available
99 // ... which means "unlimited"
100 const unsigned char *end_;  
101 // a copy of the name associated with this reader
102 char name_;  
103 // a flag to store EOF or error status
104 int error_;  
105 
106 #endif // FL_IMAGE_READER_H

34.220  Fl_Message.h

/* Note: Do not FL_EXPORT this class, it’s for internal use only */
42
43 class Fl_Message_Box : public Fl_Box {
44 public:
45 Fl_Message_Box(int x, int y, int w, int h) : Fl_Box(x, y, w, h) {}
46 int handle(int e) FL_OVERRIDE;
47 }; // class Fl_Message_Box

72 /* Note: Do not FL_EXPORT this class, it’s for internal use only */
74 class Fl_Message {
75 
76 // static variables and methods
77 
78 private:
79 static Fl_Box *message_icon_; // returned by Fl_Message::message_icon()
80 static const char *message_title_;  
81 static const char *message_title_default_;  
82 static const char *message_icon_label_;  
83 // icon label for next dialog (STR #2762)
84 static const char *message_icon_label_;  
85 // Note: since Fl_Message objects are destroyed before fl_input()  
86 // and fl_password() return their input text, we *need* to store  
87 // the text in an internal (static) buffer. :-(  
88 // The newer functions fl_input_str() and fl_password_str() return the  
89 // text in an Fl_String object that must be allocated and free()'d by  
90 // the caller.
91 static char *input_buffer_; // points to the allocated text buffer  
92 static int input_size_; // size of allocated text buffer

Generated by Doxygen
34.221  Fl_Native_File_Chooser_Kdialog.H

// the callback for all buttons:
static void button_cb_(Fl_Widget *w, void *d);  

// the window callback:
static void window_cb_(Fl_Widget *w, void *d);  

// resize to make text and buttons fit
void resizeform();  

public:
static Fl_Box *message_icon();  
static void message_title(const char *title);  
static void message_title_default(const char *title);  
static void icon_label(const char *str);  
static void message_position(const int x, const int y, const int center) {  
form_x_ = x;  
form_y_ = y;  
form_position_ = center ? 2 : 1;  
}  
static void message_position(Fl_Widget *widget) {  
form_x_ = widget->x() + widget->w() / 2;  
form_y_ = widget->y() + widget->h() / 2;  
form_position_ = 2;  
}  
static int message_position(int *x, int *y) {  
if (x)  
*x = form_position_ ? form_x_ : -1;  
if (y)  
*y = form_position_ ? form_y_ : -1;  
return form_position_;  
}  
static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }  
static int message_hotspot() { return enable_hotspot_; }  
int window_closed()const {  
return window_closed_;  
}  

private:
Fl_Window *window_;  
Fl_Message_Box *message_;  
Fl_Box *icon_;  
Fl_Button *button_[3];  
Fl_Input *input_;  
int retval_;  
int window_closed_;  

static int enable_hotspot_;  
static int form_x_;  
static int form_y_;  
static int form_position_;  

public:
// Constructor  
Fl_Message(const char *iconlabel);  
~Fl_Message() { delete window_; }  
int innards(const char *fmt, va_list ap, const char *b0, const char *b1, const char *b2);  
const char *input_innards(const char *fmt, va_list ap, const char *defstr, uchar type, int maxchar = -1);  

#define __src_Fl_Message_h_

// FLTK native file chooser widget : KDE version  
// Copyright 2021-2022 by Bill Spitzak and others.  
// This library is free software. Distribution and use rights are outlined in  
// the file "COPYING" which should have been included with this file. If this  
// file is missing or damaged, see the license at:
#ifndef FL_KDIALOG_NATIVE_FILE_CHOOSER_H
#define FL_KDIALOG_NATIVE_FILE_CHOOSER_H 1

#include <FL/Fl_Native_File_Chooser.H>

class Fl_Kdialog_Native_File_Chooser_Driver : public Fl_Native_File_Chooser_FLTK_Driver {
friend class Fl_Native_File_Chooser;
friend class Fl_Zenity_Native_File_Chooser_Driver;
struct fnfc_pipe_struct {
  char *all_files;
  int fd;
};
static void fnfc_fd_cb(int fd, fnfc_pipe_struct *data);
char **_pathnames;
int _tpathnames;
char *directory;
char *_preset_file;
char * _title;
static bool did_find_kdialog;
static bool have_looked_for_kdialog;
Fl_Kdialog_Native_File_Chooser_Driver(int val);
~Fl_Kdialog_Native_File_Chooser_Driver();
int count() const FL_OVERRIDE;
const char *filename() const FL_OVERRIDE;
const char *filename(int i) const FL_OVERRIDE;
virtual char *build_command();
int show() FL_OVERRIDE;
char *parse_filter(const char *f);
const char *filter() const FL_OVERRIDE;
void filter(const char *f) FL_OVERRIDE;
int filters() const FL_OVERRIDE;
preset_file(const char *val) FL_OVERRIDE;
const char *preset_file() const FL_OVERRIDE;
directory(const char *val) FL_OVERRIDE;
const char *directory() const FL_OVERRIDE;
title(const char *val) FL_OVERRIDE;
const char *title() const FL_OVERRIDE;
};
#endif // FL_KDIALOG_NATIVE_FILE_CHOOSER_H

34.222  Fl_Native_File_Chooser_Zenity.H

#ifndef FL_ZENITY_NATIVE_FILE_CHOOSER_H
#define FL_ZENITY_NATIVE_FILE_CHOOSER_H 1

#include "Fl_Native_File_Chooser_Kdialog.H"

class Fl_Zenity_Native_File_Chooser_Driver : public Fl_Kdialog_Native_File_Chooser_Driver {
friend class Fl_Native_File_Chooser;
static bool did_find_zenity;
static bool have_looked_for_zenity;
Fl_Zenity_Native_File_Chooser_Driver(int val);
};
#endif // FL_ZENITY_NATIVE_FILE_CHOOSER_H


## 34.223 fl_oxy.h

```c
1 //
2 // "Oxy" Scheme drawing routines for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 2011 by Dmitrij K. aka "kdiman"
5 // Copyright 2012-2022 by Bill Spitzak and others.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please report all bugs and problems on the following page:
14 //
15 // https://www.fltk.org/str.php
16 //
17
18 #ifndef fl_oxy_h
19 #define fl_oxy_h
20
21 #include <FL/Fl.H>
22
23 #define OXY_ARROW
24 // draw an arrow GUI element for the 'oxy' scheme
25 //
26 // bb bounding box
27 // t arrow type
28 // o orientation
29 // c arrow color
30 extern FL_EXPORT void oxy_arrow(Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color col);
31 #endif // fl_oxy_h
```

## 34.224 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>
```

### 34.224.1 Detailed Description

Implementation of class Fl_Paged_Device.

## 34.225 fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.

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

### 34.225.1 Detailed Description

Drawing and clipping routines for rectangles.

## 34.226 Fl_Screen_Driver.H

```c
1 //
2 // All screen related calls in a driver style class.
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
```
#ifndef FL_SCREEN_DRIVER_H
#define FL_SCREEN_DRIVER_H

#include <FL/fl_types.h>
#include <FL/Fl.H> // for Fl_Timeout_Handler
#include <FL/Fl_Text_Editor.H>

// TODO: add text composition?
// TODO: add Fl::display
// TODO: add copy/paste, drag/drop?
// TODO: get key/get mouse?
// TODO: system colors/colormaps
// TODO: system menu?
// TODO: native filechooser
// TODO: native message boxes
// TODO: read screen to image
// TODO: application shortcuts

class Fl_Window;
class Fl_RGB_Image;
class Fl_Group;
class Fl_Input;
class Fl_System_Driver;

class Fl_Screen_Driver {

protected:
Fl_Screen_Driver();
virtual ~Fl_Screen_Driver();

static const int MAX_SCREENS = 16;

int num_screens;
static float fl_intersection(int x1, int y1, int w1, int h1,
int x2, int y2, int w2, int h2);

public:
static int keyboard_screen_scaling; // true means ctrl/+/-/0/ resize windows
static char bg_set;
static char bg2_set;
static char fg_set;
static Fl_System_Driver *system_driver;
// These flags are useful after calling XParseGeometry(). They indicate which of its
// arguments contain meaningful data upon return.
static const int fl_NoValue;
static const int fl_WidthValue;
static const int fl_HeightValue;
static const int fl_XValue;
static const int fl_YValue;
static const int fl_XNegative;
static const int fl_YNegative;
// key_table and key_table_size are used in fl_shortcut to translate key names
struct Keyname {
unsigned int key;
const char * name;
} *key_table;
int key_table_size;

virtual float scale(int) { return 1; }
virtual void scale(int /*n*/, float /*f*/) {}
static Fl_Screen_Driver *newScreenDriver();
// implement to process the -display argument and support the DISPLAY env var
virtual int XParseGeometry(const char * string, int* x, int* y, unsigned int* width, unsigned int* height);
virtual void own_colormap() {}
virtual const char *shortcut_add_key_name(unsigned key, char *p, char *buf, const char **);
virtual int need_menu_handle_part1_extra() {return 0;}
// whether a platform uses additional code in Fl_Menu::handle_part1(int e)
virtual int need_menu_handle_part2() {return 0;}
// whether a platform uses additional code in Fl_Menu::handle(int e)
virtual int get_key(int) { return 0; }
virtual int visual(int flags);
// --- screen configuration
virtual void init() {} // default, FL_OVERRIDE in driver!
virtual int x() { return 0; }
virtual int y() { return 0; }
virtual int w() { return 800; } // default, FL_OVERRIDE in driver!
virtual int h() { return 600; } // default, FL_OVERRIDE in driver!
virtual int screen_count();
void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my);
virtual void screen_xywh(int &X, int &Y, int &W, int &H, int /*n*/); // FL_OVERRIDE in driver!
X = 0;
Y = 0;
W = 800;
H = 600;
void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh);
virtual int screen_num(int x, int y);
virtual int screen_num(int x, int y, int w, int h);
virtual int screen_dpi(float &h, float &v, int n = 0) { // FL_OVERRIDE in driver!
    h = 72;
v = 72;
(void);
}
void screen_work_area(int &X, int &Y, int &W, int &H, int mx, int my);
virtual void screen_work_area(int &X, int &Y, int &W, int &H, int n) {
    screen_xywh(X, Y, W, H, n);
}
virtual void beep(int) {} // default, FL_OVERRIDE in driver!
virtual void flush() {} // must FL_OVERRIDE
virtual void grab(Fl_Window *) {} // default, FL_OVERRIDE in driver!
virtual int parse_color(const char *p, uchar &r, uchar &g, uchar &b);
virtual void get_system_colors() {} // default, FL_OVERRIDE in driver!
const char *get_system_scheme();
static int secret_input_character;
// Implement to indicate whether complex text input may involve marked text.
148 When it does, has_marked_text returns non zero.
virtual int has_marked_text() const { return 0; }
// implement so text-editing widgets support dead keys
152 virtual int compose(int &del) {
    del = 0;
    return 0;
}
// default implementation may be enough
virtual void compose_reset();
// implement to support drag-n-drop. use_selection = 1 means the GUI is welcome to display
// the selected text during the D&D operation
virtual int dnd(int use_selection = 0) { (void)use_selection; return 0; }
// null means no platform-specific key bindings for Fl_Text_Editor
// Fl_Text_Editor::key_binding *text_editor_extra_key_bindings;
// default implementation may be enough
virtual int text_display_can_leak() const { return 0; }
// if no keyboard is connected on a touch or pen device, the system on-screen keyboard is
// we no longer need the on-screen keyboard; it's up to the system to hide it
virtual void request_keyboard() {} // default, FL_OVERRIDE in driver!
virtual void release_keyboard() {} // default, FL_OVERRIDE in driver!

/win is the window to capture from, or NULL to capture from the current offscreen
virtual Fl_RGB_Image *read_win_rectangle(int /*X*/, int /*Y*/, int /*w*/, int /*h*/, Fl_Window *,
            bool may_capture_subwins = false,
            bool *did_capture_subwins = NULL) {
    (void)may_capture_subwins;
    (void)did_capture_subwins;
}
1454 File Documentation

```c
195  return NULL;
196 }
197 static void write_image_inside(Fl_RGB_Image *to, Fl_RGB_Image *from, int to_x, int to_y);
198 static Fl_RGB_Image *traverse_to_gl_subwindows(Fl_Group *g, int x, int y, int w, int h,
199  Fl_RGB_Image *full_img);
200 static size_t convert_crlf(char *s, size_t len);
201 // optional platform-specific key handling for Fl_Input widget
202 // the default implementation may be enough
203 virtual int input_widget_handle_key(int key, unsigned mods, unsigned shift, Fl_Input *input);
204 // implement to support Fl::get_mouse()
205 virtual int get_mouse(int &x, int &y); { return 0; }
206 // optional methods to enable/disable input methods for complex scripts
207 virtual void enable_latin() {}
208 virtual void disable_latin() {}
209 // calls open_display_platform() and then does platform-independent work
210 void open_display();
211 // implement to open access to the display
212 virtual void open_display_platform() {}
213 // optional method to close display access
214 virtual void close_display() {}
215 // compute dimensions of an Fl_Offscreen
216 virtual void offscreen_size(Fl_Offscreen, int &width, int &height) {} ()
217 void rescale_all_windows_from_screen(int screen, float f);
218 static void transient_scale_display(float f, int nscreen);
220 // need export to fltk_gl.so because used in glut_compatibility.cxx
221 static FL_EXPORT int scale_handler(int event);
222 virtual void desktop_scale_factor() {};
223 void use_startup_scale_factor();
224 enum APP_SCALING_CAPABILITY {
225  NO_APP_SCALING = 0,
226  SYSTEMWIDE_APP_SCALING,
227  PER_SCREEN_APP_SCALING
228 }
229 virtual APP_SCALING_CAPABILITY rescalable() { return NO_APP_SCALING; }
230 // supports Fl_Window::default_icons()
231 virtual void default_icons(const Fl_RGB_Image *icons[], int count);
232 // implement to support copy-to-clipboard
233 virtual void copy(const char *text, int &len, int &clipboard, const char *type);
234 // implement to support paste-from-clipboard
235 virtual void paste(Fl_Widget _, int &clipboard, const char *type);
236 // implement to support paste-from-clipboard
237 virtual int clipboard_contains(const char *type) { return 0; }
238 // implement to support paste-from-clipboard
239 virtual void clipboard_notify_change();
240 ...
241 #endif // !FL SCREEN DRIVER_H
```

34.227 Fl_String.cxx File Reference

Basic Fl_String class for FLTK.

#include `<FL/Fl_String.H`
#include `<stdio.h`
#include `<stdlib.h`
#include `<string.h`
#include `<limits.h`

**Functions**

- **Fl_String operator+** (const Fl_String &lhs, const char *rhs)
  
  Concatenate two strings.

- **Fl_String operator+** (const Fl_String &lhs, const Fl_String &rhs)
  
  Concatenate two strings.

- **bool operator==** (const Fl_String &lhs, const Fl_String &rhs)

  Compare two strings.
34.227.1 Detailed Description
Basic Fl_String class for FLTK.

34.227.2 Function Documentation

34.227.2.1 operator+() [1/2]

```c
Fl_String operator+ (  
    const Fl_String & lhs,  
    const char * rhs )
```
Concatenate two strings.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> lhs</td>
<td>first string</td>
</tr>
<tr>
<td><strong>in</strong> rhs</td>
<td>second C-style string</td>
</tr>
</tbody>
</table>

Returns
self

34.227.2.2 operator+() [2/2]

```c
Fl_String operator+ (  
    const Fl_String & lhs,  
    const Fl_String & rhs )
```
Concatenate two strings.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> lhs</td>
<td>first string</td>
</tr>
<tr>
<td><strong>in</strong> rhs</td>
<td>second string</td>
</tr>
</tbody>
</table>

Returns
self

34.227.2.3 operator==()

```c
bool operator== (  
    const Fl_String & lhs,  
    const Fl_String & rhs )
```
Compare two strings.

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> lhs</td>
<td>first string</td>
</tr>
<tr>
<td><strong>in</strong> rhs</td>
<td>second string</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

true if strings are the same size and have the same content

34.228  Fl_Sys_Menu_Bar_Driver.H

1 //
2 // system menu bar widget for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2017 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16 #ifndef Fl_Sys_Menu_Bar_Driver_H
17 #define Fl_Sys_Menu_Bar_Driver_H
18
19 #if !defined(FL_DOXYGEN)
20 #
21 #include <FL/Fl_Sys_Menu_Bar.H>
22
23 class Fl_Sys_Menu_Bar_Driver {
24     friend class Fl_Sys_Menu_Bar;
25 public:
26     static Fl_Sys_Menu_Bar::window_menu_style_enum window_menu_style_;
27     static Fl_Sys_Menu_Bar_Driver *driver_; // to be assigned with a unique object of this class or of a
28     derived class
29     Fl_Sys_Menu_Bar *bar;
30     Fl_Sys_Menu_Bar_Driver();
31     virtual ~Fl_Sys_Menu_Bar_Driver();
32     virtual void update() {};
33     virtual void draw() { bar->Fl_Menu_Bar::draw();}
34     virtual void about(Fl_Callback *, void *) {}
35     virtual int add(const char * label, int shortcut, Fl_Callback *cb, void *user_data, int flags) {
36         return bar->Fl_Menu_Bar::add(label, shortcut, cb, user_data, flags);
37     }
38     virtual int add(const char * str) { return bar->Fl_Menu_Bar::add(str); }
39     virtual int insert(int index, const char * label, int shortcut, Fl_Callback *cb, void *user_data, int
40         flags) {
41         return bar->Fl_Menu_Bar::insert(index, label, shortcut, cb, user_data, flags);
42     }
43     virtual void menu(const Fl_Menu_Item *m) { bar->Fl_Menu_Bar::menu(m); }
44     virtual void shortcut(int i, int s) { bar->Fl_Menu_Bar::shortcut(i, s); }
45     virtual void setonly(Fl_Menu_Item *item) { bar->Fl_Menu_Bar::setonly(item); }
46     virtual void clear() { bar->Fl_Menu_Bar::clear(); };
47     virtual int clear_submenu(int index) { return bar->Fl_Menu_Bar::clear_submenu(index); }
48     virtual void replace(int index, const char *name) { bar->Fl_Menu_Bar::replace(index, name); }
49     virtual void mode(int i, int fl) { bar->Fl_Menu_Bar::mode(i, fl); }
50     virtual void create_window_menu() {};
51     static Fl_Sys_Menu_Bar::window_menu_style_enum window_menu_style() { return window_menu_style_; }
52     static void window_menu_style(Fl_Sys_Menu_Bar::window_menu_style_enum style) { window_menu_style_ =
53         style; }
54     ;
55     ;
56     #endif // !defined(FL_DOXYGEN)
57     ;
58     #endif // Fl_Sys_Menu_Bar_Driver_H

34.229  Fl_System_Driver.H

1 //
2 // A base class for platform specific system calls
3 // for the Fast Light Tool Kit (FLTK).
4 //
5 // Copyright 2010-2022 by Bill Spitzak and others.
6 //
7 // This library is free software. Distribution and use rights are outlined in
8 // the file "COPYING" which should have been included with this file. If this
9 // file is missing or damaged, see the license at:
10 //
11 // https://www.fltk.org/COPYING.php
12 //
13 // Please see the following page on how to report bugs and issues:

Generated by Doxygen
/* Class hierarchy
+ Fl_System_Driver
 | + Fl_Posix_System_Driver
 | | + Fl_Unix_System_Driver
 | | + Fl_Darwin_System_Driver
 | + Fl_WinAPI_System_Driver */

#ifndef FL_SYSTEM_DRIVER_H
#define FL_SYSTEM_DRIVER_H

#include <FL/Fl.H>
#include <FL/Fl_Export.H>
#include <FL/filename.H>
#include <FL/Fl_Preferences.H>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>

class Fl_File_Icon;
class Fl_File_Browser;
class Fl_Pixmap;
class Fl_Widget;
class Fl_Sys_Menu_Bar_Driver;

class Fl_System_Driver {
friend class Fl;
protected:
// implement once for each platform
static Fl_System_Driver *newSystemDriver();
Fl_System_Driver();
static bool awake_ring_empty();
public:
virtual ~Fl_System_Driver();
static int command_key;
static int control_key;

// implement if the system adds unwanted program argument(s)
virtual int single_arg(const char *) { return 0; }
// implement if the system adds unwanted program argument pair(s)
virtual int arg_and_value(const char * /*name*/, const char * /*value*/) { return 0; }

void warning(const char * format, ...) {
    // implement to set the default effect of Fl::warning()
    
    // implement to set the default effect of Fl::error()
    
    // implement to set the default effect of Fl::fatal()
    
    // implement these to support cross-platform file operations
    virtual char *utf2mbcs(const char *s) {return (char*)s;}
    virtual char *getenv(const char*) {return NULL;}
    virtual int putenv(const char *) {return -1;}
    virtual int open(const char * /*f*/, int /*oflags*/, int /*pmode*/) {return -1;}

    // implement these to support cross-platform string operations
    virtual char *strdup(const char *) {return NULL;}
    virtual int open_ext(const char * f, int /*binary*/, int oflags, int pmode) {
        return this->open(f, oflags, pmode);
    }
    virtual FILE *fopen(const char* f, const char *mode);
    virtual int system(const char*) {return -1;}
    virtual int execlp(const char * /*file*/, char * /*const */ + /*argv*/[/*const */ + /*argv*/] { return -1;}
    virtual int chmod(const char * /*f*/, int /*mode*/) {return -1;}
    virtual int access(const char * /*f*/, int /*mode*/) { return -1;}
    virtual int fstat(const char * /*f*/, struct stat *) { return -1;}
    virtual char *getcwd(char* /*b*/, int /*l*/) {return NULL;}
    virtual int chdir(const char *) {return -1;}
    virtual int unlink(const char *) {return -1;}
    virtual int mkdir(const char * /*f*/, int /*mode*/) {return -1;}
    virtual int rmdir(const char *) {return -1;}
    virtual int rename(const char * /*f*/, const char * /*n*/) {return -1;}

    // the default implementation of these utf8... functions should be enough
    virtual unsigned utf8towc(const char * src, unsigned srclen, wchar_t * dst, unsigned dstlen);
    virtual unsigned utf8fromwc(char * dst, unsigned dstlen, const wchar_t * src, unsigned srclen);
};
virtual int utf8locall() { return 1; }

virtual unsigned utf8_from_mb(const char *src, unsigned srlen, char *dst, unsigned dstlen);

virtual unsigned utf8_from_mb(char *dst, dst, unsigned dstlen, const char *src, unsigned srclen);

implement to shield fprintf() from locale changes in decimal point

virtual int icolocale_vsnprintf(char *output, size_t output_size, const char *format, va_list args);
virtual int icolocale_vprintf(FILE *output, const char *format, va_list args);

implement scandir-like function

virtual const char *getpwnam(const char *) { return NULL; }

virtual int filename_absolute(char *to, int tolen, const char *from);

// the default implementation of filename_absolute() is in src/filename_absolute.cxx and may be enough

virtual int filename_relative(char *to, int tolen, const char *from, const char *base);

// the default implementation of filename_relative() is in src/filename_relative.cxx and may be enough

virtual int filename_isdir(const char *n);

// the default implementation of filename_isdir() is in src/filename_isdir.cxx and may be enough

virtual int filename_isdir_quick(const char *n);

// the default implementation of filename_isdir_quick() is in src/filename_isdir.cxx and may be enough

virtual int filename_list(const char * /*d*/, dirent ***pfiles, Fl_File_Sort_F *sort);

// implement scandir-like function

virtual int filename_expand(char *to, int tolen, const char *from);

// the default implementation of filename_expand() may be enough

virtual const char *filename_name(const char *buf) {return buf;}

// implement to support Fl_filename_name()

virtual const char *filename_ext(const char *buf);

// the default implementation of filename_ext() is in src/filename_ext.cxx and may be enough

virtual int filename_isdir(const char *n);

// the default implementation of filename_isdir() is in src/filename_isdir.cxx and may be enough

virtual int filename_isdir_quick(const char *n);

// the default implementation of filename_isdir_quick() is in src/filename_isdir.cxx and may be enough

virtual int open_uri(const char * /*uri*/, char * /*msg*/, int /*msglen*/) {return 0;}

// implement to support Fl_open_uri()

virtual int utf8from_mb(char * dst, unsigned dstlen, const char* src, unsigned srclen);
virtual int utf8to_mb(const char * src, unsigned srclen, char* dst, unsigned dstlen);

virtual int utf8locale() {return 1;}
// return TRUE when file names are case insensitive

virtual int case_insensitive_filenames() {return 0;}

// return TRUE when file names are case insensitive

virtual int dot_file_hidden() {return 0;}

// return TRUE means that files whose name begins with dot are hidden

virtual int colon_is_drive() {return 0;}

// return TRUE means : indicates a drive letter in file names

virtual int backslash_as_slash() {return 0;}

// return TRUE means same as / in file names

virtual const char *filesystems_label() { return "File Systems"; }

// the default implementation is most probably enough

virtual const char *home_directory_name() { return ""; }

// implement to return the user's home directory name

virtual const char *next_dir_sep(const char *start) { return strchr(start, '/'); }

// the default implementation is most probably enough

virtual char *preference_rootnode(Fl_Preferences *, Fl_Preferences::Root, const char * /*vendor*/, const char * /*application*/) {return NULL;}

// implement to support Fl_Preferences

virtual int file_browser_load_directory(const char *directory, char *filename, size_t name_size, char *errmsg=NULL, int errmsg_sz=0);

// implement to support Fl_File_Browser::load()

virtual int file_browser_load_filesystem(Fl_File_Browser *, char * /*filename*/, int /*lname*/, Fl_File_Icon *) {return 0;}

// implement to support Fl_File_Browser::load_directory() should be enough

virtual int file_browser_load_directory(const char * /*d*/, struct dirent **, struct dirent **, int /*dirflags*/, Fl_File_Sort_F *sort, int /*count*/, char *errmsg=NULL, int errmsg_sz=0);

// implement to support Fl_File_Browser::load()

virtual void newUUID(char *uuidBuffer) { uuidBuffer[0] = 0; }

// implement to support Fl_Preferences

virtual void *load(const char *); {return NULL;}

// implement to support Fl_Plugin_Manager::load()

virtual void *thread_message() {return NULL;}

// implement to support threading

virtual void unlock() {}

// implement to support Fl_File_IIcon

virtual void lock() { return 1; }

virtual void unlock() {} ;

virtual void* thread_message() {return NULL;}

virtual int utf8locall();

virtual int utf8locall(char *src, unsigned srlen, char *dst, unsigned dstlen);

virtual int utf8locall(char *dst, dst, unsigned dstlen, char *src, unsigned srclen);

implement to shield fprintf() from locale changes in decimal point

virtual int icolocale_vsnprintf(char *output, size_t output_size, char *format, va_list args);
virtual int icolocale_vprintf(FILE *output, char *format, va_list args);

implement scandir-like function

virtual const char *getpwnam(const char *) { return NULL; }

virtual int filename_absolute(char *to, int tolen, char *from, const char *base);

// the default implementation of filename_absolute() is in src/filename_absolute.cxx and may be enough

virtual int filename_relative(char *to, int tolen, char *from, const char *base);

// the default implementation of filename_relative() is in src/filename_relative.cxx and may be enough

virtual int filename_isdir(const char *n);

// the default implementation of filename_isdir() is in src/filename_isdir.cxx and may be enough

virtual int filename_isdir_quick(const char *n);

// the default implementation of filename_isdir_quick() is in src/filename_isdir.cxx and may be enough

virtual int filename_list(const char * /*d*/, dirent ***pfiles, Fl_File_Sort_F *sort);

// implement scandir-like function

virtual int filename_expand(char *to, int tolen, char *from);

// the default implementation of filename_expand() may be enough

virtual const char *filename_name(const char *buf) {return buf;}

// implement to support Fl_filename_name()

virtual const char *filename_ext(const char *buf);

// the default implementation of filename_ext() is in src/filename_ext.cxx and may be enough

virtual int filename_isdir(const char *n);

// the default implementation of filename_isdir() is in src/filename_isdir.cxx and may be enough

virtual int filename_isdir_quick(const char *n);

// the default implementation of filename_isdir_quick() is in src/filename_isdir.cxx and may be enough

virtual int open_uri(const char * /*uri*/, char * /*msg*/, int /*msglen*/) {return 0;}

// implement to support Fl_open_uri()

virtual int utf8from_mb(char * dst, unsigned dstlen, const char* src, unsigned srclen);
virtual int utf8to_mb(const char * src, unsigned srclen, char* dst, unsigned dstlen);

virtual int utf8locale() {return 1;}
// return TRUE when file names are case insensitive

virtual int case_insensitive_filenames() {return 0;}

// return TRUE when file names are case insensitive

virtual int dot_file_hidden() {return 0;}

// return TRUE means that files whose name begins with dot are hidden

virtual int colon_is_drive() {return 0;}

// return TRUE means : indicates a drive letter in file names

virtual int backslash_as_slash() {return 0;}

// return TRUE means same as / in file names

virtual const char *filesystems_label() { return "File Systems"; }

// return TRUE means files whose name begins with dot are hidden

virtual int dot_file_hidden() { return 0; }

// return TRUE when file names are case insensitive

virtual int colon_is_drive() {return 0;}

// return TRUE means files whose name begins with dot are hidden

virtual int backslash_as_slash() {return 0;}

// the implementations of local_to_latin1() and latin1_to_local() are in fl_encoding_latin1.cxx

virtual const char *local_to_latin1(const char *t, int n);

virtual const char *local_to_latin1(const char *, int n);

// the default implementation of local_to_latin1() and latin1_to_local() are in fl_encoding_latin1.cxx

virtual const char *mac_roman_to_local(const char *t, int n);

virtual const char *local_to_mac_roman(const char *t, int n);

// the implementations of local_to_mac_roman() and mac_roman_to_local() are in

// in Fl_Tree_Prefs.cxx and can be enough

// the default implementations of tree_openpixmap() and tree_closepixmap() are
virtual Fl_Pixmap *tree_openpixmap();
virtual Fl_Pixmap *tree_closepixmap();
static const char *const tree_open_xpm[]; // used by tree_openpixmap()
static const char *const tree_close_xpm[]; // used by tree_closepixmap()
virtual int tree_connector_style();
virtual void add_fd(int fd, int when, Fl_FD_Handler cb, void* = 0);
virtual void add_fd(int fd, Fl_FD_Handler cb, void* = 0);
virtual void remove_fd(int, int when);
virtual void remove_fd(int);
// the default implementation of open_callback() may be enough
virtual open_callback(void (*)(const char *));
// The default implementation may be enough.
virtual void gettimeofday(time_t *sec, int *usec);
virtual const char *shift_name() { return "Shift"; }
virtual const char *meta_name() { return "Meta"; }
virtual const char *alt_name() { return "Alt"; }
virtual const char *control_name() { return "Ctrl"; }
virtual Fl_Sys_Menu_Bar_Driver *sys_menu_bar_driver() { return NULL; }
virtual void lock_ring() {}
virtual void unlock_ring() {}
virtual double wait(double); // must FL_OVERRIDE
virtual int ready() { return 0; } // must FL_OVERRIDE
virtual int close_fd(int) { return -1; } // to close a file descriptor

#include "Fl_Timeout.h"
#include "Fl_System_Driver.H"
#include <stdio.h>

#include "FL/Fl.H"

class Fl_Timeout

The internal class Fl_Timeout handles all timeout related functions.

#define FL_TIMEOUT_DEBUG 0

34.230 Fl_Timeout.cxx File Reference

34.231 Fl_Timeout.h File Reference

Fl_Timeout handling.

34.231.1 Detailed Description

Fl_Timeout handling.
This file contains implementations of:

- Fl::add_timeout()
- Fl::repeat_timeout()
- Fl::has_timeout()
- Fl::remove_timeout()

and related methods of class Fl_Timeout.
// Header for timeout support functions for the Fast Light Tool Kit (FLTK).
// Author: Albrecht Schlosser
// Copyright 2021-2022 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#ifndef _src_Fl_Timeout_h_
define _src_Fl_Timeout_h_
#include <FL/Fl.H>
define FL_TIMEOUT_DEBUG 0 // 1 = include debugging features, 0 = no

class Fl_Timeout {
protected:
    Fl_Timeout *next; // ** Link to next timeout
    Fl_Timeout_Handler callback; // the user's callback
    void *data; // the user's callback data
    double time; // delay until timeout
    int skip; // skip "new" (inserted) timers (issue #450)

    // constructor
    Fl_Timeout() {
        next = 0;
        callback = 0;
        data = 0;
        time = 0;
        skip = 0;
    }

    // destructor
    ~Fl_Timeout() {}

    // get a new timer entry from the pool or allocate a new one
    static Fl_Timeout *get(double time, Fl_Timeout_Handler cb, void *data);

    // insert this timer into the active timer queue, sorted by expiration time
    void insert();

    // remove this timer from the active timer queue and
    // add it to the "current" timer stack
    void make_current();

    // remove this timer from the current timer stack and
    // add it to the list of free timers
    void release();

    double delay() {
        return time;
    }

    void delay(double t) {
        time = t;
    }

public:
    // Returns whether the given timeout is active.
    static int has_timeout(Fl_Timeout_Handler cb, void *data);

    // Add or remove timeouts
    static void add_timeout(double time, Fl_Timeout_Handler cb, void *data);
    static void repeat_timeout(double time, Fl_Timeout_Handler cb, void *data);
    static void remove_timeout(Fl_Timeout_Handler cb, void *data);

    // Elapse timeouts, i.e. calculate new delay time of all timers.
    static void elapse_timeouts();

    // Elapse timeouts and call timer callbacks.
};
### 34.233 fl_vertex.cxx File Reference

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

```cpp
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>
```

#### 34.233.1 Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

### 34.234 Fl_Window_Driver.H

```cpp
#include <FL/Fl_Export.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Overlay_Window.H>
#include <stdlib.h>

class Fl_X;

class Fl_Image;

class Fl_RGB_Image;

class Fl_Image_Surface;

class Fl_Window_Driver {
  // A base class for platform specific window handling code
  // for the Fast Light Tool Kit (FLTK).
  // Copyright 2010-2023 by Bill Spitzak and others.
  // This library is free software. Distribution and use rights are outlined in
  // the file "COPYING" which should have been included with this file. If this
  // file is missing or damaged, see the license at:
  // https://www.fltk.org/COPYING.php
  // Please see the following page on how to report bugs and issues:
  // https://www.fltk.org/bugs.php

  #ifndef FL_WINDOW_DRIVER_H
  #define FL_WINDOW_DRIVER_H

  #include <FL/Fl_Export.H>
  #include <FL/Fl_Window.H>
  #include <FL/Fl_Overlay_Window.H>
  #include <stdlib.h>

  class Fl_X;
  class Fl_Image;
  class Fl_RGB_Image;
  class Fl_Image_Surface;

  class Fl_Window_Driver {

  #endif // _src_Fl_Timeout_h_
```
1462 File Documentation

59 # Fl_Window *pWindow;
60 int screen_num_; // number of screen where window is mapped
public:
62 # Fl_Window_Driver(Fl_Window *);
63 virtual ~Fl_Window_Driver();
64 static Fl_Window_Driver *newWindowDriver(Fl_Window *);
65 static fl_uintptr_t xid(const Fl_Window *win);
66 static Fl_Window *find(fl_uintptr_t xid);
67 int wait_for_expose_value;
68 Fl_Image_Surface *other_xid; // offscreen bitmap (overlay and double-buffered windows)
69 int screen_num();
70 void screen_num(int n) { screen_num_ = n; }
71
72 // --- frequently used accessors to public window data
73 int x()const { return pWindow->x(); }
74 int y()const { return pWindow->y(); }
75 int w()const { return pWindow->w(); }
76 int h()const { return pWindow->h(); }
77 int border()const { return pWindow->border(); }
78 int visible()const { return pWindow->visible(); }
79 int visible_r()const { return pWindow->visible_r(); }
80 int shown()const { return pWindow->shown(); }
81 Fl_Group *parent()const { return pWindow->parent(); }
82
83 // --- accessors to private window data
84 int minw();
85 int minh();
86 int maxw();
87 int maxh();
88 int dw();
89 int dh();
100 int aspect();
101 unsigned char size_range_set();
102 int is_resizable() { return pWindow->is_resizable(); }
103 void is_a_rescale{bool b} [ is_a_rescale_ = b;]
104 int fullscreen_screen_top();
105 int fullscreen_screen_bottom();
106 int fullscreen_screen_left();
107 int fullscreen_screen_right();
108 int force_position();
109 void force_position(int c);
110 void x(int X);
111 void y(int Y);
112 void current(Fl_Window *c);
113 char show_iconic() { return Fl_Window::show_iconic_; }
114 void show_iconic(char c) { Fl_Window::show_iconic_ = c; }
115 void flx(Fl_X *x) { pWindow->flx_ = x; }
116 Fl_Cursor cursor_default() { return pWindow->cursor_default; }
117 void destroy_double_buffer();
118 Fl_Window *overlay() {
119 return pWindow->as_overlay_window() ? pWindow->as_overlay_window()->overlay_ : NULL;
120 }
121 }
122 void overlay(Fl_Window *o) { if (pWindow->as_overlay_window()) pWindow->as_overlay_window()->overlay_ = o; }
123 void resize_after_scale_change(int ns, float old_f, float new_f);
124
125 // --- window data
130 virtual int decorated_w() { return w(); } // default, should be overridden by driver
131 virtual int decorated_h() { return h(); }
132 virtual const Fl_Image * shape() { return NULL; }
133
134 // --- window management
135 virtual void take_focus();
136 virtual void flush(); // the default implementation may be enough
137 virtual void flush_double();
138 virtual void flush_overlay();
139 virtual void draw_begin();
140 virtual void draw_end();
141 void draw();
142 virtual void makeCurrent();
143 virtual void label(const char *name, const char *mininame);
144 virtual void makeWindow();
145 virtual void wait_for_expose();
146 virtual void show();
147 virtual void resize(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
148 virtual void hide() {}
149 int hide_common();
150 virtual void map() {}
151 virtual void unmap() {}
152 virtual void fullscreen_on() {}
153 virtual voidfullscreen_off(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
154 virtual void use_border();
155 virtual void size_range();
156
157 GetWaylandDisplay();
virtual void iconize() {}
virtual void iconize(int top, int left, int right, int bottom) {
  top = left = right = bottom = 0;
}
virtual void show_with_args_begin() {}
virtual void show_with_args_end(int /*argc*/, char ** /*argv*/) {}
virtual int can_do_overlay();
virtual void redraw_overlay();

// --- window cursor stuff
virtual int set_cursor(Fl_Cursor);
virtual int set_cursor(const Fl_RGB_Image*, int, int);

// --- window shape stuff
void shape_pixmap_(Fl_Image* pixmap); // platform-independent, support function
virtual void shape(const Fl_RGB_Image*) {} // platform-independent, support function
virtual void shape_alpha_(Fl_Image*, int /*offset*/) {}

// --- window icon stuff
virtual void icons(const Fl_RGB_Image* /*icons*/[], int /*count*/) {}
virtual const void* icon() const { return NULL; }
virtual void icon(const void*) {}
virtual void free_icons();

// --- window printing/drawing helper
virtual void capture_titlebar_and_borders(Fl_RGB_Image* top, Fl_RGB_Image* left, Fl_RGB_Image* bottom, Fl_RGB_Image* right);
virtual int scroll(int /*src_x*/, int /*src_y*/, int /*src_w*/, int /*src_h*/, int /*dest_x*/, int /*dest_y*/, void (*)(void*, int, int, int, int), void*) { return 0; }
static inline Fl_Window_Driver* driver(const Fl_Window* win) { return win->pWindowDriver; }

// --- support for menu windows
virtual void reposition_menu_window(int x, int y);
virtual void menu_window_area(int &X, int &Y, int &W, int &H, int nscreen = -1);
static Fl_Window* menu_parent(int* display_height = NULL);
static Fl_Window* menu_leftorigin(Fl_Window*);
static Fl_Window* menu_title(Fl_Window*);
static int menuheight(Fl_Window*);
static int menuitemheight(Fl_Window*);
static int menu bartitle(Fl_Window*);
static int menu selected(Fl_Window*);
static int menu selected(int n = 0);
static int menu selected_item(Fl_Window*);
static bool is_floating_title(Fl_Window*);
static int menu_itemheight(Fl_Window*);
static int menu_itemheight(int n);

virtual fl_uintptr_t os_id() { return 0; }
}

#endif // FL_WINDOW_DRIVER_H

34.235 fl_write_png.cxx File Reference

PNG image support functions.
#include "config.h"
#include "FL/Fl_PNG_Image.H"
#include "FL/Fl_RGB_Image.H"
#include "FL/fl_string_functions.h"
#include "FL/fl_utf8.h"
#include "stdio.h"
#include "time.h"

Functions

- int fl_write_png (const char *filename, const char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.
- int fl_write_png (const char *filename, const unsigned char *pixels, int w, int h, int d, int ld)
  Write raw image data to a PNG image file.
- int fl_write_png (const char *filename, Fl_RGB_Image *img)
  Write an RGB(A) image to a PNG image file.
34.235.1 Detailed Description

PNG image support functions.

34.235.2 Function Documentation

### 34.235.2.1 fl_write_png()

```c
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><code>filename</code></td>
<td>Output filename, extension should be '.png'</td>
</tr>
<tr>
<td><code>pixels</code></td>
<td>Image data</td>
</tr>
<tr>
<td><code>w</code></td>
<td>Image data width</td>
</tr>
<tr>
<td><code>h</code></td>
<td>Image data height</td>
</tr>
<tr>
<td><code>d</code></td>
<td>Image depth: 1 = GRAY, 2 = GRAY + alpha, 3 = RGB, 4 = RGBA</td>
</tr>
<tr>
<td><code>ld</code></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)`

#### 34.235.2.2 fl_write_png()

```c
int fl_write_png (const char *filename,
                  const unsigned char *pixels,
                  int w,
                  int h,
                  int d,
                  int ld)
```

Write raw image data to a PNG image file.

See also

`fl_write_png(const char *filename, const char *pixels, int w, int h, int d, int ld)`
34.235.2.3  fl_write_png() [3/3]

int fl_write_png (const char * filename, Fl_RGB_Image * img)

Write an RGB(A) image to a PNG image file.
This is a very basic and restricted function to create a PNG image file from an RGB image (Fl_RGB_Image).
The image data must be aligned w/o gaps, i.e. ld() MUST be zero or equal to data_w() * data_h().
The image file is always written with the original image size data_w() and data_h(), even if the image has been scaled.
Image depth 1 (gray), 2 (gray + alpha channel), 3 (RGB) and 4 (RGBA) are supported.

Note
Currently there is no error handling except for errors when opening the file. This may be changed in the future.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>Output filename, extension should be 'png'</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>img</td>
<td>RGB image to be written</td>
</tr>
</tbody>
</table>

Returns

success (0) or error code: negative values are errors

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>success, file has been written</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>png or zlib library not available</td>
</tr>
<tr>
<td>-2</td>
<td>file open error</td>
</tr>
</tbody>
</table>

See also

fl_write_png(const char *, int, int, int, const unsigned char *)

34.236  Fl_XColor.H

// X-specific color definitions for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php

#include <config.h>
#include <FL/Enumerations.H>

// one of these for each color in fltk's "colormap";
// if overlays are enabled, another one for the overlay
struct Fl_XColor {unsigned char r,g,b; // actual color used by X
unsigned char mapped; // true when XAllocColor done
unsigned long pixel; // the X pixel to use
};
extern Fl_XColor fl_xmap[][256];

// mask & shifts to produce xcolor for truecolor visuals:
30 extern unsigned char fl_redmask, fl_greenmask, fl_bluemask;
31 extern int fl_redshift, fl_greenshift, fl_blueshift, fl_extrashift;

34.237  flstring.h

1 /*
2 * Common string header file for the Fast Light Tool Kit (FLTK).
3 * Internal use only (see "important note" below).
4 *
5 * Copyright 1998-2020 by Bill Spitzak and others.
6 *
7 * This library is free software. Distribution and use rights are outlined in
8 * the file "COPYING" which should have been included with this file. If this
9 * file is missing or damaged, see the license at:
10 *
11 * https://www.fltk.org/COPYING.php
12 *
13 * Please see the following page on how to report bugs and issues:
14 *
15 * https://www.fltk.org/bugs.php
16 */
17
18 / *
19 * Important note: this header file includes '<config.h>' !
20 *
21 * This header MUST NOT be included in public headers (i.e. in 'FL/') and
22 * SHOULD NOT be included in test and demo programs (i.e. in 'test/' or
23 * 'examples/') because it includes '<config.h>'.
24 */
25
26 ifndef flstring_h
27 # define flstring_h
28
29 # include <FL/Fl_Export.H>
30 # include <config.h>
31 # include <stdio.h>
32 # include <stdarg.h>
33 # include <string.h>
34 # ifdef HAVE_STRINGS_H
35 # include <strings.h>
36 # endif /* HAVE_STRINGS_H */
37 # include <ctype.h>
38
39 /*
40 * Apparently Unixware defines "index" to strchr (!) rather than
41 * providing a proper entry point or not providing the (obsolete)
42 * BSD function. Make sure index is not defined...
43 */
44
45 # ifdef index
46 # undef index
47 # endif /* index */
48
49 /*
50 * Visual C++ 2005 incorrectly displays a warning about the use of
51 * POSIX APIs on Windows, which is supposed to be POSIX compliant...
52 * Some of these functions are also defined in ISO C99...
53 */
54
55 # if defined(_WIN32) && !defined(__CYGWIN__) && defined(__MINGW32__)
56 # define strcasecmp(s,t) _stricmp((s), (t))
57 # define strncasecmp(s,t,n) _strnicmp((s), (t), (n))
58 # endif /* _WIN32 && ... */
59
60 # ifdef __cplusplus
61 extern "C" {
62 # endif /* __cplusplus */
63
64 FL_EXPORT extern int fl_snprintf(char *, size_t, const char *, ...);
65 # ifndef HAVE_SNPRINTF
66 # define snprintf fl_snprintf
67 # endif /* !HAVE_SNPRINTF */
68
69 FL_EXPORT extern int fl_vsnprintf(char *, size_t, const char *, va_list ap);
70 # ifndef HAVE_VSNPRINTF
71 # define vsnprintf fl_vsnprintf
72 # endif /* !HAVE_VSNPRINTF */
73
74 /*
75 * strlcpy() and strlcat() are some really useful BSD string functions
76 * that work the way strcpy() and strncat() *should* have worked.
77 */
78
79 FL_EXPORT extern size_t fl_strlcat(char *, const char *, size_t);
80 # ifndef HAVE_STRLCAT
81
82 Generated by Doxygen
34.238 freeglut_teapot_data.h

1 /*
2 * freeglut_teapot_data.h
3 *
4 * The freeglut library teapot data include file.
5 *
6 * Permission is hereby granted, free of charge, to any person obtaining a
7 * copy of this software and associated documentation files (the "Software"),
8 * to deal in the Software without restriction, including without limitation
9 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
10 * and/or sell copies of the Software, and to permit persons to whom the
11 * Software is furnished to do so, subject to the following conditions:
12 *
13 * The above copyright notice and this permission notice shall be included
14 * in all copies or substantial portions of the Software.
15 *
16 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
17 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
19 * THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
20 * IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
21 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
22 */
23
24 #ifndef FREEGLUT_TEAPOT_DATA_H
25 #define FREEGLUT_TEAPOT_DATA_H
26
27 /*
28 * Original teapot code copyright follows:
29 */
30
31 /*
32 * (c) Copyright 1993, Silicon Graphics, Inc.
33 *
34 * ALL RIGHTS RESERVED
35 *
36 * Permission to use, copy, modify, and distribute this software
37 * for any purpose and without fee is hereby granted, provided
38 * that the above copyright notice appear in all copies and that
39 * both the copyright notice and this permission notice appear in
40 * supporting documentation, and that the name of Silicon
41 * Graphics, Inc. not be used in advertising or publicity
42 * pertaining to the software without specific, written prior permission.
43 *
44 * THE MATERIAL EMBODIED ON THIS SOFTWARE IS PROVIDED TO YOU
45 * "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR
46 * OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
47 * MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO
48 * EVENT SHALL SILICON GRAPHICS, INC. BE LIABLE TO YOU OR ANYONE
49 * ELSE FOR ANY DIRECT, SPECIAL, INCIDENTAL, INDIRECT OR
50 * CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER,
51 * INCLUDING WITHOUT LIMITATION, LOSS OF PROFIT, LOSS OF USE,
52 * SAVINGS OR REVENUE, OR THE CLAIMS OF THIRD PARTIES, WHETHER OR
53 * NOT SILICON GRAPHICS, INC. HAS BEEN ADVISED OF THE POSSIBILITY
54 * OF SUCH LOSS, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
55 * ARISING OUT OF OR IN CONNECTION WITH THE POSSESSION, USE OR
56 * PERFORMANCE OF THIS SOFTWARE.
57 *
58 * US Government Users Restricted Rights
59 *
60 * Use, duplication, or disclosure by the Government is subject to
61 * restrictions set forth in FAR 52.227.19(c)(2) or subparagraph
62 * (c)(1)(ii) of the Rights in Technical Data and Computer
```c
static int patchdata[16] =
{
  { 102, 103, 104, 105,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15 }, /* rim */
  { 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 }, /* body */
  { 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39 }, /* lid */
  { 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56 }, /* handle */
  { 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83 }, /* spout */
};

static double cpdata[3][3] =
{
  {0.2, 0, 2.7}, {0.2, -0.112, 2.7}, {0.112, -0.2, 2.7}, {0, -0.2, 2.7},
  {1.3375, 0, 2.53125}, {1.3375, -0.749, 2.53125}, {0.749, -1.3375, 2.53125}, {0, -1.3375, 2.53125}, {1.4375, 0, 2.53125}, {1.4375, -0.805, 2.53125}, {0.805, -1.4375, 2.53125}, {0, -1.4375, 2.53125}, {1.5, 0, 2.4}, {1.5, -0.84, 2.4}, {0.84, -1.5, 2.4}, {0, -1.5, 2.4},
  {1.75, -0.98, 1.875}, {0.98, -1.75, 1.875}, {0, -1.75, 1.875}, {2, 0, 1.35}, {2, -1.12, 1.35}, {1.12, -2, 1.35},
};

static double tex[2][2][2] =
{
  { {0.0, 0.0}, {1.0, 0.0} },
  { {0.0, 1.0}, {1.0, 1.0} }
};
```

Generated by Doxygen
# mediumarrow.h

```c
#define mediumarrow_width 16
#define mediumarrow_height 16
static const unsigned char mediumarrow_bits[] = {
  0x40, 0x00, 0x60, 0x00, 0x70, 0x00, 0x78, 0x00, 0xfc, 0x3f, 0x78, 0x00,
  0x70, 0x00, 0x60, 0x02, 0x40, 0x06, 0x00, 0xe, 0x0e, 0x80, 0x0e, 0x1e, 0xcf, 0x3f,
  0x00, 0x1e, 0x00, 0x0e, 0x00, 0x06, 0x00, 0x02};
```

## numericsort.c File Reference

```c
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <FL/platform_types.h>
#include <FL/filename.H>
#include <FL/fl_utf8.h>
```

### Functions

- int fl_casenumericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-insensitive).
- int fl_numericsort (struct dirent **A, struct dirent **B)
  
  Compares directory entries alphanumerically (case-sensitive).

## Function Documentation

### fl_casenumericsort()

```c
int fl_casenumericsort (struct dirent **A, struct dirent **B )
```

Compares directory entries alphanumerically (case-insensitive).

**Note**

This comparison is UTF-8 aware.

**See also**

fl_numericsort()

### 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
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>A &lt; B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>A == B</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>A &gt; B</td>
<td></td>
</tr>
</tbody>
</table>

Note

This comparison is UTF-8 aware.

See also

fl_casenumERICsort()

34.241 print_button.h

1 //
2 // Header for "Print Window" functions for the Fast Light Tool Kit (FLTK).
3 //
4 // Copyright 1998-2022 by Bill Spitzak and others.
5 //
6 // This library is free software. Distribution and use rights are outlined in
7 // the file "COPYING" which should have been included with this file. If this
8 // file is missing or damaged, see the license at:
9 //
10 // https://www.fltk.org/COPYING.php
11 //
12 // Please see the following page on how to report bugs and issues:
13 //
14 // https://www.fltk.org/bugs.php
15 //
16
17 #ifndef _SRC_FL_PRINT_BUTTON_H_
18 #define _SRC_FL_PRINT_BUTTON_H_
19 #ifdef __GNUC__
20 #endif
21 #include <FL/Fl_Window.H>
22 // These are all internal functions, do not FL_EXPORT these functions!
23 // These functions are implemented in src/print_button.cxx

Generated by Doxygen
// Print panel for the Fast Light Tool Kit (FLTK).
// Copyright 1998-2010 by Bill Spitzak and others.
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
// https://www.fltk.org/COPYING.php
// Please see the following page on how to report bugs and issues:
// https://www.fltk.org/bugs.php
// This is a temporary file. It is only for development and will
// probably be removed later.

#ifndef print_panel_h
#define print_panel_h
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Round_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Spinner.H>
#include <FL/Fl_Check_Button.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Progress.H>
enum printing_style {SystemV, BSD};
static Fl_Double_Window* make_print_panel();
static void print_cb(Fl_Return_Button *, void *);
static printing_style print_load();
static void print_update_status();
#endif

#define slowarrow_width 16
#define slowarrow_height 16
static const unsigned char slowarrow_bits[] = {
  0x40, 0x00, 0x40, 0x00, 0x60, 0x00, 0x60, 0x00, 0xf0, 0x0f, 0x60, 0x00,
  0x60, 0x00, 0x40, 0x02, 0x40, 0x02, 0x00, 0x06, 0x00, 0x06, 0xf0, 0x0f,
  0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x0f, 0x0f, 0x0f,
};

#define slowarrowwidth 16
#define slowarrowheight 16
#define slowarrow_width 16
#define slowarrow_height 16
static const unsigned char slowarrow_bits[] = {
  0x40, 0x00, 0x40, 0x00, 0x60, 0x00, 0x60, 0x00, 0xf0, 0x0f, 0x60, 0x00,
  0x60, 0x00, 0x40, 0x02, 0x40, 0x02, 0x00, 0x06, 0x00, 0x06, 0xf0, 0x0f,
  0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x00, 0x06, 0x0f, 0x0f, 0x0f,
};
This file may only contain common, platform-independent function declarations used internally in FLTK. It may be included everywhere in source files in the library, but not in public header files.

```c
#ifndef _SRC__FL_UTF8_H
#define _SRC__FL_UTF8_H

#ifdef __cplusplus
extern "C" {
#endif

unsigned short XUtf8IsNonSpacing(unsigned int ucs);
unsigned short XUtf8IsRightToLeft(unsigned int ucs);

int XUtf8Tolower(int ucs);
int XUtf8Toupper(int ucs);

#ifdef __cplusplus
}
#endif
#endif /* _SRC__FL_UTF8_H */
```

### 34.245 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.*

### 34.245.1 Detailed Description

Portable vsnprintf() implementation.

### 34.245.2 Function Documentation

#### 34.245.2.1 fl_vsnprintf()

```c
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.

This function guarantees:

- access to vsnprintf(), even on systems that don't have it (FLTK'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 FLTK has vsnprintf(), fl_vsnprintf() is just a wrapper around the compiler's provided function. Otherwise, if the function is NOT available, FLTK's own built-in version is provided.

The FLTK 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, l, o, u, x
- Pointer format: p
- String/char: c, s, n

34.246  Xutf8.h

1 /*
2 * Author: Jean-Marc Lienher ( http://oksid.ch )
3 * Copyright 2000-2010 by O'ksi'D.
4 * This library is free software. Distribution and use rights are outlined in
5 * the file "COPYING" which should have been included with this file. If this
6 * file is missing or damaged, see the license at:
7 * https://www.fltk.org/COPYING.php
8 * Please see the following page on how to report bugs and issues;
9 * https://www.fltk.org/bugs.php
10 */
11
12 #if ! ( defined(_Xutf8_h) || defined(FL_DOXYGEN) )
13 #define _Xutf8_h
14
15 # ifdef __cplusplus
16 extern "C" {
17 # endif
18 
19 # include <X11/X.h>
20 # include <X11/Xlib.h>
21 # include <X11/Xlocale.h>
22 # include <X11/Xutil.h>
23 #include <FL/Fl_Export.H>
24 
25 /*
26 typedef struct {
27 int nb_font;
28 char **font_name_list;
29 int *encodings;
30 XFontStruct **fonts;
31 Font fid;
32 int ascent;
33 int descent;
34 int *ranges;
35 } XUtf8FontStruct;
36 */
37
38 # define XCreateUtf8FontStruct
39 20 extern "C" { 
40 19 # define _Xutf8_h
41 18 # ifdef __cplusplus
42 17 extern "C" {
43 16 # include <X11/X.h>
44 15 # include <X11/Xlib.h>
45 14 # include <X11/Xlocale.h>
46 13 # include <X11/Xutil.h>
47 12 #include <FL/Fl_Export.H>
48 11 typedef struct {
49 int nb_font;
50 char **font_name_list;
51 int *encodings;
52 XFontStruct **fonts;
53 Font fid;
54 int ascent;
55 int descent;
56 int *ranges;
57 } XUtf8FontStruct;
58 */
59 
60 XUtf8FontStruct *
61 XCreateUtf8FontStruct {
62 Display *dpy,
63 const char *base_font_name_list);
64 
65 void
66 
67 Generated by Doxygen
XUtf8DrawString:
  Display *display,
  Drawable d,
  XUtf8FontStruct *font_set,
  GC gc,
  int x,
  int y,
  const char *string,
  int num_bytes);

void XUtf8_measure_extents:
  Display *display,
  Drawable d,
  XUtf8FontStruct *font_set,
  GC gc,
  int *xx,
  int *yy,
  int *ww,
  int *hh,
  const char *string,
  int num_bytes);

void XUtf8DrawRtlString:
  Display *display,
  Drawable d,
  XUtf8FontStruct *font_set,
  GC gc,
  int x,
  int y,
  const char *string,
  int num_bytes);

void XUtf8DrawImageString:
  Display *display,
  Drawable d,
  XUtf8FontStruct *font_set,
  GC gc,
  int x,
  int y,
  const char *string,
  int num_bytes);

int XUtf8TextWidth:
  XUtf8FontStruct *font_set,
  const char *string,
  int num_bytes);

int XUtf8UcsWidth:
  XUtf8FontStruct *font_set,
  unsigned int ucs);

int FL_EXPORT fl_XGetUtf8FontAndGlyph:
  XUtf8FontStruct *font_set,
  unsigned int ucs,
  XFontStruct **fnt,
  unsigned short *id);

void XFreeUtf8FontStruct:
  Display *dpy,
  XUtf8FontStruct *font_set);

int XConvertUtf8ToUcs:
  const unsigned char *buf,
  int len,
  unsigned int *ucs);

int XConvertUcsToUtf8:
  unsigned int ucs,
  char *buf);
```c
int len);

int XFastConvertUtf8ToUcs(const unsigned char *buf,
int len,
unsigned int *ucs);

long XKeysymToUcs(KeySym keysym);

#ifdef X_HAVE_UTF8_STRING
#define XUtf8LookupString Xutf8LookupString
#else
int XUtf8LookupString(
XIC ic,
char* buffer_return,
int bytes_buffer,
KeySym* keysym,
Status* status_return);
#endif
#endif

#ifdef __cplusplus
}
#endif
#endif

static const unsigned short ucs_table_0041[] = {
U+0041 0x0061,
U+0042 0x0062,
U+0043 0x0063,
U+0044 0x0064,
U+0045 0x0065,
U+0046 0x0066,
U+0047 0x0067,
U+0048 0x0068,
U+0049 0x0069,
U+004A 0x006A,
U+004B 0x006B,
U+004C 0x006C,
U+004D 0x006D,
U+004E 0x006E,
U+004F 0x006F,
};
```
227 /* U+0120 */ 0x0121,
228 0x00,
229 /* U+0122 */ 0x0123,
230 0x00,
231 /* U+0124 */ 0x0125,
232 0x00,
233 /* U+0126 */ 0x0127,
234 0x00,
235 /* U+0128 */ 0x0129,
236 0x00,
237 /* U+012A */ 0x012B,
238 0x00,
239 /* U+012C */ 0x012D,
240 0x00,
241 /* U+012E */ 0x012F,
242 0x00,
243 /* U+0130 */ 0x0131,
244 0x00,
245 /* U+0132 */ 0x0133,
246 0x00,
247 /* U+0134 */ 0x0135,
248 0x00,
249 /* U+0136 */ 0x0137,
250 0x00,
251 0x00,
252 /* U+0139 */ 0x013A,
253 0x00,
254 /* U+013B */ 0x013C,
255 0x00,
256 /* U+013D */ 0x013E,
257 0x00,
258 /* U+013F */ 0x0140,
259 0x00,
260 /* U+0141 */ 0x0142,
261 0x00,
262 /* U+0143 */ 0x0144,
263 0x00,
264 /* U+0145 */ 0x0146,
265 0x00,
266 /* U+0147 */ 0x0148,
267 0x00,
268 0x00,
269 /* U+014A */ 0x014B,
270 0x00,
271 /* U+014C */ 0x014D,
272 0x00,
273 /* U+014E */ 0x014F,
274 0x00,
275 /* U+0150 */ 0x0151,
276 0x00,
277 /* U+0152 */ 0x0153,
278 0x00,
279 /* U+0154 */ 0x0155,
280 0x00,
281 /* U+0156 */ 0x0157,
282 0x00,
283 /* U+0158 */ 0x0159,
284 0x00,
285 /* U+015A */ 0x015B,
286 0x00,
287 /* U+015C */ 0x015D,
288 0x00,
289 /* U+015E */ 0x015F,
290 0x00,
291 /* U+0160 */ 0x0161,
292 0x00,
293 /* U+0162 */ 0x0163,
294 0x00,
295 /* U+0164 */ 0x0165,
296 0x00,
297 /* U+0166 */ 0x0167,
298 0x00,
299 /* U+0168 */ 0x0169,
300 0x00,
301 /* U+016A */ 0x016B,
302 0x00,
303 /* U+016C */ 0x016D,
304 0x00,
305 /* U+016E */ 0x016F,
306 0x00,
307 /* U+0170 */ 0x0171,
308 0x00,
309 /* U+0172 */ 0x0173,
310 0x00,
311 /* U+0174 */ 0x0175,
312 0x00,
313 /* U+0176 */ 0x0177,
314 0x00,
315 / u+017B /* 0xFF,  
316 / u+0179 /* 0x017A,  
317 0x00,  
318 / u+017B /* 0x017C,  
319 0x00,  
320 / u+017D /* 0x017E,  
321 0x00,  
322 0x00,  
323 0x00,  
324 / u+0181 /* 0x0253,  
325 / u+0182 /* 0x0183,  
326 0x00,  
327 / u+0184 /* 0x0185,  
328 0x00,  
329 / u+0186 /* 0x0254,  
330 / u+0187 /* 0x0188,  
331 0x00,  
332 / u+0189 /* 0x0,  
333 / u+018A /* 0x0257,  
334 / u+018B /* 0x018C,  
335 0x00,  
336 0x00,  
337 / u+018E /* 0x0258,  
338 / u+018F /* 0x0259,  
339 / u+0190 /* 0x025B,  
340 / u+0191 /* 0x0192,  
341 0x00,  
342 / u+0193 /* 0x0260,  
343 / u+0194 /* 0x0263,  
344 0x00,  
345 / u+0196 /* 0x0269,  
346 / u+0197 /* 0x0268,  
347 / u+0198 /* 0x0199,  
348 0x00,  
349 0x00,  
350 0x00,  
351 / u+019C /* 0x026F,  
352 / u+019D /* 0x0272,  
353 0x00,  
354 / u+019F /* 0x0,  
355 / u+01A0 /* 0x01A1,  
356 0x00,  
357 / u+01A2 /* 0x01A3,  
358 0x00,  
359 / u+01A4 /* 0x01A5,  
360 0x00,  
361 0x00,  
362 / u+01A7 /* 0x01A8,  
363 0x00,  
364 / u+01A9 /* 0x0283,  
365 0x00,  
366 0x00,  
367 / u+01AC /* 0x01AD,  
368 0x00,  
369 / u+01AE /* 0x0288,  
370 / u+01AF /* 0x0180,  
371 0x00,  
372 / u+01B1 /* 0x028A,  
373 / u+01B2 /* 0x028B,  
374 / u+01B3 /* 0x0184,  
375 0x00,  
376 / u+01B5 /* 0x0186,  
377 0x00,  
378 / u+01B7 /* 0x0292,  
379 / u+01B8 /* 0x0189,  
380 0x00,  
381 0x00,  
382 0x00,  
383 / u+01BC /* 0x018D,  
384 0x00,  
385 0x00,  
386 0x00,  
387 0x00,  
388 0x00,  
389 0x00,  
390 0x00,  
391 / u+01C4 /* 0x01C6,  
392 / u+01C5 /* 0x0,  
393 0x00,  
394 / u+01C7 /* 0x01C9,  
395 / u+01C8 /* 0x0,  
396 0x00,  
397 / u+01CA /* 0x01CC,  
398 / u+01CB /* 0x0,  
399 0x00,  
400 / u+01CD /* 0x01CE,
401 0x00,
402 } /* U+01CF */ 0x01D0,
403 0x00,
404 } /* U+01D1 */ 0x01D2,
405 0x00,
406 } /* U+01D3 */ 0x01D4,
407 0x00,
408 } /* U+01D5 */ 0x01D6,
409 0x00,
410 } /* U+01D7 */ 0x01D8,
411 0x00,
412 } /* U+01D9 */ 0x01DA,
413 0x00,
414 } /* U+01DB */ 0x01DC,
415 0x00,
416 0x00,
417 } /* U+01DE */ 0x01DF,
418 0x00,
419 } /* U+01E0 */ 0x01E1,
420 0x00,
421 } /* U+01E2 */ 0x01E3,
422 0x00,
423 } /* U+01E4 */ 0x01E5,
424 0x00,
425 } /* U+01E6 */ 0x01E7,
426 0x00,
427 } /* U+01E8 */ 0x01E9,
428 0x00,
429 } /* U+01EA */ 0x01EB,
430 0x00,
431 } /* U+01EC */ 0x01ED,
432 0x00,
433 } /* U+01EE */ 0x01EF,
434 0x00,
435 0x00,
436 } /* U+01F1 */ 0x01F3,
437 } /* U+01F2 */ 0x00,
438 0x00,
439 } /* U+01F4 */ 0x01F5,
440 0x00,
441 0x00,
442 0x00,
443 0x00,
444 0x00,
445 } /* U+01FA */ 0x01FB,
446 0x00,
447 } /* U+01FC */ 0x01FD,
448 0x00,
449 } /* U+01FE */ 0x01FF,
450 0x00,
451 } /* U+0200 */ 0x0201,
452 0x00,
453 } /* U+0202 */ 0x0203,
454 0x00,
455 } /* U+0204 */ 0x0205,
456 0x00,
457 } /* U+0206 */ 0x0207,
458 0x00,
459 } /* U+0208 */ 0x0209,
460 0x00,
461 } /* U+020A */ 0x020B,
462 0x00,
463 } /* U+020C */ 0x020D,
464 0x00,
465 } /* U+020E */ 0x020F,
466 0x00,
467 } /* U+0210 */ 0x0211,
468 0x00,
469 } /* U+0212 */ 0x0213,
470 0x00,
471 } /* U+0214 */ 0x0215,
472 0x00,
473 } /* U+0216 */ 0x0217,
575 0x00,
576 0x00,
577 0x00,
578 0x00,
579 /* U+0280 */ 0x0,
580 /* U+0281 */ 0x0,
581 0x00,
582 0x00,
583 0x00,
584 0x00,
585 0x00,
586 0x00,
587 0x00,
588 0x00,
589 0x00,
590 0x00,
591 0x00,
592 0x00,
593 0x00,
594 /* U+028F */ 0x0,
595 0x00,
596 0x00,
597 0x00,
598 0x00,
599 0x00,
600 0x00,
601 0x00,
602 0x00,
603 0x00,
604 /* U+0299 */ 0x0,
605 0x00,
606 /* U+029B */ 0x0,
607 /* U+029C */ 0x0,
608 0x00,
609 0x00,
610 /* U+029F */ 0x0,
611 0x00,
612 0x00,
613 0x00,
614 0x00,
615 0x00,
616 0x00,
617 0x00,
618 0x00,
619 0x00,
620 0x00,
621 0x00,
622 0x00,
623 0x00,
624 0x00,
625 0x00,
626 0x00,
627 0x00,
628 0x00,
629 0x00,
630 0x00,
631 0x00,
632 0x00,
633 /* U+02B6 */ 0x0,
634 ;
635 636 static const unsigned short ucs_table_0386[] = {
637 /* U+0386 */ 0x03AC,
638 0x00,
639 /* U+0388 */ 0x03AD,
640 /* U+0389 */ 0x03AE,
641 /* U+038A */ 0x03AF,
642 /* U+038C */ 0x03CC,
643 /* U+038D */ 0x03CD,
644 /* U+038E */ 0x03CE,
645 /* U+038F */ 0x03CF,
646 /* U+0390 */ 0x03D0,
647 /* U+0391 */ 0x03D1,
648 /* U+0392 */ 0x03D2,
649 /* U+0393 */ 0x03D3,
650 /* U+0394 */ 0x03D4,
651 /* U+0395 */ 0x03D5,
652 /* U+0396 */ 0x03D6,
653 /* U+0397 */ 0x03D7,
654 /* U+0398 */ 0x03D8,
655 /* U+0399 */ 0x03D9,
656 /* U+039A */ 0x03DA,
657 /* U+039B */ 0x03DB,
658 /* U+039C */ 0x03DC,
659 /* U+039D */ 0x03DD,
660 /* U+039E */ 0x03DE,
662 /* U+039F */ 0x03BF,
663 /* U+03A0 */ 0x03C0,
664 /* U+03A1 */ 0x03C1,
665 0x00,
666 /* U+03A2 */ 0x03C2,
667 /* U+03A3 */ 0x03C3,
668 /* U+03A4 */ 0x03C4,
669 /* U+03A5 */ 0x03C5,
670 /* U+03A6 */ 0x03C6,
671 /* U+03A7 */ 0x03C7,
672 /* U+03A8 */ 0x03C8,
673 /* U+03A9 */ 0x03C9,
674 /* U+03AA */ 0x03CA,
675 /* U+03AB */ 0x03CB,
676 0x00,
677 0x00,
678 0x00,
679 0x00,
680 0x00,
681 0x00,
682 0x00,
683 0x00,
684 0x00,
685 0x00,
686 0x00,
687 0x00,
688 0x00,
689 0x00,
690 0x00,
691 0x00,
692 0x00,
693 0x00,
694 0x00,
695 0x00,
696 0x00,
697 0x00,
698 0x00,
699 0x00,
700 0x00,
701 0x00,
702 0x00,
703 0x00,
704 0x00,
705 0x00,
706 0x00,
707 0x00,
708 0x00,
709 0x00,
710 0x00,
711 0x00,
712 0x00,
713 /* U+03D2 */ 0x03D2,
714 /* U+03D3 */ 0x03D3,
715 /* U+03D4 */ 0x03D4,
716 0x00,
717 0x00,
718 0x00,
719 0x00,
720 0x00,
721 /* U+03DA */ 0x03DA,
722 0x00,
723 /* U+03DC */ 0x03DC,
724 0x00,
725 /* U+03DE */ 0x03DE,
726 0x00,
727 /* U+03E0 */ 0x03E0,
728 0x00,
729 /* U+03E2 */ 0x03E3,
730 0x00,
731 /* U+03E4 */ 0x03E5,
732 0x00,
733 /* U+03E6 */ 0x03E7,
734 0x00,
735 /* U+03E8 */ 0x03E9,
736 0x00,
737 /* U+03EA */ 0x03EB,
738 0x00,
739 /* U+03EC */ 0x03ED,
740 0x00,
741 /* U+03EE */ 0x03EF,
742 0x00,
743 0x00,
744 0x00,
745 0x00,
746 0x00,
747 0x00,
| 749 | 0x00, |
| 750 | 0x00, |
| 751 | 0x00, |
| 752 | 0x00, |
| 753 | 0x00, |
| 754 | 0x00, |
| 755 | 0x00, |
| 756 | 0x00, |
| 757 | 0x00, |
| 758 | 0x00, |
| 759 | 0x00, |
| 760 | /* U+0401 */ 0x0451, |
| 761 | /* U+0402 */ 0x0452, |
| 762 | /* U+0403 */ 0x0453, |
| 763 | /* U+0404 */ 0x0454, |
| 764 | /* U+0405 */ 0x0455, |
| 765 | /* U+0406 */ 0x0456, |
| 766 | /* U+0407 */ 0x0457, |
| 767 | /* U+0408 */ 0x0458, |
| 768 | /* U+0409 */ 0x0459, |
| 769 | /* U+040A */ 0x045A, |
| 770 | /* U+040B */ 0x045B, |
| 771 | /* U+040C */ 0x045C, |
| 772 | 0x00, |
| 773 | /* U+040D */ 0x045D, |
| 774 | /* U+040E */ 0x045E, |
| 775 | /* U+040F */ 0x045F, |
| 776 | /* U+0410 */ 0x0460, |
| 777 | /* U+0411 */ 0x0461, |
| 778 | /* U+0412 */ 0x0462, |
| 779 | /* U+0413 */ 0x0463, |
| 780 | /* U+0414 */ 0x0464, |
| 781 | /* U+0415 */ 0x0465, |
| 782 | /* U+0416 */ 0x0466, |
| 783 | /* U+0417 */ 0x0467, |
| 784 | /* U+0418 */ 0x0468, |
| 785 | /* U+0419 */ 0x0469, |
| 786 | /* U+041A */ 0x046A, |
| 787 | /* U+041B */ 0x046B, |
| 788 | /* U+041C */ 0x046C, |
| 789 | /* U+041D */ 0x046D, |
| 790 | /* U+041E */ 0x046E, |
| 791 | /* U+041F */ 0x046F, |
| 792 | /* U+0420 */ 0x0470, |
| 793 | /* U+0421 */ 0x0471, |
| 794 | /* U+0422 */ 0x0472, |
| 795 | /* U+0423 */ 0x0473, |
| 796 | /* U+0424 */ 0x0474, |
| 797 | /* U+0425 */ 0x0475, |
| 798 | /* U+0426 */ 0x0476, |
| 799 | /* U+0427 */ 0x0477, |
| 800 | /* U+0428 */ 0x0478, |
| 801 | /* U+0429 */ 0x0479, |
| 802 | /* U+042A */ 0x047A, |
| 803 | /* U+042B */ 0x047B, |
| 804 | /* U+042C */ 0x047C, |
| 805 | /* U+042D */ 0x047D, |
| 806 | /* U+042E */ 0x047E, |
| 807 | /* U+042F */ 0x047F, |
| 808 | 0x00, |
| 809 | 0x00, |
| 810 | 0x00, |
| 811 | 0x00, |
| 812 | 0x00, |
| 813 | 0x00, |
| 814 | 0x00, |
| 815 | 0x00, |
| 816 | 0x00, |
| 817 | 0x00, |
| 818 | 0x00, |
| 819 | 0x00, |
| 820 | 0x00, |
| 821 | 0x00, |
| 822 | 0x00, |
| 823 | 0x00, |
| 824 | 0x00, |
| 825 | 0x00, |
| 826 | 0x00, |
| 827 | 0x00, |
| 828 | 0x00, |
| 829 | 0x00, |
| 830 | 0x00, |
| 831 | 0x00, |
| 832 | 0x00, |
| 833 | 0x00, |
| 834 | 0x00, |
| 835 | 0x00, |
34.247 case.h

```c
836 0x00,
837 0x00,
838 0x00,
839 0x00,
840 0x00,
841 0x00,
842 0x00,
843 0x00,
844 0x00,
845 0x00,
846 0x00,
847 0x00,
848 0x00,
849 0x00,
850 0x00,
851 0x00,
852 0x00,
853 0x00,
854 0x00,
855 /* U+0460 */ 0x0461,
856 0x00,
857 /* U+0462 */ 0x0463,
858 0x00,
859 /* U+0464 */ 0x0465,
860 0x00,
861 /* U+0466 */ 0x0467,
862 0x00,
863 /* U+0468 */ 0x0469,
864 0x00,
865 /* U+046A */ 0x046B,
866 0x00,
867 /* U+046C */ 0x046D,
868 0x00,
869 /* U+046E */ 0x046F,
870 0x00,
871 /* U+0470 */ 0x0471,
872 0x00,
873 /* U+0472 */ 0x0473,
874 0x00,
875 /* U+0474 */ 0x0475,
876 0x00,
877 /* U+0476 */ 0x0477,
878 0x00,
879 /* U+0478 */ 0x0479,
880 0x00,
881 /* U+047A */ 0x047B,
882 0x00,
883 /* U+047C */ 0x047D,
884 0x00,
885 /* U+047E */ 0x047F,
886 0x00,
887 /* U+0480 */ 0x0481,
888 0x00,
889 0x00,
890 0x00,
891 0x00,
892 0x00,
893 0x00,
894 0x00,
895 0x00,
896 0x00,
897 0x00,
898 0x00,
899 0x00,
900 0x00,
901 0x00,
902 0x00,
903 /* U+0490 */ 0x0491,
904 0x00,
905 /* U+0492 */ 0x0493,
906 0x00,
907 /* U+0494 */ 0x0495,
908 0x00,
909 /* U+0496 */ 0x0497,
910 0x00,
911 /* U+0498 */ 0x0499,
912 0x00,
913 /* U+049A */ 0x049B,
914 0x00,
915 /* U+049C */ 0x049D,
916 0x00,
917 /* U+049E */ 0x049F,
918 0x00,
919 /* U+04A0 */ 0x04A1,
920 0x00,
921 /* U+04A2 */ 0x04A3,
922 0x00,
```

Generated by Doxygen
// U+04A4 */ 0x04A5,
924 0x00,
925 /* U+04A6 */ 0x04A7,
926 0x00,
927 /* U+04A8 */ 0x04A9,
928 0x00,
929 /* U+04AA */ 0x04AB,
930 0x00,
931 /* U+04AC */ 0x04AD,
932 0x00,
933 /* U+04AE */ 0x04AF,
934 0x00,
935 /* U+04B0 */ 0x04B1,
936 0x00,
937 /* U+04B2 */ 0x04B3,
938 0x00,
939 /* U+04B4 */ 0x04B5,
940 0x00,
941 /* U+04B6 */ 0x04B7,
942 0x00,
943 /* U+04B8 */ 0x04B9,
944 0x00,
945 /* U+04BA */ 0x04BB,
946 0x00,
947 /* U+04BC */ 0x04BD,
948 0x00,
949 /* U+04BE */ 0x04BF,
950 0x00,
951 /* U+04C0 */ 0x04C1,
952 0x00,
953 /* U+04C2 */ 0x04C3,
954 0x00,
955 /* U+04C4 */ 0x04C5,
956 0x00,
957 0x00,
958 /* U+04C7 */ 0x04C8,
959 0x00,
960 0x00,
961 /* U+04CB */ 0x04CC,
962 0x00,
963 /* U+04CD */ 0x04CE,
964 0x00,
965 0x00,
966 /* U+04D0 */ 0x04D1,
967 0x00,
968 /* U+04D2 */ 0x04D3,
969 0x00,
970 /* U+04D4 */ 0x04D5,
971 0x00,
972 /* U+04D6 */ 0x04D7,
973 0x00,
974 /* U+04D8 */ 0x04D9,
975 0x00,
976 /* U+04DA */ 0x04DB,
977 0x00,
978 /* U+04DC */ 0x04DD,
979 0x00,
980 /* U+04DE */ 0x04DF,
981 0x00,
982 /* U+04E0 */ 0x04E1,
983 0x00,
984 /* U+04E2 */ 0x04E3,
985 0x00,
986 /* U+04E4 */ 0x04E5,
987 0x00,
988 /* U+04E6 */ 0x04E7,
989 0x00,
990 /* U+04E8 */ 0x04E9,
991 0x00,
992 /* U+04EA */ 0x04EB,
993 0x00,
994 0x00,
995 0x00,
996 0x00,
997 /* U+04EE */ 0x04EF,
998 0x00,
999 /* U+04F0 */ 0x04F1,
1000 0x00,
1001 /* U+04F2 */ 0x04F3,
1002 0x00,
1003 /* U+04F4 */ 0x04F5,
1004 0x00,
1005 0x00,
1006 0x00,
1007 /* U+04F8 */ 0x04F9,
1010 0x00,
1011 0x00,
1012 0x00,
1013 0x00,
1014 0x00,
1015 0x00,
1016 0x00,
1017 0x00,
1018 0x00,
1019 0x00,
1020 0x00,
1021 0x00,
1022 0x00,
1023 0x00,
1024 0x00,
1025 0x00,
1026 0x00,
1027 0x00,
1028 0x00,
1029 0x00,
1030 0x00,
1031 0x00,
1032 0x00,
1033 0x00,
1034 0x00,
1035 0x00,
1036 0x00,
1037 0x00,
1038 0x00,
1039 0x00,
1040 0x00,
1041 0x00,
1042 0x00,
1043 0x00,
1044 0x00,
1045 0x00,
1046 0x00,
1047 0x00,
1048 0x00,
1049 0x00,
1050 0x00,
1051 0x00,
1052 0x00,
1053 0x00,
1054 0x00,
1055 0x00,
1056 0x00,
1057 0x00,
1058 0x00,
1059 0x00,
1060 0x00,
1061 0x00,
1062 0x00,
1063 0x00,
1064 /* U+0531 */ 0x0561,
1065 /* U+0532 */ 0x0562,
1066 /* U+0533 */ 0x0563,
1067 /* U+0534 */ 0x0564,
1068 /* U+0535 */ 0x0565,
1069 /* U+0536 */ 0x0566,
1070 /* U+0537 */ 0x0567,
1071 /* U+0538 */ 0x0568,
1072 /* U+0539 */ 0x0569,
1073 /* U+053A */ 0x056A,
1074 /* U+053B */ 0x056B,
1075 /* U+053C */ 0x056C,
1076 /* U+053D */ 0x056D,
1077 /* U+053E */ 0x056E,
1078 /* U+053F */ 0x056F,
1079 /* U+0540 */ 0x0570,
1080 /* U+0541 */ 0x0571,
1081 /* U+0542 */ 0x0572,
1082 /* U+0543 */ 0x0573,
1083 /* U+0544 */ 0x0574,
1084 /* U+0545 */ 0x0575,
1085 /* U+0546 */ 0x0576,
1086 /* U+0547 */ 0x0577,
1087 /* U+0548 */ 0x0578,
1088 /* U+0549 */ 0x0579,
1089 /* U+054A */ 0x057A,
1090 /* U+054B */ 0x057B,
1091 /* U+054C */ 0x057C,
1092 /* U+054D */ 0x057D,
1093 /* U+054E */ 0x057E,
1094 /* U+054F */ 0x057F,
1095 /* U+0550 */ 0x0580,
1096 /* U+0551 */ 0x0581,
1097 /* U+0552 */ 0x0582,
1098 /* U+0553 */ 0x0583,
1099 /* U+0554 */ 0x0584,
1100 /* U+0555 */ 0x0585,
1101 /* U+0556 */ 0x0586,
1102 /* U+0557 */ 0x0587,
1103 /* U+0558 */ 0x0588,
1104 /* U+0559 */ 0x0589,
1105 /* U+055A */ 0x058A,
1106 /* U+055B */ 0x058B,
1107 /* U+055C */ 0x058C,
1108 /* U+055D */ 0x058D,
1109 /* U+055E */ 0x058E,
1110 /* U+055F */ 0x058F,
1111 /* U+0560 */ 0x0590,
1112 /* U+0561 */ 0x0591,
1113 /* U+0562 */ 0x0592,
1114 /* U+0563 */ 0x0593,
1115 /* U+0564 */ 0x0594,
1116 /* U+0565 */ 0x0595,
1117 /* U+0566 */ 0x0596,
1118 /* U+0567 */ 0x0597,
1119 /* U+0568 */ 0x0598,
1120 /* U+0569 */ 0x0599,
1121 /* U+056A */ 0x059A,
1122 /* U+056B */ 0x059B,
1123 /* U+056C */ 0x059C,
1124 /* U+056D */ 0x059D,
1125 /* U+056E */ 0x059E,
1126 /* U+056F */ 0x059F,
1127 /* U+0570 */ 0x05A0,
1128 /* U+0571 */ 0x05A1,
1129 /* U+0572 */ 0x05A2,
1130 /* U+0573 */ 0x05A3,
1131 /* U+0574 */ 0x05A4,
1132 /* U+0575 */ 0x05A5,
1133 /* U+0576 */ 0x05A6,
1134 /* U+0577 */ 0x05A7,
1135 /* U+0578 */ 0x05A8,
1136 /* U+0579 */ 0x05A9,
1137 /* U+057A */ 0x05AA,
1138 /* U+057B */ 0x05AB,
1139 /* U+057C */ 0x05AC,
1140 /* U+057D */ 0x05AD,
1141 /* U+057E */ 0x05AE,
1142 /* U+057F */ 0x05AF,
1143 /* U+0580 */ 0x05B0,
1144 /* U+0581 */ 0x05B1,
1145 /* U+0582 */ 0x05B2,
1146 /* U+0583 */ 0x05B3,
1147 /* U+0584 */ 0x05B4,
1148 /* U+0585 */ 0x05B5,
1149 /* U+0586 */ 0x05B6,
1150 /* U+0587 */ 0x05B7,
1151 /* U+0588 */ 0x05B8,
1152 /* U+0589 */ 0x05B9,
1153 /* U+058A */ 0x05BA,
1154 /* U+058B */ 0x05BB,
1155 /* U+058C */ 0x05BC,
1156 /* U+058D */ 0x05BD,
1157 /* U+058E */ 0x05BE,
1158 /* U+058F */ 0x05BF,
1159 /* U+0590 */ 0x05C0,
1160 /* U+0591 */ 0x05C1,
1161 /* U+0592 */ 0x05C2,
1162 /* U+0593 */ 0x05C3,
1163 /* U+0594 */ 0x05C5,
1164 /* U+0595 */ 0x05C6,
1165 /* U+0596 */ 0x05C7,
1166 /* U+0597 */ 0x05C8,
1167 /* U+0598 */ 0x05C9,
1168 /* U+0599 */ 0x05CA,
1169 /* U+059A */ 0x05CB,
1170 /* U+059B */ 0x05CC,
1171 /* U+059C */ 0x05CD,
1172 /* U+059D */ 0x05CE,
1173 /* U+059E */ 0x05CF,
1174 /* U+059F */ 0x05D0,
1175 /* U+05A0 */ 0x05D1,
1176 /* U+05A1 */ 0x05D2,
1177 /* U+05A2 */ 0x05D3,
1178 /* U+05A3 */ 0x05D4,
1179 /* U+05A4 */ 0x05D5,
1180 /* U+05A5 */ 0x05D6,
1181 /* U+05A6 */ 0x05D7,
1182 /* U+05A7 */ 0x05D8,
1183 /* U+05A8 */ 0x05D9,
/* U+1E26 */ 0x1E27, 0x00,
/* U+1E28 */ 0x1E29, 0x00,
/* U+1E2A */ 0x1E2B, 0x00,
/* U+1E2C */ 0x1E2D, 0x00,
/* U+1E2E */ 0x1E2F, 0x00,
/* U+1E30 */ 0x1E31, 0x00,
/* U+1E32 */ 0x1E33, 0x00,
/* U+1E34 */ 0x1E35, 0x00,
/* U+1E36 */ 0x1E37, 0x00,
/* U+1E38 */ 0x1E39, 0x00,
/* U+1E3A */ 0x1E3B, 0x00,
/* U+1E3C */ 0x1E3D, 0x00,
/* U+1E3E */ 0x1E3F, 0x00,
/* U+1E40 */ 0x1E41, 0x00,
/* U+1E42 */ 0x1E43, 0x00,
/* U+1E44 */ 0x1E45, 0x00,
/* U+1E46 */ 0x1E47, 0x00,
/* U+1E48 */ 0x1E49, 0x00,
/* U+1E4A */ 0x1E4B, 0x00,
/* U+1E4C */ 0x1E4D, 0x00,
/* U+1E4E */ 0x1E4F, 0x00,
/* U+1E50 */ 0x1E51, 0x00,
/* U+1E52 */ 0x1E53, 0x00,
/* U+1E54 */ 0x1E55, 0x00,
/* U+1E56 */ 0x1E57, 0x00,
/* U+1E58 */ 0x1E59, 0x00,
/* U+1E5A */ 0x1E5B, 0x00,
/* U+1E5C */ 0x1E5D, 0x00,
/* U+1E5E */ 0x1E5F, 0x00,
/* U+1E60 */ 0x1E61, 0x00,
/* U+1E62 */ 0x1E63, 0x00,
/* U+1E64 */ 0x1E65, 0x00,
/* U+1E66 */ 0x1E67, 0x00,
/* U+1E68 */ 0x1E69, 0x00,
/* U+1E6A */ 0x1E6B, 0x00,
/* U+1E6C */ 0x1E6D, 0x00,
/* U+1E6E */ 0x1E6F, 0x00,
/* U+1E70 */ 0x1E71, 0x00,
/* U+1E72 */ 0x1E73, 0x00,
1271 0x00,
1272 /* U+1E7E */ 0x1E7E,
1273 0x00,
1274 /* U+1E80 */ 0x1E80,
1275 0x00,
1276 /* U+1E82 */ 0x1E82,
1277 0x00,
1278 /* U+1E84 */ 0x1E84,
1279 0x00,
1280 /* U+1E86 */ 0x1E86,
1281 0x00,
1282 /* U+1E88 */ 0x1E88,
1283 0x00,
1284 /* U+1E8A */ 0x1E8A,
1285 0x00,
1286 /* U+1E8C */ 0x1E8C,
1287 0x00,
1288 /* U+1E8E */ 0x1E8E,
1289 0x00,
1290 /* U+1E90 */ 0x1E90,
1291 0x00,
1292 /* U+1E92 */ 0x1E92,
1293 0x00,
1294 /* U+1E94 */ 0x1E94,
1295 0x00,
1296 0x00,
1297 0x00,
1298 0x00,
1299 0x00,
1300 0x00,
1301 0x00,
1302 0x00,
1303 0x00,
1304 0x00,
1305 0x00,
1306 /* U+1EA0 */ 0x1EA0,
1307 0x00,
1308 /* U+1EA2 */ 0x1EA2,
1309 0x00,
1310 /* U+1EA4 */ 0x1EA4,
1311 0x00,
1312 /* U+1EA6 */ 0x1EA6,
1313 0x00,
1314 /* U+1EA8 */ 0x1EA8,
1315 0x00,
1316 /* U+1EAA */ 0x1EAA,
1317 0x00,
1318 /* U+1EAC */ 0x1EAC,
1319 0x00,
1320 /* U+1EAE */ 0x1EAE,
1321 0x00,
1322 /* U+1EB0 */ 0x1EB0,
1323 0x00,
1324 /* U+1EB2 */ 0x1EB2,
1325 0x00,
1326 /* U+1EB4 */ 0x1EB4,
1327 0x00,
1328 /* U+1EB6 */ 0x1EB6,
1329 0x00,
1330 /* U+1EB8 */ 0x1EB8,
1331 0x00,
1332 /* U+1EBA */ 0x1EBA,
1333 0x00,
1334 /* U+1EBE */ 0x1EBE,
1335 0x00,
1336 /* U+1EFC */ 0x1EFC,
1337 0x00,
1338 /* U+1EC0 */ 0x1EC0,
1339 0x00,
1340 /* U+1EC2 */ 0x1EC2,
1341 0x00,
1342 /* U+1EC4 */ 0x1EC4,
1343 0x00,
1344 /* U+1EC6 */ 0x1EC6,
1345 0x00,
1346 /* U+1EC8 */ 0x1EC8,
1347 0x00,
1348 /* U+1ECA */ 0x1ECA,
1349 0x00,
1350 /* U+1EC2 */ 0x1EC2,
1358 /* U+1ED4 */ 0x1ED5,
1359 0x00,
1360 /* U+1ED6 */ 0x1ED7,
1361 0x00,
1362 /* U+1ED8 */ 0x1ED9,
1363 0x00,
1364 /* U+1EDA */ 0x1EDB,
1365 0x00,
1366 /* U+1EDC */ 0x1EDD,
1367 0x00,
1368 /* U+1EDE */ 0x1EE1,
1369 0x00,
1370 /* U+1EE2 */ 0x1EE3,
1371 0x00,
1372 /* U+1EE4 */ 0x1EE5,
1373 0x00,
1374 /* U+1EE6 */ 0x1EE7,
1375 0x00,
1376 /* U+1EE8 */ 0x1EE9,
1377 0x00,
1378 /* U+1EEA */ 0x1EEB,
1379 0x00,
1380 /* U+1EEC */ 0x1EEF,
1381 0x00,
1382 /* U+1EEE */ 0x1EEF,
1383 0x00,
1384 /* U+1EF0 */ 0x1EF1,
1385 0x00,
1386 /* U+1EF2 */ 0x1EF3,
1387 0x00,
1388 /* U+1EF4 */ 0x1EF5,
1389 0x00,
1390 /* U+1EF6 */ 0x1EF7,
1391 0x00,
1392 /* U+1EF8 */ 0x1EF9,
1393 0x00,
1394 /* U+1F08 */ 0x1F00,
1395 0x00,
1396 0x00,
1397 0x00,
1398 0x00,
1399 0x00,
1400 0x00,
1401 0x00,
1402 0x00,
1403 0x00,
1404 0x00,
1405 0x00,
1406 0x00,
1407 0x00,
1408 0x00,
1409 0x00,
1410 /* U+1F0A */ 0x1F0B,
1411 0x00,
1412 /* U+1F0C */ 0x1F0D,
1413 0x00,
1414 /* U+1F0E */ 0x1F0F,
1415 0x00,
1416 /* U+1F10 */ 0x1F11,
1417 0x00,
1418 /* U+1F12 */ 0x1F13,
1419 0x00,
1420 0x00,
1421 0x00,
1422 0x00,
1423 0x00,
1424 0x00,
1425 0x00,
1426 /* U+1F18 */ 0x1F19,
1427 0x00,
1428 /* U+1F1A */ 0x1F1B,
1429 0x00,
1430 /* U+1F1C */ 0x1F1D,
1431 0x00,
1432 0x00,
1433 0x00,
1434 0x00,
1435 0x00,
1436 0x00,
1437 0x00,
1438 0x00,
1439 0x00,
1440 0x00,
1441 0x00,
1442 /* U+1F28 */ 0x1F29,
1443 0x00,
1444 /* U+1F2A */ 0x1F2B,
1445 /* U+1F2B */ 0x1F23,
1446 /* U+1F2C */ 0x1F24,
1447 /* U+1F2D */ 0x1F25,
1448 /* U+1F2E */ 0x1F26,
1449 /* U+1F2F */ 0x1F27,
1450 0x00,
1451 0x00,
1452 0x00,
1453 0x00,
1454 0x00,
1455 0x00,
1456 0x00,
1457 0x00,
1458 /* U+1F30 */ 0x1F33,
1459 /* U+1F31 */ 0x1F34,
1460 /* U+1F32 */ 0x1F35,
1461 /* U+1F33 */ 0x1F36,
1462 /* U+1F34 */ 0x1F37,
1463 /* U+1F35 */ 0x1F38,
1464 /* U+1F36 */ 0x1F39,
1465 /* U+1F37 */ 0x1F3A,
1466 0x00,
1467 0x00,
1468 0x00,
1469 0x00,
1470 0x00,
1471 0x00,
1472 0x00,
1473 0x00,
1474 /* U+1F40 */ 0x1F41,
1475 /* U+1F41 */ 0x1F42,
1476 /* U+1F42 */ 0x1F43,
1477 /* U+1F43 */ 0x1F44,
1478 /* U+1F44 */ 0x1F45,
1479 /* U+1F45 */ 0x1F46,
1480 0x00,
1481 0x00,
1482 0x00,
1483 0x00,
1484 0x00,
1485 0x00,
1486 0x00,
1487 0x00,
1488 0x00,
1489 0x00,
1490 0x00,
1491 /* U+1F51 */ 0x1F52,
1492 0x00,
1493 /* U+1F52 */ 0x1F53,
1494 0x00,
1495 /* U+1F53 */ 0x1F54,
1496 0x00,
1497 /* U+1F54 */ 0x1F55,
1498 0x00,
1499 0x00,
1500 0x00,
1501 0x00,
1502 0x00,
1503 0x00,
1504 0x00,
1505 0x00,
1506 /* U+1F60 */ 0x1F61,
1507 /* U+1F61 */ 0x1F62,
1508 /* U+1F62 */ 0x1F63,
1509 /* U+1F63 */ 0x1F64,
1510 /* U+1F64 */ 0x1F65,
1511 /* U+1F65 */ 0x1F66,
1512 /* U+1F66 */ 0x1F67,
1513 /* U+1F67 */ 0x1F68,
1532 0x00,
1533 0x00,
1534 0x00,
1535 0x00,
1536 0x00,
1537 0x00,
1538 /* U+1F88 */ 0x0,
1539 /* U+1F89 */ 0x0,
1540 /* U+1F8A */ 0x0,
1541 /* U+1F8B */ 0x0,
1542 /* U+1F8C */ 0x0,
1543 /* U+1F8D */ 0x0,
1544 /* U+1F8E */ 0x0,
1545 /* U+1F8F */ 0x0,
1546 0x00,
1547 0x00,
1548 0x00,
1549 0x00,
1550 0x00,
1551 0x00,
1552 0x00,
1553 0x00,
1554 /* U+1F98 */ 0x0,
1555 /* U+1F99 */ 0x0,
1556 /* U+1F9A */ 0x0,
1557 /* U+1F9B */ 0x0,
1558 /* U+1F9C */ 0x0,
1559 /* U+1F9D */ 0x0,
1560 /* U+1F9E */ 0x0,
1561 /* U+1F9F */ 0x0,
1562 0x00,
1563 0x00,
1564 0x00,
1565 0x00,
1566 0x00,
1567 0x00,
1568 0x00,
1569 0x00,
1570 /* U+1FA8 */ 0x0,
1571 /* U+1FA9 */ 0x0,
1572 /* U+1FAA */ 0x0,
1573 /* U+1FAB */ 0x0,
1574 /* U+1FAC */ 0x0,
1575 /* U+1FAD */ 0x0,
1576 /* U+1FAE */ 0x0,
1577 /* U+1FAF */ 0x0,
1578 0x00,
1579 0x00,
1580 0x00,
1581 0x00,
1582 0x00,
1583 0x00,
1584 0x00,
1585 0x00,
1586 /* U+1FB8 */ 0x1FB0,
1587 /* U+1FB9 */ 0x1FB1,
1588 /* U+1FBA */ 0x1F70,
1589 /* U+1FBB */ 0x1F71,
1590 /* U+1FBC */ 0x0,
1591 0x00,
1592 0x00,
1593 0x00,
1594 0x00,
1595 0x00,
1596 0x00,
1597 0x00,
1598 0x00,
1599 0x00,
1600 0x00,
1601 0x00,
1602 /* U+1FC8 */ 0x1F72,
1603 /* U+1FC9 */ 0x1F73,
1604 /* U+1FCA */ 0x1F74,
1605 /* U+1FCB */ 0x1F75,
1606 /* U+1FCC */ 0x0,
1607 0x00,
1608 0x00,
1609 0x00,
1610 0x00,
1611 0x00,
1612 0x00,
1613 0x00,
1614 0x00,
1615 0x00,
1616 0x00,
1617 0x00,
1618 /* U+1FDB */ 0x1FD0,
1619 /* U+1FD9 */ 0x1FD1,
1620 /* U+1FDA */ 0x1F76,
1621 /* U+1FDB */ 0x1F77,
1622 0x00,
1623 0x00,
1624 0x00,
1625 0x00,
1626 0x00,
1627 0x00,
1628 0x00,
1629 0x00,
1630 0x00,
1631 0x00,
1632 0x00,
1633 0x00,
1634 /* U+1FEB */ 0x1F78,
1635 /* U+1FEC */ 0x1F7B,
1636 /* U+1FE8 */ 0x1FE0,
1637 /* U+1FE9 */ 0x1FE1,
1638 /* U+1FEA */ 0x1F7A,
1639 /* U+1FF8 */ 0x1F78,
1640 /* U+1FF9 */ 0x1F79,
1641 /* U+1FFA */ 0x1F7C,
1642 /* U+1FFB */ 0x1F7D,
1643 /* U+1FFC */ 0x0,
1644 /* U+1FEB */ 0x0,
1645 0x00,
1646 0x00,
1647 0x00,
1648 0x00,
1649 0x00,
1650 /* U+1FEC */ 0x1FE5,
1651 /* U+1FED */ 0x0,
1652 /* U+1FEE */ 0x0,
1653 /* U+1FF9 */ 0x1FE1,
1654 /* U+1FFE */ 0x0,
1655 */
1656 static const unsigned short ucs_table_2102[] = {
1657 / U+2102 */ 0x0,
1658 / U+2108 */ 0x0,
1659 0x00,
1660 0x00,
1661 0x00,
1662 0x00,
1663 0x00,
1664 0x00,
1665 0x00,
1666 0x00,
1667 /* U+210B */ 0x0,
1668 /* U+210C */ 0x0,
1669 /* U+2110 */ 0x0,
1670 /* U+2111 */ 0x0,
1671 0x00,
1672 /* U+211E */ 0x0,
1673 /* U+211F */ 0x0,
1674 /* U+2112 */ 0x0,
1675 0x00,
1676 0x00,
1677 /* U+2115 */ 0x0,
1678 0x00,
1679 0x00,
1680 /* U+211B */ 0x0,
1681 /* U+211C */ 0x0,
1682 /* U+211D */ 0x0,
1683 /* U+211E */ 0x0,
1684 /* U+211F */ 0x0,
1685 /* U+2122 */ 0x0,
1686 0x00,
1687 0x00,
1688 0x00,
1689 0x00,
1690 0x00,
1691 0x00,
1692 /* U+2124 */ 0x0,
1693 0x00,
1694 0x00,
1695 0x00,
1696 /* U+2128 */ 0x0,
1697 0x00,
1698 0x00,
1699 0x00,
1700 /* U+212C */ 0x0,
1701 /* U+212D */ 0x0,
1702 0x00,
1703 0x00,
1704 /* U+2130 */ 0x0,
1705 /* U+2131 */ 0x0,
1706 */
1707 Generated by Doxygen
```c
static const unsigned short ucs_table_24B6[] = {
    /* U+24B6 */ 0x24D0,
    /* U+24B7 */ 0x24D1,
    /* U+24B8 */ 0x24D2,
    /* U+24B9 */ 0x24D3,
    /* U+24BA */ 0x24D4,
    /* U+24BB */ 0x24D5,
    /* U+24BC */ 0x24D6,
    /* U+24BD */ 0x24D7,
    /* U+24BE */ 0x24D8,
    /* U+24BF */ 0x24D9,
    /* U+24C0 */ 0x24DA,
    /* U+24C1 */ 0x24DB,
    /* U+24C2 */ 0x24DC,
    /* U+24C3 */ 0x24DD,
    /* U+24C4 */ 0x24DE,
    /* U+24C5 */ 0x24DF,
    /* U+24C6 */ 0x24E0,
    /* U+24C7 */ 0x24E1,
    /* U+24C8 */ 0x24E2,
    /* U+24C9 */ 0x24E3,
    /* U+24CA */ 0x24E4,
    /* U+24CB */ 0x24E5,
    /* U+24CC */ 0x24E6,
    /* U+24CD */ 0x24E7,
    /* U+24CE */ 0x24E8,
    /* U+24CF */ 0x24E9,
    /* U+24D0 */ 0x24E0,
    /* U+24D1 */ 0x24E1,
    /* U+24D2 */ 0x24E2,
    /* U+24D3 */ 0x24E3,
    /* U+24D4 */ 0x24E4,
    /* U+24D5 */ 0x24E5,
    /* U+24D6 */ 0x24E6,
    /* U+24D7 */ 0x24E7,
    /* U+24D8 */ 0x24E8,
    /* U+24D9 */ 0x24E9,
    /* U+24DA */ 0x24EA,
    /* U+24DB */ 0x24EB,
    /* U+24DC */ 0x24ED,
    /* U+24DD */ 0x24EE,
    /* U+24DE */ 0x24EF,
    /* U+24DF */ 0x24F0,
    /* U+24E0 */ 0x24F1,
    /* U+24E1 */ 0x24F2,
    /* U+24E2 */ 0x24F3,
    /* U+24E3 */ 0x24F4,
    /* U+24E4 */ 0x24F5,
    /* U+24E5 */ 0x24F6,
    /* U+24E6 */ 0x24F7,
    /* U+24E7 */ 0x24F8,
    /* U+24E8 */ 0x24F9,
    /* U+24E9 */ 0x24FA,
    /* U+24EA */ 0x24FB,
    /* U+24EB */ 0x24FC,
    /* U+24ED */ 0x24FD,
    /* U+24EE */ 0x24FE,
    /* U+24EF */ 0x24FF,
    /* U+24F0 */ 0x0,
};
```

```c
static const unsigned short ucs_table_33CE[] = {
    /* U+33CE */ 0x0,
};
```

```c
static const unsigned short ucs_table_FF21[] = {
    /* U+FF21 */ 0xFF41,
    /* U+FF22 */ 0xFF42,
    /* U+FF23 */ 0xFF43,
    /* U+FF24 */ 0xFF44,
    /* U+FF25 */ 0xFF45,
    /* U+FF26 */ 0xFF46,
    /* U+FF27 */ 0xFF47,
    /* U+FF28 */ 0xFF48,
    /* U+FF29 */ 0xFF49,
    /* U+FF2A */ 0xFF4A,
    /* U+FF2B */ 0xFF4B,
    /* U+FF2C */ 0xFF4C,
    /* U+FF2D */ 0xFF4D,
    /* U+FF2E */ 0xFF4E,
    /* U+FF2F */ 0xFF4F,
    /* U+FF30 */ 0xFF50,
    /* U+FF31 */ 0xFF51,
    /* U+FF32 */ 0xFF52,
    /* U+FF33 */ 0xFF53,
    /* U+FF34 */ 0xFF54,
    /* U+FF35 */ 0xFF55,
    /* U+FF36 */ 0xFF56,
    /* U+FF37 */ 0xFF57,
    /* U+FF38 */ 0xFF58,
    /* U+FF39 */ 0xFF59,
    /* U+FF3A */ 0xFF5A,
};
```

```c
1748 dingbats_.h
```

```c
static const char unicode_to_dingbats_1b_0020[] = {
    /* U+0020 */ 0x20,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
    0x00,
};
```

34.248 dingbats_.h

```c
1 /* dingbats */
2
3 static const char unicode_to_dingbats_1b_0020[] = {
4    /* U+0020 */ 0x20,
5    0x00,
6    0x00,
7    0x00,
8    0x00,
9    0x00,
10   0x00,
11   0x00,
12   0x00,
13   0x00,
14   0x00,
15   0x00,
16   0x00,
17   0x00,
```
105 0x00,
106 0x00,
107 0x00,
108 0x00,
109 0x00,
110 0x00,
111 0x00,
112 0x00,
113 0x00,
114 0x00,
115 0x00,
116 0x00,
117 0x00,
118 0x00,
119 0x00,
120 0x00,
121 0x00,
122 0x00,
123 0x00,
124 0x00,
125 0x00,
126 0x00,
127 0x00,
128 0x00,
129 0x00,
130 0x00,
131 0x00,
132 /* U+00A0 */ 0x20,
133 }
134
135 static const char unicode_to_dingbats_1b_2192[] = {
136 /* U+2192 */ (char)0xD5,
137 0x00,
138 /* U+2194 */ (char)0xD6,
139 /* U+2195 */ (char)0xD7,
140 }
141
142 static const char unicode_to_dingbats_1b_2460[] = {
143 /* U+2460 */ (char)0xAC,
144 /* U+2461 */ (char)0xAD,
145 /* U+2462 */ (char)0xAE,
146 /* U+2463 */ (char)0xAF,
147 /* U+2464 */ (char)0xB0,
148 /* U+2465 */ (char)0xB1,
149 /* U+2466 */ (char)0xB2,
150 /* U+2467 */ (char)0xB3,
151 /* U+2468 */ (char)0xB4,
152 /* U+2469 */ (char)0xB5,
153 }
154
155 static const char unicode_to_dingbats_1b_25A0[] = {
156 /* U+25A0 */ 0x6E,
157 0x00,
158 0x00,
159 0x00,
160 0x00,
161 0x00,
162 0x00,
163 0x00,
164 0x00,
165 0x00,
166 0x00,
167 0x00,
168 0x00,
169 0x00,
170 0x00,
171 0x00,
172 0x00,
173 0x00,
174 /* U+25B2 */ 0x73,
175 0x00,
176 0x00,
177 0x00,
178 0x00,
179 0x00,
180 0x00,
181 0x00,
182 0x00,
183 0x00,
184 /* U+25B4 */ 0x74,
185 0x00,
186 0x00,
187 0x00,
188 0x00,
189 0x00,
190 0x00,
191 0x00,

static const char unicode_to_dingbats_1b_2701[] = {
    /* U+2701 */ 0x21,
    /* U+2702 */ 0x22,
    /* U+2703 */ 0x23,
    /* U+2704 */ 0x24,
    /* U+2705 */ 0x25,
    /* U+2706 */ 0x26,
    /* U+2707 */ 0x27,
    /* U+2708 */ 0x28,
    /* U+2709 */ 0x29,
    /* U+270a */ 0x2a,
    /* U+270b */ 0x2b,
    /* U+270c */ 0x2c,
    /* U+270d */ 0x2d,
    /* U+270e */ 0x2e,
    /* U+270f */ 0x2f,
    /* U+2710 */ 0x30,
    /* U+2711 */ 0x31,
    /* U+2712 */ 0x32,
    /* U+2713 */ 0x33,
    /* U+2714 */ 0x34,
    /* U+2715 */ 0x35,
    /* U+2716 */ 0x36,
    /* U+2717 */ 0x37,
    /* U+2718 */ 0x38,
    /* U+2719 */ 0x39,
    /* U+271a */ 0x3a,
    /* U+271b */ 0x3b,
    /* U+271c */ 0x3c,
    /* U+271d */ 0x3d,
    /* U+271e */ 0x3e,
    /* U+271f */ 0x3f,
};
34.248 dingbats.h

453 0x00,
454 /* U+2761 */ (char)0xA1,
455 /* U+2762 */ (char)0xA2,
456 /* U+2763 */ (char)0xA3,
457 /* U+2764 */ (char)0xA4,
458 /* U+2765 */ (char)0xA5,
459 /* U+2766 */ (char)0xA6,
460 /* U+2767 */ (char)0xA7,
461 0x00,
462 0x00,
463 0x00,
464 0x00,
465 0x00,
466 0x00,
467 0x00,
468 0x00,
469 0x00,
470 0x00,
471 0x00,
472 0x00,
473 0x00,
474 0x00,
475 /* U+2776 */ (char)0xB6,
476 /* U+2777 */ (char)0xB7,
477 /* U+2778 */ (char)0xB8,
478 /* U+2779 */ (char)0xB9,
479 /* U+277A */ (char)0xBA,
480 /* U+277B */ (char)0xBB,
481 /* U+277C */ (char)0xBC,
482 /* U+277D */ (char)0xBD,
483 /* U+277E */ (char)0xBE,
484 /* U+277F */ (char)0xBF,
485 /* U+2780 */ (char)0xC0,
486 /* U+2781 */ (char)0xC1,
487 /* U+2782 */ (char)0xC2,
488 /* U+2783 */ (char)0xC3,
489 /* U+2784 */ (char)0xC4,
490 /* U+2785 */ (char)0xC5,
491 /* U+2786 */ (char)0xC6,
492 /* U+2787 */ (char)0xC7,
493 /* U+2788 */ (char)0xC8,
494 /* U+2789 */ (char)0xC9,
495 /* U+278A */ (char)0xCA,
496 /* U+278B */ (char)0xCB,
497 /* U+278C */ (char)0xCC,
498 /* U+278D */ (char)0xCD,
499 /* U+278E */ (char)0xCE,
500 /* U+278F */ (char)0xCF,
501 /* U+2790 */ (char)0xD0,
502 /* U+2791 */ (char)0xD1,
503 /* U+2792 */ (char)0xD2,
504 /* U+2793 */ (char)0xD3,
505 /* U+2794 */ (char)0xD4,
506 0x00,
507 0x00,
508 0x00,
509 /* U+2798 */ (char)0xD8,
510 /* U+2799 */ (char)0xD9,
511 /* U+279A */ (char)0xDA,
512 /* U+279B */ (char)0xDB,
513 /* U+279C */ (char)0xDC,
514 /* U+279D */ (char)0xDD,
515 /* U+279E */ (char)0xDE,
516 /* U+279F */ (char)0xDF,
517 /* U+27A0 */ (char)0xE0,
518 /* U+27A1 */ (char)0xE1,
519 /* U+27A2 */ (char)0xE2,
520 /* U+27A3 */ (char)0xE3,
521 /* U+27A4 */ (char)0xE4,
522 /* U+27A5 */ (char)0xE5,
523 /* U+27A6 */ (char)0xE6,
524 /* U+27A7 */ (char)0xE7,
525 /* U+27A8 */ (char)0xE8,
526 /* U+27A9 */ (char)0xE9,
527 /* U+27AA */ (char)0xEA,
528 /* U+27AB */ (char)0xEB,
529 /* U+27AC */ (char)0xEC,
530 /* U+27AD */ (char)0xED,
531 /* U+27AE */ (char)0xEE,
532 /* U+27AF */ (char)0xEF,
533 0x00,
534 /* U+27B0 */ (char)0xF0,
535 /* U+27B1 */ (char)0xF1,
536 /* U+27B2 */ (char)0xF2,
537 /* U+27B3 */ (char)0xF3,
538 /* U+27B4 */ (char)0xF4,
539 /* U+27B5 */ (char)0xF5,
static const char unicode_to_dingbats_1b_F8D7[] = {
    /* U+F8D7 */ (char)0x80,
    /* U+F8D8 */ (char)0x81,
    /* U+F8D9 */ (char)0x82,
    /* U+F8DA */ (char)0x83,
    /* U+F8DB */ (char)0x84,
    /* U+F8DC */ (char)0x85,
    /* U+F8DD */ (char)0x86,
    /* U+F8DE */ (char)0x87,
    /* U+F8DF */ (char)0x88,
    /* U+F8E0 */ (char)0x89,
    /* U+F8E1 */ (char)0x8A,
    /* U+F8E2 */ (char)0x8B,
    /* U+F8E3 */ (char)0x8C,
    /* U+F8E4 */ (char)0x8D,
};

#define _UCS_TABLE_0300(x) (x)

static const unsigned short ucs_table_0300[] = {
    /* U+0300 */ 0x0060,
    /* U+0301 */ 0x00B4,
    /* U+0302 */ 0x005E,
    /* U+0303 */ 0x02DC,
    /* U+0304 */ 0x00AF,
    /* U+0305 */ 0x203E,
    /* U+0306 */ 0x02D8,
    /* U+0307 */ 0x02D9,
    /* U+0308 */ 0x00A8,
    /* U+0309 */ 0x0309,
    /* U+030A */ 0x02DA,
    /* U+030B */ 0x02DD,
    /* U+030C */ 0x030C,
    /* U+030D */ 0x030D,
    /* U+030E */ 0x030E,
    /* U+030F */ 0x030F,
    /* U+0310 */ 0x0310,
    /* U+0311 */ 0x0311,
    /* U+0312 */ 0x0312,
    /* U+0313 */ 0x1FBD,
    /* U+0314 */ 0x1FFE,
    /* U+0315 */ 0x0315,
    /* U+0316 */ 0x0316,
    /* U+0317 */ 0x0317,
    /* U+0318 */ 0x0318,
    /* U+0319 */ 0x0319,
    /* U+031A */ 0x031A,
    /* U+031B */ 0x031B,
    /* U+031C */ 0x031C,
    /* U+031D */ 0x00B8,
    /* U+031E */ 0x02DB,
    /* U+031F */ 0x031F,
    /* U+0320 */ 0x0320,
    /* U+0321 */ 0x0321,
    /* U+0322 */ 0x0322,
    /* U+0323 */ 0x0323,
    /* U+0324 */ 0x0324,
    /* U+0325 */ 0x0325,
    /* U+0326 */ 0x0326,
    /* U+0327 */ 0x00B8,
    /* U+0328 */ 0x0328,
    /* U+0329 */ 0x0329,
    /* U+032A */ 0x032A,
    /* U+032B */ 0x032B,
    /* U+032C */ 0x032C,
    /* U+032D */ 0x032D,
    /* U+032E */ 0x032E,
    /* U+032F */ 0x032F,
    /* U+0330 */ 0x0330,
    /* U+0331 */ 0x0331,
    /* U+0332 */ 0x0332,
    /* U+0333 */ 0x0333,
    /* U+0334 */ 0x0334,
57 /* U+0335 */ 0x0335,
58 /* U+0336 */ 0x0336,
59 /* U+0337 */ 0x0337,
60 /* U+0338 */ 0x0338,
61 /* U+0339 */ 0x0339,
62 /* U+033a */ 0x033a,
63 /* U+033b */ 0x033b,
64 /* U+033c */ 0x033c,
65 /* U+033d */ 0x033d,
66 /* U+033e */ 0x033e,
67 /* U+033f */ 0x033f,
68 /* U+0340 */ 0x0340,
69 /* U+0341 */ 0x0341,
70 /* U+0342 */ 0x0342,
71 /* U+0343 */ 0x0343,
72 /* U+0344 */ 0x0344,
73 /* U+0345 */ 0x0345,
74 0x00,
75 0x00,
76 0x00,
77 0x00,
78 0x00,
79 0x00,
80 0x00,
81 0x00,
82 0x00,
83 0x00,
84 0x00,
85 0x00,
86 0x00,
87 0x00,
88 0x00,
89 0x00,
90 0x00,
91 0x00,
92 0x00,
93 0x00,
94 0x00,
95 0x00,
96 0x00,
97 0x00,
98 0x00,
99 0x00,
100 /* U+0360 */ 0x0360,
101 /* U+0361 */ 0x0361,
102 );
103

104 static const unsigned short ucs_table_0483[] = {
105 /* U+0483 */ 0x0483,
106 /* U+0484 */ 0x0484,
107 /* U+0485 */ 0x0485,
108 /* U+0486 */ 0x0486,
109 );
110

111 static const unsigned short ucs_table_0591[] = {
112 /* U+0591 */ 0x0591,
113 /* U+0592 */ 0x0592,
114 /* U+0593 */ 0x0593,
115 /* U+0594 */ 0x0594,
116 /* U+0595 */ 0x0595,
117 /* U+0596 */ 0x0596,
118 /* U+0597 */ 0x0597,
119 /* U+0598 */ 0x0598,
120 /* U+0599 */ 0x0599,
121 /* U+059a */ 0x059a,
122 /* U+059b */ 0x059b,
123 /* U+059c */ 0x059c,
124 /* U+059d */ 0x059d,
125 /* U+059e */ 0x059e,
126 /* U+059f */ 0x059f,
127 /* U+05a0 */ 0x05a0,
128 /* U+05a1 */ 0x05a1,
129 0x00,
130 /* U+05a2 */ 0x05a2,
131 /* U+05a3 */ 0x05a3,
132 /* U+05a4 */ 0x05a4,
133 /* U+05a5 */ 0x05a5,
134 /* U+05a6 */ 0x05a6,
135 /* U+05a7 */ 0x05a7,
136 /* U+05a8 */ 0x05a8,
137 /* U+05a9 */ 0x05a9,
138 /* U+05aa */ 0x05aa,
139 /* U+05ab */ 0x05ab,
140 /* U+05ac */ 0x05ac,
141 /* U+05ad */ 0x05ad,
142 /* U+05ae */ 0x05ae,
143 /* U+05af */ 0x05af,
144 /* U+05B1 */ 0x05B1,
145 /* U+05B2 */ 0x05B2,
146 /* U+05B3 */ 0x05B3,
147 /* U+05B4 */ 0x05B4,
148 /* U+05B5 */ 0x05B5,
149 /* U+05B6 */ 0x05B6,
150 /* U+05B7 */ 0x05B7,
151 /* U+05B8 */ 0x05B8,
152 /* U+05B9 */ 0x05B9,
153 0x00,
154 /* U+05BB */ 0x05BB,
155 /* U+05BC */ 0x05BC,
156 /* U+05BD */ 0x05BD,
157 0x00,
158 /* U+05BF */ 0x05BF,
159 0x00,
160 /* U+05C1 */ 0x05C1,
161 /* U+05C2 */ 0x05C2,
162 0x00,
163 /* U+05C4 */ 0x05C4,
164 0x00,
165 
166 static const unsigned short ucs_table_064B[] = {
167 /* U+064B */ 0xFE70,
168 /* U+064C */ 0xFE72,
169 /* U+064D */ 0xFE74,
170 /* U+064E */ 0xFE76,
171 /* U+064F */ 0xFE78,
172 /* U+0650 */ 0xFE7A,
173 /* U+0651 */ 0xFE7C,
174 /* U+0652 */ 0xFE7E,
175 0x00,
176 0x00,
177 0x00,
178 0x00,
179 0x00,
180 0x00,
181 0x00,
182 0x00,
183 0x00,
184 0x00,
185 0x00,
186 0x00,
187 0x00,
188 0x00,
189 0x00,
190 0x00,
191 0x00,
192 0x00,
193 0x00,
194 0x00,
195 0x00,
196 0x00,
197 0x00,
198 0x00,
199 0x00,
200 0x00,
201 0x00,
202 0x00,
203 0x00,
204 /* U+0670 */ 0x0670,
205 0x00,
206 0x00,
207 0x00,
208 0x00,
209 0x00,
210 0x00,
211 0x00,
212 0x00,
213 0x00,
214 0x00,
215 0x00,
216 0x00,
217 0x00,
218 0x00,
219 0x00,
220 0x00,
221 0x00,
222 0x00,
223 0x00,
224 0x00,
225 0x00,
226 0x00,
227 0x00,
228 0x00,
229 0x00,
230 0x00,
34.249 spacing.h

```c
231 0x00,
232 0x00,
233 0x00,
234 0x00,
235 0x00,
236 0x00,
237 0x00,
238 0x00,
239 0x00,
240 0x00,
241 0x00,
242 0x00,
243 0x00,
244 0x00,
245 0x00,
246 0x00,
247 0x00,
248 0x00,
249 0x00,
250 0x00,
251 0x00,
252 0x00,
253 0x00,
254 0x00,
255 0x00,
256 0x00,
257 0x00,
258 0x00,
259 0x00,
260 0x00,
261 0x00,
262 0x00,
263 0x00,
264 0x00,
265 0x00,
266 0x00,
267 0x00,
268 0x00,
269 0x00,
270 0x00,
271 0x00,
272 0x00,
273 0x00,
274 0x00,
275 0x00,
276 0x00,
277 0x00,
278 0x00,
279 0x00,
280 0x00,
281 0x00,
282 0x00,
283 0x00,
284 0x00,
285 0x00,
286 0x00,
287 0x00,
288 0x00,
289 0x00,
290 0x00,
291 0x00,
292 0x00,
293 0x00,
294 0x00,
295 0x00,
296 0x00,
297 0x00,
298 0x00,
299 0x00,
300 0x00,
301 0x00,
302 0x00,
303 0x00,
304 0x00,
305 0x00,
306 /* U+06D6 */ 0x06D6,
307 /* U+06D7 */ 0x06D7,
308 /* U+06D8 */ 0x06D8,
309 /* U+06D9 */ 0x06D9,
310 /* U+06DA */ 0x06DA,
311 /* U+06DB */ 0x06DB,
312 /* U+06DC */ 0x06DC,
313 0x00,
314 0x00,
315 /* U+06DF */ 0x06DF,
316 /* U+06E0 */ 0x06E0,
317 /* U+06E1 */ 0x06E1,
```

Generated by Doxygen
1506 File Documentation

1507 /* U+06E2 */ 0x06E2,
1508 /* U+06E3 */ 0x06E3,
1509 /* U+06E4 */ 0x06E4,
1510 0x00,
1511 0x00,
1512 /* U+06E7 */ 0x06E7,
1513 /* U+06E8 */ 0x06E8,
1514 0x00,
1515 /* U+06EA */ 0x06EA,
1516 /* U+06EB */ 0x06EB,
1517 /* U+06EC */ 0x06EC,
1518 /* U+06ED */ 0x06ED,
1519 }
1520
1521 static const unsigned short ucs_table_0901[] = {
1522 /* U+0901 */ 0x0901,
1523 /* U+0902 */ 0x0902,
1524 0x00,
1525 0x00,
1526 0x00,
1527 0x00,
1528 0x00,
1529 0x00,
1530 0x00,
1531 0x00,
1532 0x00,
1533 0x00,
1534 0x00,
1535 0x00,
1536 0x00,
1537 0x00,
1538 0x00,
1539 0x00,
1540 0x00,
1541 0x00,
1542 0x00,
1543 0x00,
1544 0x00,
1545 0x00,
1546 0x00,
1547 0x00,
1548 0x00,
1549 0x00,
1550 0x00,
1551 0x00,
1552 0x00,
1553 0x00,
1554 0x00,
1555 0x00,
1556 0x00,
1557 0x00,
1558 0x00,
1559 0x00,
1560 0x00,
1561 0x00,
1562 0x00,
1563 0x00,
1564 0x00,
1565 0x00,
1566 0x00,
1567 0x00,
1568 0x00,
1569 0x00,
1570 0x00,
1571 0x00,
1572 0x00,
1573 0x00,
1574 0x00,
1575 0x00,
1576 0x00,
1577 0x00,
1578 0x00,
1579 0x00,
1580 0x00,
1581 0x00,
1582 0x00,
1583 0x00,
1584 0x00,
1585 0x00,
1586 0x00,
1587 0x00,
1588 0x00,
1589 0x00,
1590 0x00,
1591 0x00,
1592 /* U+093C */ 0x093C,
1593 0x00,
1594 0x00,
1595 0x00,
1596 0x00,
1597 /* U+0941 */ 0x0941,
1598 /* U+0942 */ 0x0942,
1599 /* U+0943 */ 0x0943,
1600 /* U+0944 */ 0x0944,
1601 /* U+0945 */ 0x0945,
1602 /* U+0946 */ 0x0946,
1603 /* U+0947 */ 0x0947,
1604 /* U+0948 */ 0x0948,
579 0x00,
580 0x00,
581 0x00,
582 0x00,
583 0x00,
584 0x00,
585 0x00,
586 0x00,
587 0x00,
588 0x00,
589 0x00,
590 /* U+0A02 */ 0x0A02,
591 0x00,
592 0x00,
593 0x00,
594 0x00,
595 0x00,
596 0x00,
597 0x00,
598 0x00,
599 0x00,
600 0x00,
601 0x00,
602 0x00,
603 0x00,
604 0x00,
605 0x00,
606 0x00,
607 0x00,
608 0x00,
609 0x00,
610 0x00,
611 0x00,
612 0x00,
613 0x00,
614 0x00,
615 0x00,
616 0x00,
617 0x00,
618 0x00,
619 0x00,
620 0x00,
621 0x00,
622 0x00,
623 0x00,
624 0x00,
625 0x00,
626 0x00,
627 0x00,
628 0x00,
629 0x00,
630 0x00,
631 0x00,
632 0x00,
633 0x00,
634 0x00,
635 0x00,
636 0x00,
637 0x00,
638 0x00,
639 0x00,
640 0x00,
641 0x00,
642 0x00,
643 0x00,
644 0x00,
645 0x00,
646 0x00,
647 0x00,
648 /* U+0A3C */ 0x0A3C,
649 0x00,
650 0x00,
651 0x00,
652 0x00,
653 /* U+0A41 */ 0x0A41,
654 /* U+0A42 */ 0x0A42,
655 0x00,
656 0x00,
657 0x00,
658 0x00,
659 /* U+0A47 */ 0x0A47,
660 /* U+0A48 */ 0x0A48,
661 0x00,
662 0x00,
663 /* U+0A4B */ 0x0A4B,
664 /* U+0A4C */ 0x0A4C,
665 /* U+0A4D */ 0x0A4D,
/* U+0ABC */ 0x0ABC,
    780 0x00,
    781 / / U+0AC1 */ 0x0AC1,
    782 /* U+0AC2 */ 0x0AC2,
    783 /* U+0AC3 */ 0x0AC3,
    784 /* U+0AC4 */ 0x0AC4,
    785 /* U+0AC5 */ 0x0AC5,
    786 0x00,
    787 /* U+0AC7 */ 0x0AC7,
    788 /* U+0AC8 */ 0x0AC8,
    789 0x00,
    790 0x00,
    791 0x00,
    792 0x00,
    793 /* U+0ACD */ 0x0ACD,
    794 0x00,
    795 0x00,
    796 0x00,
    797 0x00,
    798 0x00,
    799 0x00,
    800 0x00,
    801 0x00,
    802 0x00,
    803 0x00,
    804 0x00,
    805 0x00,
    806 0x00,
    807 0x00,
    808 0x00,
    809 0x00,
    810 0x00,
    811 0x00,
    812 0x00,
    813 0x00,
    814 0x00,
    815 0x00,
    816 0x00,
    817 0x00,
    818 0x00,
    819 0x00,
    820 0x00,
    821 0x00,
    822 0x00,
    823 0x00,
    824 0x00,
    825 0x00,
    826 0x00,
    827 0x00,
    828 0x00,
    829 0x00,
    830 0x00,
    831 0x00,
    832 0x00,
    833 0x00,
    834 0x00,
    835 0x00,
    836 0x00,
    837 0x00,
    838 0x00,
    839 0x00,
1014 0x00,
1015 0x00,
1016 0x00,
1017 0x00,
1018 0x00,
1019 0x00,
1020 0x00,
1021 0x00,
1022 0x00,
1023 0x00,
1024 0x00,
1025 0x00,
1026 0x00,
1027 0x00,
1028 0x00,
1029 0x00,
1030 0x00,
1031 0x00,
1032 0x00,
1033 0x00,
1034 0x00,
1035 0x00,
1036 / U+0BC0 / 0x0BC0,
1037 0x00,
1038 0x00,
1039 0x00,
1040 0x00,
1041 0x00,
1042 0x00,
1043 0x00,
1044 0x00,
1045 0x00,
1046 0x00,
1047 0x00,
1048 0x00,
1049 / U+0BCD / 0x0BCD,
1050 0x00,
1051 0x00,
1052 0x00,
1053 0x00,
1054 0x00,
1055 0x00,
1056 0x00,
1057 0x00,
1058 0x00,
1059 0x00,
1060 0x00,
1061 0x00,
1062 0x00,
1063 0x00,
1064 0x00,
1065 0x00,
1066 0x00,
1067 0x00,
1068 0x00,
1069 0x00,
1070 0x00,
1071 0x00,
1072 0x00,
1073 0x00,
1074 0x00,
1075 0x00,
1076 0x00,
1077 0x00,
1078 0x00,
1079 0x00,
1080 0x00,
1081 0x00,
1082 0x00,
1083 0x00,
1084 0x00,
1085 0x00,
1086 0x00,
1087 0x00,
1088 0x00,
1089 0x00,
1090 0x00,
1091 0x00,
1092 0x00,
1093 0x00,
1094 0x00,
1095 0x00,
1096 0x00,
1097 0x00,
1098 0x00,
1099 0x00,
1100 0x00,
1275 0x00,
1276 0x00,
1277 0x00,
1278 0x00,
1279 0x00,
1280 0x00,
1281 0x00,
1282 0x00,
1283 0x00,
1284 0x00,
1285 0x00,
1286 0x00,
1287 0x00,
1288 0x00,
1289 0x00,
1290 0x00,
1291 /* U+0CBF */ 0x0CBF,
1292 0x00,
1293 0x00,
1294 0x00,
1295 0x00,
1296 0x00,
1297 0x00,
1298 /* U+0CC6 */ 0x0CC6,
1299 0x00,
1300 0x00,
1301 0x00,
1302 0x00,
1303 0x00,
1304 /* U+0CCC */ 0x0CCC,
1305 /* U+0CCD */ 0x0CCD,
1306 0x00,
1307 0x00,
1308 0x00,
1309 0x00,
1310 0x00,
1311 0x00,
1312 0x00,
1313 0x00,
1314 0x00,
1315 0x00,
1316 0x00,
1317 0x00,
1318 0x00,
1319 0x00,
1320 0x00,
1321 0x00,
1322 0x00,
1323 0x00,
1324 0x00,
1325 0x00,
1326 0x00,
1327 0x00,
1328 0x00,
1329 0x00,
1330 0x00,
1331 0x00,
1332 0x00,
1333 0x00,
1334 0x00,
1335 0x00,
1336 0x00,
1337 0x00,
1338 0x00,
1339 0x00,
1340 0x00,
1341 0x00,
1342 0x00,
1343 0x00,
1344 0x00,
1345 0x00,
1346 0x00,
1347 0x00,
1348 0x00,
1349 0x00,
1350 0x00,
1351 0x00,
1352 0x00,
1353 0x00,
1354 0x00,
1355 0x00,
1356 0x00,
1357 0x00,
1358 0x00,
1359 0x00,
1360 0x00,
1361 0x00,
1362 0x00,
1363 0x00,
1364 0x00,
1365 0x00,
1366 0x00,
1367 0x00,
1368 0x00,
1369 0x00,
1370 0x00,
1371 0x00,
1372 0x00,
1373 0x00,
1374 0x00,
1375 0x00,
1376 0x00,
1377 0x00,
1378 0x00,
1379 0x00,
1380 0x00,
1381 0x00,
1382 0x00,
1383 0x00,
1384 0x00,
1385 0x00,
1386 0x00,
1387 0x00,
1388 0x00,
1389 0x00,
1390 0x00,
1391 0x00,
1392 0x00,
1393 0x00,
1394 0x00,
1395 0x00,
1396 0x00,
1397 0x00,
1398 0x00,
1399 0x00,
1400 0x00,
1401 0x00,
1402 0x00,
1403 0x00,
1404 0x00,
1405 0x00,
1406 0x00,
1407 0x00,
1408 0x00,
1409 0x00,
1410 0x00,
1411 0x00,
1412 0x00,
1413 0x00,
1414 0x00,
1415 0x00,
1416 0x00,
1417 0x00,
1418 0x00,
1419 0x00,
1420 0x00,
1421 /* U+0D41 */ 0x0D41,
1422 /* U+0D42 */ 0x0D42,
1423 /* U+0D43 */ 0x0D43,
1424 0x00,
1425 0x00,
1426 0x00,
1427 0x00,
1428 0x00,
1429 0x00,
1430 0x00,
1431 0x00,
1432 0x00,
1433 /* U+0D4D */ 0x0D4D,
1434 };
1435
1436 static const unsigned short ucs_table_0E31[] = {
1437 /* U+0E31 */ 0x0E31,
1438 /* U+0E34 */ 0x0E34,
1439 /* U+0E35 */ 0x0E35,
1440 /* U+0E36 */ 0x0E36,
1441 /* U+0E37 */ 0x0E37,
1442 /* U+0E38 */ 0x0E38,
1443 /* U+0E39 */ 0x0E39,
1444 /* U+0E3A */ 0x0E3A,
1445 /* U+0E3B */ 0x0E3B,
1446 /* U+0E3C */ 0x0E3C,
1447 0x00,
```c
1884 0x00,
1885 0x00,
1886 0x00,
1887 0x00,
1888 0x00,
1889 0x00,
1890 0x00,
1891 0x00,
1892 0x00,
1893 0x00,
1894 0x00,
1895 0x00,
1896 0x00,
1897 0x00,
1898 0x00,
1899 0x00,
1900 0x00,
1901 0x00,
1902 0x00,
1903 0x00,
1904 0x00,
1905 0x00,
1906 0x00,
1907 0x00,
1908 0x00,
1909 0x00,
1910 0x00,
1911 0x00,
1912 0x00,
1913 0x00,
1914 0x00,
1915 0x00,
1916 0x00,
1917 0x00,
1918 0x00,
1919 0x00,
1920 0x00,
1921 0x00,
1922 0x00,
1923 0x00,
1924 0x00,
1925 0x00,
1926 0x00,
1927 0x00,
1928 0x00,
1929 0x00,
1930 0x00,
1931 0x00,
1932 0x00,
1933 0x00,
1934 0x00,
1935 0x00,
1936 0x00,
1937 0x00,
1938 0x00,
1939 0x00,
1940 0x00,
1941 0x00,
1942 0x00,
1943 0x00,
1944 0x00,
1945 0x00,
1946 0x00,
1947 0x00,
1948 0x00,
1949 0x00,
1950 0x00,
1951 0x00,
1952 0x00,
1953 0x00,
1954 0x00,
1955 0x00,
1956 0x00,
1957 0x00,
1958 0x00,
1959 0x00,
1960 0x00,
1961 0x00,
1962 0x00,
1963 0x00,
1964 0x00,
1965 /* U+3099 */ 0x309B,
1966 /* U+309A */ 0x309C,
1967 */
1968 */
1969 static const unsigned short ucs_table_FB1E[] = {
1970 /* U+FB1E */ 0xFB1E,
```

Generated by Doxygen
1972
1973 static const unsigned short ucs_table_FE20[] = {
1974 /* U+FE20 */ 0xFE20,
1975 /* U+FE21 */ 0xFE21,
1976 /* U+FE22 */ 0xFE22,
1977 /* U+FE23 */ 0xFE23,
1978
};

34.250 symbol_h

1 /* symbol */
2 static const char unicode_to_symbol_1b_0020[] = {
3 /* U+0020 */ 0x20,
4 /* U+0021 */ 0x21,
5 /* U+0023 */ 0x23,
6 /* U+0025 */ 0x25,
7 /* U+0026 */ 0x26,
8 /* U+0028 */ 0x28,
9 /* U+0029 */ 0x29,
10 /* U+002B */ 0x2B,
11 /* U+002C */ 0x2C,
12 /* U+0030 */ 0x30,
13 /* U+0031 */ 0x31,
14 /* U+0032 */ 0x32,
15 /* U+0033 */ 0x33,
16 /* U+0034 */ 0x34,
17 /* U+0035 */ 0x35,
18 /* U+0036 */ 0x36,
19 /* U+0037 */ 0x37,
20 /* U+0038 */ 0x38,
21 /* U+0039 */ 0x39,
22 /* U+003A */ 0x3A,
23 /* U+003B */ 0x3B,
24 /* U+003C */ 0x3C,
25 /* U+003D */ 0x3D,
26 /* U+003E */ 0x3E,
27 /* U+003F */ 0x3F,
28 /* U+0040 */ 0x40,
29 /* U+0041 */ 0x41,
30 /* U+0042 */ 0x42,
31 /* U+0043 */ 0x43,
32 /* U+0044 */ 0x44,
33 /* U+0045 */ 0x45,
34 /* U+0046 */ 0x46,
35 /* U+0047 */ 0x47,
36 /* U+0048 */ 0x48,
37 /* U+0049 */ 0x49,
38 /* U+004A */ 0x4A,
39 /* U+004B */ 0x4B,
40 /* U+004C */ 0x4C,
41 /* U+004D */ 0x4D,
42 /* U+004E */ 0x4E,
43 /* U+004F */ 0x4F,
44 /* U+0050 */ 0x50,
45 /* U+0051 */ 0x51,
46 /* U+0052 */ 0x52,
47 /* U+0053 */ 0x53,
48 /* U+0054 */ 0x54,
49 /* U+0055 */ 0x55,
50 /* U+0056 */ 0x56,
51 /* U+0057 */ 0x57,
52 /* U+0058 */ 0x58,
53 /* U+0059 */ 0x59,
54 /* U+005A */ 0x5A,
55 /* U+005B */ 0x5B,
56 /* U+005C */ 0x5C,
57 /* U+005D */ 0x5D,
58 /* U+005E */ 0x5E,
59 /* U+005F */ 0x5F,
60 /* U+0060 */ 0x60,
61 /* U+0061 */ 0x61,
62 /* U+0062 */ 0x62,
63 /* U+0063 */ 0x63,
64 /* U+0064 */ 0x64,
65 /* U+0065 */ 0x65,
66 /* U+0066 */ 0x66,
67 /* U+0067 */ 0x67,
68 /* U+0068 */ 0x68,
69 /* U+0069 */ 0x69,
70 /* U+006A */ 0x6A,
71 /* U+006B */ 0x6B,
72 /* U+006C */ 0x6C,
73 /* U+006D */ 0x6D,
74 /* U+006E */ 0x6E,
75 /* U+006F */ 0x6F,
76 /* U+0070 */ 0x70,
77 /* U+0071 */ 0x71,
78 /* U+0072 */ 0x72,
79 /* U+0073 */ 0x73,
80 /* U+0074 */ 0x74,
81 /* U+0075 */ 0x75,
82 /* U+0076 */ 0x76,
83 /* U+0077 */ 0x77,
84 /* U+0078 */ 0x78,
85 /* U+0079 */ 0x79,
86 /* U+007A */ 0x7A,
87 /* U+007B */ 0x7B,
88 /* U+007C */ 0x7C,
89 /* U+007D */ 0x7D,
90 /* U+007E */ 0x7E,
91 /* U+007F */ 0x7F,
92 /* U+0080 */ 0x80,
93 /* U+0081 */ 0x81,
94 /* U+0082 */ 0x82,
95 /* U+0083 */ 0x83,
96 /* U+0084 */ 0x84,
97 /* U+0085 */ 0x85,
98 /* U+0086 */ 0x86,
99 /* U+0087 */ 0x87,
100 /* U+0088 */ 0x88,
101 /* U+0089 */ 0x89,
102 /* U+008A */ 0x8A,
103 /* U+008B */ 0x8B,
104 /* U+008C */ 0x8C,
105 /* U+008D */ 0x8D,
106 /* U+008E */ 0x8E,
107 /* U+008F */ 0x8F,
108 /* U+0090 */ 0x90,
109 /* U+0091 */ 0x91,
110 /* U+0092 */ 0x92,
111 /* U+0093 */ 0x93,
112 /* U+0094 */ 0x94,
113 /* U+0095 */ 0x95,
114 /* U+0096 */ 0x96,
115 /* U+0097 */ 0x97,
116 /* U+0098 */ 0x98,
117 /* U+0099 */ 0x99,
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 75 | 0x00, | 76 | 0x00, | 77 | 0x00, | 78 | 0x00, | 79 | 0x00, | 80 | 0x00, | 81 | 0x00, | 82 | 0x00, | 83 | 0x00, | 84 | 0x00, |
| 85 | 0x00, | 86 | 0x00, | 87 | 0x00, | 88 | 0x00, | 89 | 0x00, | 90 | 0x00, | 91 | 0x00, | 92 | 0x00, | 93 | 0x00, | 94 | 0x00, |
| 95 | /* U+007B */ 0x7B, | 96 | /* U+007C */ 0x7C, | 97 | /* U+007D */ 0x7D, | 98 | 0x00, | 99 | 0x00, | 100 | 0x00, | 101 | 0x00, | 102 | 0x00, | 103 | 0x00, | 104 | 0x00, |
| 105 | 0x00, | 106 | 0x00, | 107 | 0x00, | 108 | 0x00, | 109 | 0x00, | 110 | 0x00, | 111 | 0x00, | 112 | 0x00, | 113 | 0x00, | 114 | 0x00, |
| 115 | 0x00, | 116 | 0x00, | 117 | 0x00, | 118 | 0x00, | 119 | 0x00, | 120 | 0x00, | 121 | 0x00, | 122 | 0x00, | 123 | 0x00, | 124 | 0x00, |
| 125 | 0x00, | 126 | 0x00, | 127 | 0x00, | 128 | 0x00, | 129 | 0x00, | 130 | 0x00, | 131 | 0x00, | 132 | /* U+05A0 */ 0x20, | 133 | 0x00, | 134 | 0x00, |
| 135 | 0x00, | 136 | 0x00, | 137 | 0x00, | 138 | 0x00, | 139 | 0x00, | 140 | 0x00, | 141 | 0x00, | 142 | 0x00, | 143 | 0x00, |
| 144 | /* U+05AC */ (char)0xD8, | 145 | 0x00, | 146 | 0x00, | 147 | 0x00, | 148 | /* U+05B0 */ (char)0xB0, | 149 | /* U+05B1 */ (char)0xB1, | 150 | 0x00, | 151 | 0x00, | 152 | 0x00, |
| 153 | /* U+05B5 */ 0x6D, | 154 | 0x00, | 155 | 0x00, | 156 | 0x00, | 157 | 0x00, | 158 | 0x00, | 159 | 0x00, | 160 | 0x00, | 161 | 0x00, |

Generated by Doxygen
162 0x00,
163 0x00,
164 0x00,
165 0x00,
166 0x00,
167 0x00,
168 0x00,
169 0x00,
170 0x00,
171 0x00,
172 0x00,
173 0x00,
174 0x00,
175 0x00,
176 0x00,
177 0x00,
178 0x00,
179 0x00,
180 0x00,
181 0x00,
182 0x00,
183 0x00,
184 0x00,
185 0x00,
186 0x00,
187 /* U+00D7 */ (char)0xB4,
188 0x00,
189 0x00,
190 0x00,
191 0x00,
192 0x00,
193 0x00,
194 0x00,
195 0x00,
196 0x00,
197 0x00,
198 0x00,
199 0x00,
200 0x00,
201 0x00,
202 0x00,
203 0x00,
204 0x00,
205 0x00,
206 0x00,
207 0x00,
208 0x00,
209 0x00,
210 0x00,
211 0x00,
212 0x00,
213 0x00,
214 0x00,
215 0x00,
216 0x00,
217 0x00,
218 0x00,
219 /* U+00F7 */ (char)0xB8,
220 \}
221 static const char unicode_to_symbol_1b_0192[ ] = { }
222 static const char unicode_to_symbol_1b_0391[ ] = { }
223 / * U+0391 */ 0x41,
224 / * U+0392 */ 0x42,
225 / * U+0393 */ 0x47,
226 / * U+0394 */ 0x44,
227 / * U+0395 */ 0x45,
228 / * U+0396 */ 0x5A,
229 / * U+0397 */ 0x48,
230 / * U+0398 */ 0x51,
231 / * U+0399 */ 0x49,
232 / * U+039A */ 0x4B,
233 / * U+039B */ 0x4C,
234 / * U+039C */ 0x4D,
235 / * U+039D */ 0x4E,
236 / * U+039E */ 0x58,
237 / * U+039F */ 0x4F,
238 / * U+03A0 */ 0x50,
239 / * U+03A1 */ 0x52,
240 \}
241 / * U+03A2 */ 0x53,
242 / * U+03A3 */ 0x54,
243 / * U+03A4 */ 0x55,
244 / * U+03A5 */ 0x46,
249 /* U+03A7 */ 0x43,
250 /* U+03A8 */ 0x59,
251 /* U+03A9 */ 0x57,
252 0x00,
253 0x00,
254 0x00,
255 0x00,
256 0x00,
257 0x00,
258 0x00,
259 /* U+03B1 */ 0x61,
260 /* U+03B2 */ 0x62,
261 /* U+03B3 */ 0x67,
262 /* U+03B4 */ 0x64,
263 /* U+03B5 */ 0x65,
264 /* U+03B6 */ 0x7A,
265 /* U+03B7 */ 0x68,
266 /* U+03B8 */ 0x71,
267 /* U+03B9 */ 0x69,
268 /* U+03BA */ 0x6B,
269 /* U+03BB */ 0x6C,
270 /* U+03BC */ 0x6D,
271 /* U+03BD */ 0x6E,
272 /* U+03BE */ 0x78,
273 /* U+03BF */ 0x5F,
274 /* U+03C0 */ 0x72,
275 /* U+03C1 */ 0x73,
276 /* U+03C2 */ 0x74,
277 /* U+03C3 */ 0x75,
278 /* U+03C4 */ 0x76,
279 /* U+03C5 */ 0x77,
280 /* U+03C6 */ 0x79,
281 /* U+03C7 */ 0x7A,
282 /* U+03C8 */ 0x7B,
283 /* U+03C9 */ 0x7E,
284 0x00,
285 0x00,
286 0x00,
287 0x00,
288 0x00,
289 0x00,
290 0x00,
291 /* U+03D1 */ 0x4A,
292 /* U+03D2 */ (char)0xA1,
293 0x00,
294 0x00,
295 /* U+03D5 */ 0x56,
296 /* U+03D6 */ 0x5C,
297 */
298 static const char unicode_to_symbol_1b_2022[] = {
299 /* U+2022 */ (char)0xB7,
300 0x00,
301 0x00,
302 0x00,
303 0x00,
304 /* U+2026 */ (char)0xBC,
305 0x00,
306 0x00,
307 0x00,
308 0x00,
309 0x00,
310 0x00,
311 0x00,
312 0x00,
313 0x00,
314 0x00,
315 0x00,
316 /* U+2032 */ (char)0xA2,
317 /* U+2033 */ (char)0xB2,
318 0x00,
319 0x00,
320 0x00,
321 0x00,
322 0x00,
323 0x00,
324 0x00,
325 0x00,
326 0x00,
327 0x00,
328 0x00,
329 0x00,
330 0x00,
331 0x00,
332 0x00,
333 0x00,
334 /* U+2044 */ (char)0xA4,
34.250 symbol_.h

510 0x00,
511 0x00,
512 0x00,
513 0x00,
514 0x00,
515 0x00,
516 0x00,
517 0x00,
518 0x00,
519 0x00,
520 0x00,
521 0x00,
522 0x00,
523 0x00,
524 0x00,
525 0x00,
526 0x00,
527 0x00,
528 0x00,
529 0x00,
530 0x00,
531 0x00,
532 0x00,
533 0x00,
534 0x00,
535 0x00,
536 0x00,
537 0x00,
538 0x00,
539 /* U+2111 */ (char)0xC1,
540 0x00,
541 0x00,
542 0x00,
543 0x00,
544 0x00,
545 0x00,
546 /* U+2118 */ (char)0xC3,
547 0x00,
548 0x00,
549 0x00,
550 /* U+211C */ (char)0xC2,
551 0x00,
552 0x00,
553 0x00,
554 0x00,
555 0x00,
556 0x00,
557 0x00,
558 0x00,
559 0x00,
560 /* U+2126 */ 0x57,
561 0x00,
562 0x00,
563 0x00,
564 0x00,
565 0x00,
566 0x00,
567 0x00,
568 0x00,
569 0x00,
570 0x00,
571 0x00,
572 0x00,
573 0x00,
574 0x00,
575 /* U+2135 */ (char)0xC0,
576 0x00,
577 0x00,
578 0x00,
579 0x00,
580 0x00,
581 0x00,
582 0x00,
583 0x00,
584 0x00,
585 0x00,
586 0x00,
587 0x00,
588 0x00,
589 0x00,
590 0x00,
591 0x00,
592 0x00,
593 0x00,
594 0x00,
595 0x00,
596 0x00,
694 0x00,
695 0x00,
696 0x00,
697 0x00,
698 0x00,
699 0x00,
700 0x00,
701 0x00,
702 0x00,
703 /* U+21B5 */ (char)0xBF,
704 0x00,
705 0x00,
706 0x00,
707 0x00,
708 0x00,
709 0x00,
710 0x00,
711 0x00,
712 0x00,
713 0x00,
714 0x00,
715 0x00,
716 0x00,
717 0x00,
718 0x00,
719 0x00,
720 0x00,
721 0x00,
722 0x00,
723 0x00,
724 0x00,
725 0x00,
726 0x00,
727 0x00,
728 0x00,
729 0x00,
730 /* U+21D0 */ (char)0xDC,
731 /* U+21D1 */ (char)0xDD,
732 /* U+21D2 */ (char)0xDE,
733 /* U+21D3 */ (char)0xDF,
734 /* U+21D4 */ (char)0xDB,
771 0x00,
772 0x00,
773 0x00,
774 0x00,
775 0x00,
776 0x00,
777 0x00,
778 /* U+2200 */ 0x22,
779 0x00,
780 /* U+2202 */ (char)0xB6,
781 /* U+2203 */ 0x24,
782 0x00,
783 /* U+2205 */ (char)0xC6,
784 /* U+2206 */ 0x44,
785 /* U+2207 */ (char)0xD1,
786 /* U+2208 */ (char)0xCE,
787 /* U+2209 */ (char)0xCF,
788 0x00,
789 / * U+220A */ 0x27,
790 0x00,
791 0x00,
792 0x00,
793 /* U+220F */ (char)0xD5,
794 0x00,
795 /* U+2211 */ (char)0xE5,
796 /* U+2212 */ 0x2D,
797 0x00,
798 0x00,
799 /* U+2215 */ (char)0xA4,
800 0x00,
801 /* U+2217 */ 0x2A,
802 0x00,
803 0x00,
804 /* U+221A */ (char)0xA5,
805 0x00,
806 0x00,
807 /* U+221D */ (char)0xD6,
808 0x00,
809 0x00,
810 /* U+2220 */ (char)0xD9,
811 0x00,
812 0x00,
813 0x00,
814 0x00,
815 0x00,
816 0x00,
817 /* U+2229 */ (char)0xE9,
818 /* U+222A */ (char)0xA9,
819 /* U+222B */ (char)0xF2,
820 0x00,
821 /* U+222E */ 0x7E,
822 0x00,
823 0x00,
824 0x00,
825 0x00,
826 0x00,
827 0x00,
828 0x00,
829 0x00,
830 /* U+2234 */ 0x5C,
831 0x00,
832 0x00,
833 0x00,
834 0x00,
835 0x00,
836 0x00,
837 0x00,
838 /* U+223C */ 0x7E,
839 0x00,
840 0x00,
841 0x00,
842 0x00,
843 0x00,
844 0x00,
845 0x00,
846 0x00,
847 /* U+2245 */ 0x40,
848 0x00,
849 0x00,
850 /* U+2248 */ (char)0xBB,
851 0x00,
852 0x00,
853 0x00,
1032 0x00,
1033 0x00,
1034 0x00,
1035 0x00,
1036 0x00,
1037 0x00,
1038 0x00,
1039 0x00,
1040 0x00,
1041 0x00,
1042 0x00,
1043 0x00,
1044 0x00,
1045 0x00,
1046 0x00,
1047 0x00,
1048 0x00,
1049 0x00,
1050 0x00,
1051 0x00,
1052 0x00,
1053 0x00,
1054 0x00,
1055 0x00,
1056 0x00,
1057 0x00,
1058 0x00,
1059 0x00,
1060 0x00,
1061 0x00,
1062 0x00,
1063 0x00,
1064 0x00,
1065 0x00,
1066 /* U+2320 */ (char)0xF3,
1067 /* U+2321 */ (char)0xF5,
1068 0x00,
1069 0x00,
1070 0x00,
1071 0x00,
1072 0x00,
1073 0x00,
1074 0x00,
1075 /* U+2329 */ (char)0xE1,
1076 /* U+232A */ (char)0xF1,
1077 \);\n\n1078 static const char unicode_to_symbol_1b_25CA[] = {
1079 /* U+25CA */ (char)0xE0,
1080 };
1081 \};
1082 static const char unicode_to_symbol_1b_2660[] = {
1083 /* U+2660 */ (char)0xAA,
1084 0x00,
1085 0x00,
1086 0x00,
1087 /* U+2663 */ (char)0xA7,
1088 0x00,
1089 /* U+2665 */ (char)0xA9,
1090 /* U+2666 */ (char)0xA8,
1091 \};
1092 static const char unicode_to_symbol_1b_F6D9[] = {
1093 /* U+F6D9 */ (char)0xD3,
1094 /* U+F6DA */ (char)0xD2,
1095 /* U+F6DB */ (char)0xD4,
1096 \};
1097 \};
1098 static const char unicode_to_symbol_1b_F8E5[] = {
1099 /* U+F8E5 */ (char)0x60,
1100 /* U+F8E6 */ (char)0xBD,
1101 /* U+F8E7 */ (char)0xBE,
1102 /* U+F8E8 */ (char)0xE2,
1103 /* U+F8E9 */ (char)0xE3,
1104 /* U+F8EA */ (char)0xE4,
1105 /* U+F8EB */ (char)0xE6,
1106 /* U+F8EC */ (char)0xE7,
1107 /* U+F8ED */ (char)0xE8,
1108 /* U+F8EE */ (char)0xE9,
1109 /* U+F8EF */ (char)0xEA,
1110 /* U+F8F0 */ (char)0xEB,
1111 /* U+F8F1 */ (char)0xEC,
1112 /* U+F8F2 */ (char)0xED,
1113 /* U+F8F3 */ (char)0xEE,
1114 /* U+F8F4 */ (char)0xEF,
1115 /* U+F8F5 */ (char)0xF4,
1116 /* U+F8F6 */ (char)0xF6,
1117 /* U+F8F7 */ (char)0xF7,
34.251  armscii_8.h

2
3 /* ARMSCII-8 */
4
5 static const unsigned short armscii_8_2uni[96] = {
6    / * 0xa0 */
7    0x00a0, 0xfffd, 0x0587, 0x0589, 0x0029, 0x0028, 0x00bb, 0x00ab,
8    0x00ba, 0x002e, 0x055d, 0x002c, 0x002d, 0x058a, 0x2026, 0x055c,
9    / * 0xb0 */
10    0x055b, 0x055e, 0x0531, 0x0561, 0x0532, 0x0562, 0x0533, 0x0563,
11    0x0534, 0x0564, 0x0535, 0x0565, 0x0536, 0x0566, 0x0537, 0x0567,
12    / * 0xc0 */
13    0x0538, 0x0568, 0x0539, 0x0569, 0x053a, 0x056a, 0x053b, 0x056b,
14    0x053c, 0x056c, 0x053d, 0x056d, 0x053e, 0x056e, 0x053f, 0x056f,
15    / * 0xd0 */
16    0x0540, 0x0570, 0x0541, 0x0571, 0x0542, 0x0572, 0x0543, 0x0573,
17    0x0544, 0x0574, 0x0545, 0x0575, 0x0546, 0x0576, 0x0547, 0x0577,
18    / * 0xe0 */
19    0x0548, 0x0578, 0x0549, 0x0579, 0x054a, 0x057a, 0x054b, 0x057b,
20    0x054c, 0x057c, 0x054d, 0x057d, 0x054e, 0x057e, 0x054f, 0x057f,
21    / * 0xf0 */
22    0x0550, 0x0580, 0x0551, 0x0581, 0x0552, 0x0582, 0x0553, 0x0583,
23    0x0554, 0x0584, 0x0555, 0x0585, 0x0556, 0x0586, 0x055a, 0xfffd,
24    };
static int
armscii_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0028) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0028 && wc < 0x0030)
        c = armscii_8_page00[wc-0x0028];
    else if (wc >= 0x0030 && wc < 0x00a0)
        c = wc;
    else if (wc >= 0x00a0 && wc < 0x00c0)
        c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)
        c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)
        c = armscii_8_page20[wc-0x2010];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static int
ascii_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        *pwc = (ucs4_t) c;
        return 1;
    }
    return RET_ILSEQ;
}

static int
ascii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    return RET_ILSEQ;
}

static int
armscii_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0028) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0028 && wc < 0x0030)
        c = armscii_8_page00[wc-0x0028];
    else if (wc >= 0x0030 && wc < 0x00a0)
        c = wc;
    else if (wc >= 0x00a0 && wc < 0x00c0)
        c = armscii_8_page00_1[wc-0x00a0];
    else if (wc >= 0x0530 && wc < 0x0590)
        c = armscii_8_page05[wc-0x0530];
    else if (wc >= 0x2010 && wc < 0x2028)
        c = armscii_8_page20[wc-0x2010];
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

static int
ascii_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        *pwc = (ucs4_t) c;
        return 1;
    }
    return RET_ILSEQ;
}

static int
ascii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    return RET_ILSEQ;
}

#define NEED_TOWC

static const unsigned short big5_2uni_pagea1[6121] = {
    0xa1
};


/* BIG5 */
static int
big5_smbtwc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c1 = s[0];
    if ((c1 >= 0x40 && c1 <= 0x7f) || (c1 >= 0x80 && c1 <= 0xff))
        return 2;
    unsigned char c2 = s[1];
    if ((c2 >= 0x40 && c2 <= 0x7f) || (c2 >= 0x80 && c2 <= 0xff))
        return big5_2uni_pagec9[i-6280];
    if (i < 13932)
        return big5_2uni_pagea1[i];
    else
        return big5_2uni_pagea9[i-6280];

    if (wc != 0xfffd)
        return RET_ILSEQ;
    *pwc = (ucs4_t) wc;
    return RET_ILSEQ;
}
4029 static const Summary16 big5_uni2indx_pagefa[1] = {
4030 / * 0xfa00 */
4031 { 13563, 0x3000 },
4032 { 13574, 0x0000 }, { 13589, 0x0000 },
4033 { 13592, 0x0000 }, { 13595, 0x0000 },
4034 { 13598, 0x0000 }, { 13601, 0x0000 },
4035 { 13604, 0x0000 }, { 13607, 0x0000 },
4036 { 13609, 0x0000 }, { 13612, 0x0000 },
4037 { 13615, 0x0000 }, { 13618, 0x0000 },
4038 / * 0xff00 */
4039 { 13615, 0x0000 }, { 13628, 0x0000 }, { 13644, 0x0000 },
4040 { 13647, 0x0000 }, { 13654, 0x0000 }, { 13660, 0x0000 },
4041 { 13663, 0x0000 }, { 13666, 0x0000 }, { 13670, 0x0000 },
4042 { 13673, 0x00ff }, { 13676, 0x0000 }, { 13679, 0x0000 },
4043 { 13682, 0x0000 }, { 13685, 0x0000 }, { 13688, 0x0000 },
4044 { 13691, 0x0000 }, { 13694, 0x0000 }, { 13697, 0x0000 },
4045 { 13700, 0x0000 }, { 13703, 0x0000 }, { 13706, 0x0000 },
4046 { 13709, 0x0000 }, { 13712, 0x0000 }, { 13715, 0x0000 },
4047 / * 0x9a00 */
4048 { 13731, 0x0000 }, { 13734, 0x0000 }, { 13737, 0x0000 },
4049 / * 0x9b00 */
4050 { 13751, 0x0000 }, { 13754, 0x0000 }, { 13757, 0x0000 },
4051 / * 0x9c00 */
4052 { 13771, 0x0000 }, { 13774, 0x0000 }, { 13777, 0x0000 },
4053 / * 0x9d00 */
4054 { 13791, 0x0000 }, { 13794, 0x0000 }, { 13797, 0x0000 },
4055 / * 0x9e00 */
4056 { 13811, 0x0000 }, { 13814, 0x0000 }, { 13817, 0x0000 },
4057 / * 0x9f00 */
4058 { 13831, 0x0000 }, { 13834, 0x0000 }, { 13837, 0x0000 },
4059 / * 0xa000 */
4060 { 13851, 0x0000 }, { 13854, 0x0000 }, { 13857, 0x0000 },
4061 / * 0xa100 */
4062 { 13871, 0x0000 }, { 13874, 0x0000 }, { 13877, 0x0000 },
4063 / * 0xa200 */
4064 { 13891, 0x0000 }, { 13894, 0x0000 }, { 13897, 0x0000 },
4065 / * 0xa300 */
4066 { 13911, 0x0000 }, { 13914, 0x0000 }, { 13917, 0x0000 },
4067 / * 0xa400 */
4068 { 13931, 0x0000 }, { 13934, 0x0000 }, { 13937, 0x0000 },
4069 / * 0xa500 */
4070 { 13951, 0x0000 }, { 13954, 0x0000 }, { 13957, 0x0000 },
4071 / * 0xa600 */
4072 { 13971, 0x0000 }, { 13974, 0x0000 }, { 13977, 0x0000 },
4073 / * 0xa700 */
4074 { 13991, 0x0000 }, { 13994, 0x0000 }, { 13997, 0x0000 },
4075 / * 0xa800 */
4076 { 14011, 0x0000 }, { 14014, 0x0000 }, { 14017, 0x0000 },
4077 / * 0xa900 */
4078 { 14031, 0x0000 }, { 14034, 0x0000 }, { 14037, 0x0000 },
4079 / * 0xaa00 */
4080 { 14051, 0x0000 }, { 14054, 0x0000 }, { 14057, 0x0000 },
4081 / * 0xab00 */
4082 { 14071, 0x0000 }, { 14074, 0x0000 }, { 14077, 0x0000 },
4083 / * 0xac00 */
4084 { 14091, 0x0000 }, { 14094, 0x0000 }, { 14097, 0x0000 },
4085 / * 0xad00 */
4086 { 14111, 0x0000 }, { 14114, 0x0000 }, { 14117, 0x0000 },
4087 / * 0xae00 */
4088 " big5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
4089 (void)conv;
4090 if (n >= 2) {
4091 const Summary16 *summary = NULL;
4092 if (wc < 0x0100)
4093 summary = &big5_uni2indx_page00[(wc»4)];
4094 else if (wc >= 0x0200 && wc < 0x0460)
4095 summary = &big5_uni2indx_page02[(wc»4)-0x200];
4096 else if (wc >= 0x2000 && wc < 0x22c0)
4097 summary = &big5_uni2indx_page20[(wc»4)-0x200];
4098 else if (wc >= 0x2400 && wc < 0x2650)
4099 summary = &big5_uni2indx_page24[(wc»4)-0x200];
4100 else if (wc >= 0x2800 && wc < 0x2a50)
4101 else if (wc >= 0x2a60 && wc < 0x2b00)
34.254 big5_emacs.h

4116    summary = &big5_uni2indx_page24[(wc»4)-0x240];
4117 else if (wc >= 0x3000 && wc < 0x33e0) {
4118    summary = &big5_uni2indx_page30[(wc»4)-0x300];
4119 else if (wc >= 0x4e00 && wc < 0x9fb0) {
4120    summary = &big5_uni2indx_page4e[(wc»4)-0x4e0];
4121 else if (wc >= 0xfa00 && wc < 0xfa10) {
4122    summary = &big5_uni2indx_pagefa[(wc»4)-0xfa0];
4123 else if (wc >= 0xfe00 && wc < 0xff70) {
4124    summary = &big5_uni2indx_pagefe[(wc»4)-0xfe0];
4125      if (summary) {
4126      unsigned short used = summary->used;
4127      unsigned int i = wc & 0x0f;
4128      if (used & ((unsigned short) 1 « i)) {
4129      unsigned short c;
4130      /* Keep in 'used' only the bits 0..i-1. */
4131      used &= (unsigned short) 1 « i) - 1;
4132      /* Add 'summary->indx' and the number of bits set in 'used'. */
4133      used = (used & 0x3333) + ((used & 0xaaaa) » 1);
4134      used = (used & 0xaaaa) + ((used & 0xcccc) >> 2);
4135      used = (used & 0x0fff) + (used & 0xf0f0) + 4);
4136      used = (used & 0x0fff) + (used & 8);
4137      c = big5_2charset[summary->indx + used];
4138      r[0] = (c » 8); r[1] = (c & 0xff);
4139      return 2;  
4140      }
4141      }
4142      return RET_ILSEQ;
4143      }
4144      return RET_TOOSMALL;
4145    }
4146  }

34.254 big5_emacs.h

2 */
3 /*
4 BIG5-0 and BIG5-1
5 */
6 */
7 */
8 BIG5 with its 13494 characters doesn't fit in a single 94x94 or 96x96
9 block. Therefore Emacs/Mule developers, in a typically Japanese way of
10 thinking, have developed an alternative encoding of BIG5 in two 94x94
11 planes, very similar to the SHIFT_JIS encoding for JISX0208.
12
13 Conversion between BIG5 codes (s1,s2) and BIG5-0 codes (c1,c2):
14 Example. (s1,s2) = 0xA140, (c1,c2) = 0x2121.
15 0xa1 <= s1 <= 0xc7, 0x40 <= s2 <= 0x7e || 0xa1 <= s2 <= 0xfe,  
16 0x21 <= c1 <= 0x62, 0x21 <= c2 <= 0x7e.
17 Invariant:
18 157*(s1-0xA1) + (s2 < 0x80 ? s2-0x40 : s2-0x62)
19 = 94*(c1-0x21)+(c2-0x21)
20 Conversion (s1,s2) -> (c1,c2):
21 t := 157*(s1-0xA1) + (s2 < 0x80 ? s2-0x40 : s2-0x62)  
22 c1 := (t div 94) + 0x21  
23 c2 := (t mod 94) + 0x21
24 Conversion (c1,c2) -> (s1,s2):
25 t := 94*(c1-0x21)+(c2-0x21)  
26 t2 := t mod 157  
27 s1 := (t div 157) + 0xa1  
28 s2 := (t < 0x3f ? t2+0x40 : t2+0x62)
29 */
30 Conversion between BIG5 codes (s1,s2) and BIG5-1 codes (c1,c2):
31 Example. (s1,s2) = 0xC940, (c1,c2) = 0x2121.
32 0xc9 <= s1 <= 0xf9, 0x40 <= s2 <= 0x7e || 0xa1 <= s2 <= 0xfe,
33 0x21 <= c1 <= 0x72, 0x21 <= c2 <= 0x7e.
34 Invariant:
35 157*(s1-0xC9) + (s2 < 0x80 ? s2-0x40 : s2-0x62)
36 = 94*(c1-0x21)+(c2-0x21)
37 Conversion (s1,s2) -> (c1,c2):
38 t := 157*(s1-0xC9) + (s2 < 0x80 ? s2-0x40 : s2-0x62)  
39 c1 := (t div 94) + 0x21  
40 c2 := (t mod 94) + 0x21
41 Conversion (c1,c2) -> (s1,s2):
42 t := 94*(c1-0x21)+(c2-0x21)  
43 t2 := t mod 157  
44 s1 := (t div 157) + 0xc9  
45 s2 := (t < 0x3f ? t2+0x40 : t2+0x62)
46 */
47
48 static int
49 big5_0_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {  
50   unsigned char c1 = s[0];
51
Generated by Doxygen
if (c1 >= 0x21 && c1 <= 0x62) {
  if (n >= 2) {
    unsigned char c2 = s[1];
    if (c2 >= 0x21 && c2 <= 0x7e) {
      unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
      if (0) {
        /* Unoptimized. */
        unsigned char buf[2];
        buf[0] = (i / 157) + 0xa1;
        i = i % 157;
        buf[1] = i + (i < 0x3f ? 0x40 : 0x62);
        return big5_mbtowc(conv,pwc,buf,2);
      } else {
        /* Inline the implementation of big5_mbtowc. */
        if (i < 6121) {
          unsigned short wc = big5_2uni_pagea1[i];
          if (wc != 0xfffd) {  // This branch is not taken in practice.
            *pwc = (ucs4_t) wc;
            return 2;
          }
        }
        return RET_ILSEQ;
      }
    } else {
      /* Inline the implementation of big5_mbtowc. */
      if (i < 7652) {
        unsigned short wc = big5_2uni_pagec9[i];
        if (wc != 0xfffd) {  // This branch is not taken in practice.
          *pwc = (ucs4_t) wc;
          return 2;
        }
      }
      return RET_ILSEQ;
    }
  }
  return RET_TOOFEW(0);
}

static int
big5_1_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c1 = s[0];
  if (c1 >= 0x21 && c1 <= 0x72) {
    if (n >= 2) {
      unsigned char c2 = s[1];
      if (c2 >= 0x21 && c2 <= 0x7e) {
        unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
        if (0) {
          /* Unoptimized. */
          unsigned char buf[2];
          buf[0] = (i / 157) + 0xc9;
          i = i % 157;
          buf[1] = i + (i < 0x3f ? 0x40 : 0x62);
          return big5_mbtowc(conv,pwc,buf,2);
        } else {
          /* Inline the implementation of big5_mbtowc. */
          if (i < 7652) {
            unsigned short wc = big5_2uni_pagec9[i];
            if (wc != 0xfffd) {  // This branch is not taken in practice.
              *pwc = (ucs4_t) wc;
              return 2;
            }
          }
          return RET_ILSEQ;
        }
      } else {
        return RET_ILSEQ;
      }  // End of the else block.
    } else {
      return RET_ILSEQ;
    }  // End of the if block.
  } else {
    return RET_TOOFEW(0);
  }
}

static int
big5_0_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  if (n >= 2) {
    unsigned char buf[2];
    int ret = big5_wctomb(conv,buf,wc,2);
    if (ret != RET_ILSEQ) {
      unsigned char s1, s2;
      if (ret == 2) abort();
      s1 = buf[0];
      s2 = buf[1];
      if (s1 >= 0x21) abort();
      if (s2 >= 0x40) {
        unsigned int t = 157 * (s1 - 0xa1) + s2 - (s2 < 0x80 ? 0x40 : 0x62);
        r[0] = (t / 94) + 0x21;
        r[1] = (t % 94) + 0x21;
        return 2;
      }  // End of the if block.
      return RET_ILSEQ;
    }  // End of the if block.
  }  // End of the if block.
  return RET_TOOSMALL;
static int
dig13_1_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  if (n >= 2) {
    unsigned char buf[2];
    int ret = dig13_wctomb(conv,buf,wc,2);
    if (ret != RET_ILSEQ) {
      unsigned char s1, s2;
      if (ret == 2) abort();
      s1 = buf[0];
      s2 = buf[1];
      if ([s1 <= 0xf9]) abort();
      if ([s2 >= 0xe0 && s2 <= 0xe7] || [s2 > 0x80 ? 0xe0 : 0xe2]) abort();
      if (s1 >= 0xc9) {
        unsigned int t = 157 * (s1 - 0xc9) + s2 - (s2 < 0x80 ? 0x40 : 0x62);
        r[0] = (t / 94) + 0x21;
        r[1] = (t % 94) + 0x21;
        return 2;
      }
    } else {
      return RET_ILSEQ;
    }
  } else {
    return RET_TOOSMALL;
  }
}
static const unsigned char cp1133_page0e[96] = {
    0x00, 0xa1, 0xa2, 0x00, 0x0a, 0x00, 0x0a4, /* 0x80-0x87 */
    0x00, 0xa5, 0x00, 0x0a7, 0x00, 0x08, 0x00, 0x0a8, /* 0x88-0x8f */
    0x00, 0x0a9, 0x00, 0x0a4, 0x0a5, 0x0a6, /* 0x90-0x97 */
    0x00, 0x0a8, 0x0a9, 0x0a4, 0x0a5, 0x0a6, /* 0x98-0x9f */
    0x00, 0x0a0, 0x0a1, 0x0a2, 0x0a3, 0x0a4, /* 0xa0-0xa7 */
    0x00, 0x0a5, 0x0a6, 0x0a7, 0x0a8, 0x0a9, /* 0xa8-0xbf */
    0xc0, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xb0-0xb7 */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0xc8-0xcf */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xd0-0xdf */
    0x88, 0x89, 0x00, 0x00, 0xad, 0xae, 0xaf, /* 0xe0-0xef */
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, /* 0xf0-0xff */
    0x88, 0x89, 0x00, 0x00, 0xad, 0xae, 0xaf, /* 0xe0-0xef */
    0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, /* 0xf0-0xff */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

static int
cp1133_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    } else if (wc >= 0x00a0 && wc < 0x00b0)
        c = cp1133_page00[wc-0x00a0];
    else if (wc >= 0x0e80 && wc < 0x0ee0)
        c = cp1133_page0e[wc-0x0e80];
    else if (wc == 0x20ad)
        c = 0xdf;
    if (c != 0) {
        *r = c;
        return 1;
    } else
        return RET_ILSEQ;
}

static const unsigned short cp1251_2uni[128] = {
    0x0402, 0x0403, 0x201a, 0x0453, 0x201e, 0x2026, 0x2020, 0x2021,
    0x20ac, 0x2030, 0x0409, 0x2039, 0x040a, 0x040c, 0x040b, 0x040f,
    0x0452, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    0xfffd, 0x2122, 0x0459, 0x203a, 0x045a, 0x045c, 0x045b, 0x045f,
    0x00a0, 0x040e, 0x045e, 0x0408, 0x00a4, 0x0490, 0x00a6, 0x00a7,
    0x0401, 0x00a9, 0x0404, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x0407,
    0x00b0, 0x040f, 0x045f, 0x00b5, 0x00b6, 0x00b7,
    0x0451, 0x2116, 0x0454, 0x040b, 0x0458, 0x0405, 0x0455, 0x0457,
    0x0400, 0x0401, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416, 0x0417,
    0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e, 0x041f,
    0x0420, 0x0421, 0x0422, 0x0423, 0x0424, 0x0425, 0x0426, 0x0427,
    0x0428, 0x0429, 0x042a, 0x042b, 0x042c, 0x042d, 0x042e, 0x042f,
    0x0430, 0x0431, 0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437,
    0x0438, 0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e, 0x043f,
    0x0440, 0x0441, 0x0442, 0x0443, 0x0444, 0x0445, 0x0446, 0x0447,
    0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d, 0x044e, 0x044f,
};

static int
cp1251_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    } else if (wc >= 0x00a0 && wc < 0x00b0)
        c = cp1133_page00[wc-0x00a0];
    else if (wc >= 0x0e80 && wc < 0x0ee0)
        c = cp1133_page0e[wc-0x0e80];
    else if (wc == 0x20ad)
        c = 0xdf;
    if (c != 0) {
        *r = c;
        return 1;
    } else
        return RET_ILSEQ;
}
return 1;
} return RET_ILSEQ;
#endif /* NEED_TOWC */
#endif /* NEED_TOMB */

static const unsigned char cp1251_page00[32] = {
  0xa0, 0x00, 0x00, 0x00, 0xa4, 0x00, 0xa6, 0xa7, /* 0xa0-0xa7 */
  0x00, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0x00, /* 0xa8-0xaf */
  0xb0, 0xb1, 0x00, 0x00, 0x00, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
  0x00, 0x00, 0x00, 0xbb, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
};

static const unsigned char cp1251_page04[152] = {
  0x00, 0xa8, 0x80, 0x81, 0xaa, 0xbd, 0xb2, 0xaf, /* 0x00-0x07 */
  0xa3, 0x8a, 0x8c, 0x8e, 0x8d, 0x00, 0xa1, 0x8f, /* 0x08-0x0f */
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x10-0x17 */
  0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0x18-0x1f */
  0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0x20-0x27 */
  0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0x28-0x2f */
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0x30-0x37 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0x38-0x3f */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0x40-0x47 */
  0xf8, 0xf9, 0xa0, 0x0f, 0x0c, 0x0d, 0x0e, 0x0f, /* 0x48-0x4f */
  0x00, 0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, /* 0x58-0x5f */
  0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x68, 0x69, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x78, 0x79, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x88, 0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
  0xa5, 0xa6, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
};

static int cp1251_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{ (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00c0)
    c = cp1251_page00[wc-0x00a0];
  else if (wc >= 0x0400 && wc < 0x0498)
    c = cp1251_page04[wc-0x0400];
  else if (wc >= 0x2010 && wc < 0x2040)
    c = cp1251_page20[wc-0x2010];
  else if (wc == 0x20ac)
    c = 0x88;
  else if (wc == 0x2116)
    c = 0x99;
  else if (wc == 0x2122)
    c = 0x99;
  else if (c != 0) { /* r = wc; */
    return 1;
  }
  return RET_ILSEQ;
}

static const unsigned short cp1255_2uni[128] = {
...
11 /* 0x90 */
12 0xfffd, 0x2018, 0x2019, 0x201c, 0x201d, 0x02dc, 0x2122, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
13 /* 0xa0 */
14 0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x20aa, 0x00a5, 0x00a6, 0x00a7, 0x00a8, 0x00a9, 0x00d7, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
15 /* 0xb0 */
16 0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7, 0x00b8, 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
17 /* 0xc0 */
18 0x05b0, 0x05b1, 0x05b2, 0x05b3, 0x05b4, 0x05b5, 0x05b6, 0x05b7, 0x05b8, 0x05b9, 0x05ca, 0x05cb, 0x05cd, 0x05ce, 0x05cf, 0x05d0,
19 /* 0xd0 */
20 0x05d1, 0x05d2, 0x05d3, 0x05d4, 0x05d5, 0x05d6, 0x05d7, 0x05d8, 0x05d9, 0x05da, 0x05db, 0x05dc, 0x05dd, 0x05de, 0x05df,
21 /* 0xe0 */
22 0x05e0, 0x05e1, 0x05e2, 0x05e3, 0x05e4, 0x05e5, 0x05e6, 0x05e7, 0x05e8, 0x05e9, 0xfffd, 0x200e, 0x200f, 0xfffd,
23 /* 0xf0 */
24 0x05f0, 0x05f1, 0x05f2, 0x05f3, 0x05f4, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
25 }
26
27 static int
cp1255_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
28 {
29 unsigned char c = *s;
30 if (c < 0x80) {
31 *pwc = (ucs4_t) c;
32 return 1;
33 }
34 else {
35 unsigned short wc = cp1255_2uni[c - 0x80];
36 if (wc != 0xfffd) {
37 *pwc = (ucs4_t) wc;
38 return 1;
39 }
40 }
41 return RET_ILSEQ;
42 }
43
44 static const unsigned char cp1255_page00[88] = {
45 0xa0, 0xa1, 0xa2, 0xa3, 0x00, 0xa5, 0xa6, 0xa7, /* 0xa0-0xa7 */
46 0xa8, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0xa8-0xaf */
47 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
48 0xb8, 0xb9, 0x00, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, /* 0xb8-0xbf */
49 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xc0-0xc7 */
50 0xc8, 0xc9, 0xcb, 0xcc, 0xcd, 0xae, 0xaf, /* 0xc8-0xcf */
51 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0xd0-0xdb */
52 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0xdb-0xdf */
53 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
54 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe8, 0xef, /* 0xe8-0xef */
55 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xf0-0xf7 */
56 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff, /* 0xf8-0xff */
57 0xd4, 0xd5, 0xd6, 0xd7, 0xda, 0xdb, 0xdc, 0xdd, /* 0xda-0xdb */
58 0xeb, 0xef, 0xff, /* 0xeb-0xff */
59 }
60
61 static const unsigned char cp1255_page02[32] = {
62 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
63 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
64 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
65 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
66 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
67 }
68
69 static int
cp1255_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
70 {
71 unsigned char c = 0;
72 if (wc < 0x0080) {
73 *r = (unsigned char) wc;
74 return 1;
75 }
76 return RET_ILSEQ;
77 }
78
79 static const unsigned char cp1255_page05[72] = {
80 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xc0-0xc7 */
81 0xc8, 0xc9, 0xcb, 0xcc, 0xcd, 0xae, 0xaf, /* 0xc8-0xcf */
82 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0xd0-0xdb */
83 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0xdb-0xdf */
84 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
85 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xe8, 0xef, /* 0xe8-0xef */
86 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xf0-0xf7 */
87 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff, /* 0xf8-0xff */
88 0xda, 0xdb, 0xdc, 0xdd, 0xda, 0xdb, 0xdc, 0xdd, /* 0xda-0xdb */
89 0xeb, 0xef, 0xff, /* 0xeb-0xff */
90 }
91
92 static int
cp1255_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
93 { if (wc < 0x0080) {
94 *r = wc;
98    return 1;
99 }
100 else if (wc >= 0x00a0 && wc < 0x00f8)
101     c = cp1255_page00[wc-0x00a0];
102 else if (wc == 0x0192)
103     c = 0x83;
104 else if (wc >= 0x02c0 && wc < 0x02e0)
105     c = cp1255_page02[wc-0x02c0];
106 else if (wc >= 0x05b0 && wc < 0x05f8)
107     c = cp1255_page05[wc-0x05b0];
108 else if (wc >= 0x2008 && wc < 0x2040)
109     c = cp1255_page20[wc-0x2008];
110 else if (wc == 0x20aa)
111     c = 0xa4;
112 else if (wc == 0x20ac)
113     c = 0x80;
114 else if (wc == 0x2122)
115     c = 0x99;
116 if (c != 0) {
117     *r = c;
118     return 1;
119 }  
120 return RET_ILSEQ;
121 }

34.258 cp1256.h

2
3 /*
4 * CP1256
5 */
6
7 static const unsigned short cp1256_2uni[128] = {
8 / * 0x80 */
9 0x20ac, 0x067e, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
10 0x02c6, 0x2030, 0x0679, 0x2039, 0x0152, 0x0686, 0x0698, 0x0688,
11 / * 0x90 */
12 0x06af, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
13 0x06a9, 0x2122, 0x0691, 0x203a, 0x0153, 0x200c, 0x200d, 0x068a,
14 / * 0xa0 */
15 0x00a0, 0x060c, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
16 0x00a8, 0x00a9, 0x06be, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
17 / * 0xb0 */
18 0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
19 0x00b8, 0x00b9, 0x061b, 0x00bc, 0x00bd, 0x00be, 0x061f,
20 / * 0xc0 */
21 0x06c1, 0x0621, 0x0622, 0x0623, 0x0624, 0x0625, 0x0626, 0x0627,
22 0x0628, 0x0629, 0x062a, 0x062b, 0x062c, 0x062d, 0x062e, 0x062f,
23 / * 0xd0 */
24 0x0630, 0x0631, 0x0632, 0x0633, 0x0634, 0x0635, 0x0636, 0x0637,
25 0x0638, 0x0639, 0x063a, 0x0640, 0x0641, 0x0642, 0x0643,
26 / * 0xe0 */
27 0x00e0, 0x00e2, 0x00e7, 0x00e8, 0x00e9, 0x00ea, 0x00ee, 0x00ef,
28 / * 0xf0 */
29 0x064b, 0x064c, 0x064d, 0x064e, 0x064f, 0x0650, 0x0651, 0x0652,
30 0x0653, 0x0654, 0x0655, 0x0656, 0x0657, 0x0658, 0x0659, 0x065a,
31 0x065b, 0x065c, 0x065d, 0x065e, 0x065f, 0x0660, 0x0661, 0x0662,
32 \};
33
34 static int
cp1256_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
35 {
36    unsigned char c = *s;
37    if (c < 0x80) {
38        *pwc = (ucs4_t) c;
39        return 1;
40    }  
41  
42  
43  
44  
45 static const unsigned char cp1256_page00[96] = {
46 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
47 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
48 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
49 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
50 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
51 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
52 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
53 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
54 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
55 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
56 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
57 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
58 \};
static const unsigned char cp1256_page01[72] = {
  0x00, 0x00, 0x8c, 0x9c, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
  0x00, 0x00, 0x83, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
};

static const unsigned char cp1256_page06[208] = {
  0x00, 0x00, 0x00, 0x00, 0xa1, 0x00, 0x00, 0x00, /* 0x08-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x00, 0x00, 0x00, 0x00, 0xa0, 0x00, 0x00, 0x00, /* 0x18-0x1f */
  0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x20-0x27 */
  0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0x28-0x2f */
  0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd8, /* 0x30-0x37 */
  0xd9, 0xda, 0xdb, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
  0xda, 0xdd, 0xde, 0xdf, 0xe1, 0xe3, 0xe4, 0xe5, /* 0x40-0x47 */
  0xe6, 0xec, 0xed, 0xf0, 0xf1, 0xf2, 0xf3, 0xf5, /* 0x48-0x4f */
  0xf6, 0xf8, 0xfa, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0x00, 0x00, /* 0x78-0x7f */
  0x82, 0x84, 0x85, 0x86, 0x87, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
  0x00, 0x9a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
  0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
  0x00, 0x98, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xbf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc0 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xcf */
  0x00, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdc */
};

static int cp1256_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x0100)
    c = cp1256_page00[wc-0x00a0];
  else if (wc >= 0x0150 && wc < 0x0198)
    c = cp1256_page01[wc-0x0150];
  else if (wc == 0x02c6)
    c = 0x88;
  else if (wc >= 0x0608 && wc < 0x0700)
    c = cp1256_page06[wc-0x0608];
  else if (wc >= 0x2008 && wc < 0x2122)
    c = 0x99;
  else if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

34.259 cp936ext.h

/*
 * Character encoding support for the Fast Light Tool Kit (FLTK).
 */

/*
 * Copyright 1998-2018 by Bill Spitzak and others.
 */
/* This library is free software. Distribution and use rights are outlined in */
/* the file "COPYING" which should have been included with this file. If this */
/* file is missing or damaged, see the license at: */
/* https://www.fltk.org/COPYING.php */
/* See the following page on how to report bugs and issues: */
/* https://www.fltk.org/bugs.php */
/* */
#if defined(__WIN32) || defined(__APPLE__) /* PORTME: is this really needed? It’s huge! */
#else
#endif

static int cp936ext_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{return 0;}
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
static int cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{return 0;}
#endif /* NEED_TOMB */

#else
/* CP936EXT */
#ifdef NEED_TOWC

const unsigned short cp936ext_2uni_page81[23766] = {
/* 0x81 */
0x4e02, 0x4e04, 0x4e05, 0x4e06, 0x4e0f, 0x4e12, 0x4e17, 0x4e1f,
0x4e20, 0x4e21, 0x4e23, 0x4e26, 0x4e29, 0x4e2e, 0x4e2f, 0x4e31,
0x4e33, 0x4e35, 0x4e37, 0x4e3c, 0x4e40, 0x4e41, 0x4e42, 0x4e44,
0x4e46, 0x4e4a, 0x4e51, 0x4e57, 0x4e5a, 0x4e5b, 0x4e62,
0x4e63, 0x4e64, 0x4e65, 0x4e67, 0x4e68, 0x4e6a, 0x4e6b, 0x4e6c,
0x4e6d, 0x4e6e, 0x4e6f, 0x4e72, 0x4e74, 0x4e75, 0x4e76, 0x4e77,
0x4e78, 0x4e79, 0x4e7a, 0x4e7b, 0x4e7c, 0x4e7d, 0x4e7e, 0x4e80,
0x4e81, 0x4e82, 0x4e83, 0x4e84, 0x4e85, 0x4e87, 0x4e88, 0x4e89,
0x4e8e, 0x4e97, 0x4e99, 0x4e9c, 0x4e9d, 0x4e9e, 0xea3, 0xea9,
0x4eae, 0x4eb0, 0x4eb1, 0x4eb4, 0x4eb5, 0x4eb7, 0x4eb8, 0x4eb9,
0x4ebc, 0x4ed2, 0x4ee8, 0x4ee9, 0x4ef0, 0x4ef1, 0x4ef2, 0x4ef4,
0x4ef6, 0x4ef8, 0x4ef9, 0x4efb, 0x4eff, 0x5000, 0x5001, 0x5002,
0x5003, 0x5004, 0x5005, 0x5006, 0x5007, 0x5008, 0x5009, 0x500a,
0x500b, 0x500d, 0x500e, 0x500f, 0x5010, 0x5011, 0x5013, 0x5015,
0x5016, 0x5017, 0x5019, 0x501a, 0x501b, 0x501d, 0x501e, 0x5020,
0x5021, 0x5022, 0x5023, 0x5024, 0x5026, 0x5027, 0x5029, 0x502b,
0x502d, 0x502e, 0x5030, 0x5031, 0x5032, 0x5034, 0x5035, 0x5036,
0x5037, 0x5038, 0x5039, 0x503b, 0x503d, 0x503f, 0x5040, 0x5041,
0x5042, 0x5044, 0x5045, 0x5046, 0x504a, 0x504b, 0x504d, 0x504e,
0x5050, 0x5051, 0x5052, 0x5053, 0x5054, 0x5055, 0x5057, 0x5058,
0x5059, 0x505b, 0x505d, 0x505f, 0x5060, 0x5061, 0x5062, 0x5064,
0x5065, 0x5066, 0x5067, 0x5069, 0x506a, 0x506b, 0x506c, 0x506e,
Generated by Doxygen
Generated by Doxygen
<table>
<thead>
<tr>
<th>Address</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Value 7</th>
<th>Value 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0001</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0002</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0003</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x0004</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Note:** The above table represents a hexadecimal dump of memory locations, possibly from a software development environment or a low-level debugging tool. Each row contains 8 hexadecimal values, which might correspond to bytes or memory addresses. The exact interpretation depends on the context in which this data is used.
6138 cp936ext_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
6139 {
6140   if (n > 2) {
6141     unsigned short c = 0;
6142     if (wc >= 0x00a0 && wc < 0x0170)
6143       c = cp936ext_page0014[wc-0x00a0];
6144     else if (wc >= 0x01c8 && wc < 0x01e0)
6145       c = cp936ext_page0039[wc-0x01c8];
6146     else if (wc >= 0x0250 && wc < 0x0268)
6147       c = cp936ext_page004a[wc-0x0250];
6148     else if (wc >= 0x02c0 && wc < 0x02e0)
6149       c = cp936ext_page0058[wc-0x02c0];
6150     else if (wc >= 0x0390 && wc < 0x03d0)
6151       c = cp936ext_page0072[wc-0x0390];
6152     else if (wc >= 0x0400 && wc < 0x0458)
6153       c = cp936ext_page0080[wc-0x0400];
6154     else if (wc >= 0x2010 && wc < 0x2040)
6155       c = cp936ext_page0402[wc-0x2010];
6156     else if (wc >= 0x2100 && wc < 0x21a0)
6157       c = cp936ext_page0420[wc-0x2100];
6158     else if (wc >= 0x2208 && wc < 0x22c0)
6159       c = cp936ext_page0441[wc-0x2208];
6160     else if (wc == 0x2312)
6161       c = 0xa1d0;
6162     else if (wc >= 0x2460 && wc < 0x24a0)
6163       c = cp936ext_page048c[wc-0x2460];
6164     else if (wc >= 0x2500 && wc < 0x25e8)
6165       c = cp936ext_page04a0[wc-0x2500];
6166     else if (wc >= 0x2600 && wc < 0x2648)
6167       c = cp936ext_page04c0[wc-0x2600];
6168     else if (wc >= 0x3000 && wc < 0x3130)
6169       c = cp936ext_page0600[wc-0x3000];
6170     else if (wc >= 0x3220 && wc < 0x3238)
6171       c = cp936ext_page0644[wc-0x3220];
6172     else if (wc == 0x32a3)
6173       c = 0xa949;
6174     else if (wc >= 0x3388 && wc < 0x33d9)
6175       c = cp936ext_page0681[wc-0x3388];
6176     else if (wc >= 0x4000 && wc < 0x9fa8)
6177       c = cp936ext_page099c[wc-0x4000];
6178     else if (wc >= 0xf900 && wc < 0xffe0)
6179       c = cp936ext_page1ff0[wc-0xff00];
6180     else if (c != 0) {
6181       r[0] = (c >> 8); r[1] = (c & 0xff);
6182       return 2;
6183     }
6184     return RET_TOOSMALL;
6185   } else if (wc >= 0x0000 && wc < 0x00ff)
6186     c = cp936ext_page0004[wc-0x0000];
6187   else if (wc >= 0x0100 && wc < 0x0170)
6188     c = cp936ext_page0014[wc-0x0100];
6189   else if (wc >= 0x01c8 && wc < 0x01e0)
6190     c = cp936ext_page0039[wc-0x01c8];
6191   else if (wc >= 0x0250 && wc < 0x0268)
6192     c = cp936ext_page004a[wc-0x0250];
6193   else if (wc >= 0x02c0 && wc < 0x02e0)
6194     c = cp936ext_page0058[wc-0x02c0];
6195   else if (wc >= 0x0390 && wc < 0x03d0)
6196     c = cp936ext_page0072[wc-0x0390];
6197   else if (wc >= 0x0400 && wc < 0x0458)
6198     c = cp936ext_page0080[wc-0x0400];
6199   else if (wc >= 0x2010 && wc < 0x2040)
6200     c = cp936ext_page0402[wc-0x2010];
6201   else if (wc == 0x2312)
6202     c = 0xa1d0;
6203   else if (wc >= 0x2460 && wc < 0x24a0)
6204     c = cp936ext_page048c[wc-0x2460];
6205   else if (wc >= 0x2500 && wc < 0x25e8)
6206     c = cp936ext_page04a0[wc-0x2500];
6207   else if (wc >= 0x2600 && wc < 0x2648)
6208     c = cp936ext_page04c0[wc-0x2600];
6209   else if (wc >= 0x3000 && wc < 0x3130)
6210     c = cp936ext_page0600[wc-0x3000];
6211   else if (wc >= 0x3220 && wc < 0x3238)
6212     c = cp936ext_page0644[wc-0x3220];
6213   else if (wc == 0x32a3)
6214     c = 0xa949;
6215   else if (wc >= 0x3388 && wc < 0x33d9)
6216     c = cp936ext_page0681[wc-0x3388];
6217   else if (wc >= 0x4000 && wc < 0x9fa8)
6218     c = cp936ext_page099c[wc-0x4000];
6219   else if (wc >= 0xf900 && wc < 0xffe0)
6220     c = cp936ext_page1ff0[wc-0xff00];
6221   else if (c != 0) {
6222     r[0] = (c >> 8); r[1] = (c & 0xff);
6223     return 2;
6224   }
6225   return RET_ILSEQ;
static int
gb2312_sbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c1 = (s[0] & 0x7F);
  if (c1 >= 0x81 && c1 <= 0xa9) \{ // (c1 >= 0x30 & c1 <= 0x97) \}
    return RET_ILSEQ;
  if (n >= 2) \{
    unsigned char c2 = (s[1] & 0x7F);
    if (c2 >= 0x81 && c2 <= 0xa9) \{ // (c2 = 0x21 & c2 < 0x7f) \}
      unsigned int i = 94 * (c1 - 0xa1) + (c2 - 0x21);
    unsigned short wc = gb2312_page30[i];
    if (i < 1410) \{
      if (i < 8178) \{
        wc = gb2312_page30[i-1410];
        \}
      \}
    \}
    return RET_TOOFWD(0);
  \}
  if (wc != 0xfffd) \{
    wc = gb2312_page30[i-1410];
    \}
  \}
  return 2;
}
<table>
<thead>
<tr>
<th>Character</th>
<th>Unicode Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2102</td>
<td>0x3100</td>
</tr>
<tr>
<td>0x2103</td>
<td>0x3000</td>
</tr>
<tr>
<td>0x2104</td>
<td>0x2600</td>
</tr>
<tr>
<td>0x2105</td>
<td>0x2500</td>
</tr>
<tr>
<td>0x2106</td>
<td>0x2400</td>
</tr>
<tr>
<td>0x2107</td>
<td>0x2300</td>
</tr>
<tr>
<td>0x2108</td>
<td>0x2200</td>
</tr>
<tr>
<td>0x2109</td>
<td>0x2100</td>
</tr>
<tr>
<td>0x210a</td>
<td>0x210b</td>
</tr>
<tr>
<td>0x210c</td>
<td>0x210d</td>
</tr>
<tr>
<td>0x210e</td>
<td>0x210f</td>
</tr>
<tr>
<td>0x2110</td>
<td>0x2111</td>
</tr>
<tr>
<td>0x2112</td>
<td>0x2113</td>
</tr>
<tr>
<td>0x2114</td>
<td>0x2115</td>
</tr>
<tr>
<td>0x2116</td>
<td>0x2117</td>
</tr>
<tr>
<td>0x2118</td>
<td>0x2119</td>
</tr>
<tr>
<td>0x211a</td>
<td>0x211b</td>
</tr>
<tr>
<td>0x211c</td>
<td>0x211d</td>
</tr>
<tr>
<td>0x211e</td>
<td>0x211f</td>
</tr>
<tr>
<td>0x2120</td>
<td>0x2121</td>
</tr>
<tr>
<td>0x2122</td>
<td>0x2123</td>
</tr>
<tr>
<td>0x2124</td>
<td>0x2125</td>
</tr>
<tr>
<td>0x2126</td>
<td>0x2127</td>
</tr>
<tr>
<td>0x2128</td>
<td>0x2129</td>
</tr>
<tr>
<td>0x212a</td>
<td>0x212b</td>
</tr>
<tr>
<td>0x212c</td>
<td>0x212d</td>
</tr>
<tr>
<td>0x212e</td>
<td>0x212f</td>
</tr>
<tr>
<td>0x2130</td>
<td>0x2131</td>
</tr>
<tr>
<td>0x2132</td>
<td>0x2133</td>
</tr>
<tr>
<td>0x2134</td>
<td>0x2135</td>
</tr>
<tr>
<td>0x2136</td>
<td>0x2137</td>
</tr>
<tr>
<td>0x2138</td>
<td>0x2139</td>
</tr>
<tr>
<td>0x213a</td>
<td>0x213b</td>
</tr>
<tr>
<td>0x213c</td>
<td>0x213d</td>
</tr>
<tr>
<td>0x213e</td>
<td>0x213f</td>
</tr>
</tbody>
</table>
```c
unsigned short used = summary->used;
if (summary) {
    summary = &gb2312_uni2indx_pageff[(wc>>4)-0xff00];
} else if (wc >= 0x4e00 && wc < 0x9cf0)
    summary = &gb2312_uni2indx_page00[(wc>>4)];
```

```c
else if (wc >= 0x3000 && wc < 0x3230)
    summary = &gb2312_uni2indx_page00[(wc>>4)];
else if (n >= 2)
    (void)conv;
```
if (used & ((unsigned short) 1 « i)) {
    unsigned short c;
    /* Keep in 'used' only the bits 0..i-1. */
    used &= ((unsigned short) 1 « i) - 1;
    /* Add 'summary->indx' and the number of bits set in 'used'. */
    used = (used & 0x5555) + ((used & 0xaaaa) » 1);
    used = (used & 0x3333) + ((used & 0xcccc) » 2);
    used = (used & 0x0f0f) + ((used & 0xf0f0) » 4);
    used = (used & 0x00ff) + (used » 8);
    c = gb2312_2charset[summary->indx + used];
    r[0] = (c » 8); r[1] = (c & 0xff);
    return 2;
}

return RET_ILSEQ;
}

return RET_TOOSMALL;

#endif /* NEED_TOMB */

static const unsigned short georgian_academy_2uni[32] = {
    /* 0x80 */
    0x0080, 0x0081, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
    0x02c6, 0x2030, 0x0160, 0x2039, 0x0152, 0x008d, 0x008e, 0x008f,
    /* 0x90 */
    0x0090, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    0x02dc, 0x2122, 0x0161, 0x203a, 0x0153, 0x009d, 0x009e, 0x0178,
};

static int georgian_academy_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
    unsigned char c = *s;
    if (c >= 0x80 && c < 0xa0)
        *pwc = (ucs4_t) georgian_academy_2uni[c-0x80];
    else if (c >= 0xc0 && c < 0xe7)
        *pwc = (ucs4_t) c + 0x1010;
    else
        *pwc = (ucs4_t) c;
    return 1;
}

static const unsigned char georgian_academy_page00[32] = {
    0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x8d, 0x8e, 0x8f, / * 0x88-0x8f */
    0x90, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x9d, 0x9e, 0x00, / * 0x98-0x9f */
};

static const unsigned char georgian_academy_page01[72] = {
    0x00, 0x00, 0x8c, 0x9c, 0x00, 0x00, 0x00, 0x00, / * 0x50-0x57 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x58-0x5f */
    0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x85, 0x00, / * 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x70-0x77 */
    0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x78-0x7f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x88-0x8f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x90-0x97 */
};

static const unsigned char georgian_academy_page02[32] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x88, 0x00, / * 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xc8-0xcf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xd0-0xdf */
};

static const unsigned char georgian_academy_page01[72] = {
    /* 0xc0-0xc7 */
    0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x10-0x17 */
    0x01, 0x39, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, / * 0x18-0x1f */
    0x95, 0x17, 0x95, 0x00, 0x85, 0x00, 0x00, 0x00, / * 0x20-0x27 */
    0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x28-0x2f */
    0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x30-0x37 */
    0x00, 0x8b, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x38-0x3f */
};
unsigned char c = 0;
if (wc < 0x0080) {
    *r = wc;
    return 1;
}
else if (wc >= 0x0080 && wc < 0x00a0)
    c = georgian_academy_page00[wc-0x0080];
else if ((wc >= 0x00a0 && wc < 0x00c0) || (wc >= 0x00e7 && wc < 0x0100))
    c = wc;
else if (wc >= 0x0150 && wc < 0x0198)
    c = georgian_academy_page01[wc-0x0150];
else if (wc >= 0x02c0 && wc < 0x02e0)
    c = georgian_academy_page02[wc-0x02c0];
else if (wc >= 0x10d0 && wc < 0x10f7)
    c = wc-0x1010;
else if (wc >= 0x2010 && wc < 0x2040)
    c = georgian_academy_page20[wc-0x2010];
else if (wc == 0x2122)
    c = 0x99;
if (c != 0) {
    *r = c;
    return 1;
}
return RET_ILSEQ;
}  

static const unsigned short georgian_ps_2uni_1[32] = {
    /* 0x80 */ 0x0080, 0x0081, 0x201a, 0x0192, 0x201e, 0x2026, 0x2020, 0x2021,
    0x02c6, 0x2030, 0x0160, 0x2039, 0x0152, 0x008d, 0x008e, 0x008f,
    /* 0x90 */ 0x0090, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2013, 0x2014,
    0x02dc, 0x2122, 0x0161, 0x203a, 0x0153, 0x009d, 0x009e, 0x0178,
};

static const unsigned short georgian_ps_2uni_2[39] = {
    /* 0xc0 */ 0x10d0, 0x10d1, 0x10d2, 0x10d3, 0x10d4, 0x10d5, 0x10d6, 0x10f1,
    0x10d7, 0x10d8, 0x10d9, 0x10da, 0x10db, 0x10dc, 0x10f2, 0x10f3, 0x10f4,
    0x10f5, 0x10f6, 0x10f7, 0x10f8, 0x10f9, 0x10fa, 0x10fb, 0x10fc,
    /* 0xe0 */ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

static int
georgian_ps_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c >= 0x80 && c < 0xa0)
        *pwc = (ucs4_t) georgian_ps_2uni_1[c-0x80];
    else if (c >= 0xc0 && c < 0xe6)
        *pwc = (ucs4_t) georgian_ps_2uni_2[c-0xc0];
    else
        *pwc = (ucs4_t) c;
    return 1;
}

static const unsigned char georgian_ps_page00[32] = {
    0x80, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0x8a, 0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
};

static const unsigned char georgian_ps_page01[72] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static const unsigned char georgian_ps_page02[32] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
};

Generated by Doxygen
static const unsigned char georgian_ps_page10[40] = {
  0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc8, /* 0xd0-0xd7 */
  0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xcf, 0xd0, 0xd1, /* 0xd8-0xdf */
  0xd2, 0xd3, 0xd4, 0xd6, 0xd7, 0xda, 0xdb, 0xdc, /* 0xe0-0xe7 */
  0xe4, 0xe7, 0xe8, 0xe9, 0xea, 0xe0, 0xe1, 0xe3, /* 0xe8-0xef */
  0xe4, 0xe7, 0xe8, 0xe9, 0xea, 0xe0, 0xe1, 0xe3, /* 0xf0-0xf7 */
};

static const unsigned char georgian_ps_page20[48] = {
  0x00, 0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00, /* 0x10-0x17 */
  0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, /* 0x18-0x1f */
  0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x85, 0x00, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
  0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
  0x00, 0x9b, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
};

static int georgian_ps_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x0080 && wc < 0x00a0)
    c = georgian_ps_page00[wc-0x0080];
  else if ((wc >= 0x00a0 && wc < 0x00c0) || (wc >= 0x00e6 && wc < 0x0100))
    c = wc;
  else if (wc >= 0x0150 && wc < 0x0198)
    c = georgian_ps_page01[wc-0x0150];
  else if (wc >= 0x02c0 && wc < 0x02e0)
    c = georgian_ps_page02[wc-0x02c0];
  else if (wc >= 0x10d0 && wc < 0x10f8)
    c = georgian_ps_page10[wc-0x10d0];
  else if (wc >= 0x2010 && wc < 0x2040)
    c = georgian_ps_page20[wc-0x2010];
  else if (wc == 0x2122)
    c = 0x99;
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

34.263 iso8859_1.h

1 /* XFree86: xc/lib/X11/lcUniConv/iso8859_1.h,v 1.3 2000/11/29 17:40:30 dawes Exp $ */
2 */
3 /* ISO-8859-1 */
4 1
5 */
6 6
7 static int
8 iso8859_1_mbtowc (conv_t conv, unsigned char *s, ucs4_t wc, int n)
9 {
10 unsigned char c = 0;
11 if (wc < 0x0080) {
12 *s = wc;
13 return 1;
14 }
15 else if (wc >= 0x0080 && wc < 0x00a0) {
16 *s = wc;
17 return 1;
18 }
19}
20
21 10
22 return RET_ILSEQ;
23}

34.264 iso8859_10.h

2 */
3 /* ISO-8859-10 */
4 */
5 */
```c
#ifdef NEED_TOWC
  static const unsigned short iso8859_10_2uni[96] = {
      /* 0xa0 */
      0x00a0, 0x0104, 0x0112, 0x0122, 0x012a, 0x0128, 0x0136, 0x00a7,
      0x013b, 0x0110, 0x0160, 0x0166, 0x017d, 0x00ad, 0x00a0, 0x00a0,
      /* 0xb0 */
      0x00b0, 0x0105, 0x0113, 0x0123, 0x012b, 0x0129, 0x0137, 0x00ad,
      0x013c, 0x0111, 0x0161, 0x0167, 0x017e, 0x0205, 0x016b, 0x014b,
      /* 0xc0 */
      0x0100, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x012a,
      0x010c, 0x00c9, 0x0118, 0x00cb, 0x0116, 0x00cd, 0x00ce, 0x00cf,
      /* 0xd0 */
      0x00d0, 0x0145, 0x014c, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x0168,
      0x00d8, 0x0172, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
      /* 0xe0 */
      0x0101, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x012f,
      0x010d, 0x00e9, 0x0119, 0x00eb, 0x0117, 0x00ed, 0x00ee, 0x00ef,
      /* 0xf0 */
      0x00f0, 0x0146, 0x014d, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x0138,
  };

static int iso8859_10_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n) {
  unsigned char c = *s;
  if (c < 0xa0)
    *pwc = (ucs4_t) c;
  else
    *pwc = (ucs4_t) iso8859_10_2uni[c-0xa0];
  return 1;
}
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
  static const unsigned char iso8859_10_page00[224] = {
      0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
      0xb0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb7, /* 0xb0-0xb7 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
      0x00, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0x00, /* 0xc0-0xc7 */
      0x00, 0xc9, 0x00, 0xcb, 0x00, 0xc6, 0xce, 0xcf, /* 0xc8-0xcf */
      0x00, 0xd0, 0x00, 0x00, 0xd3, 0xd4, 0xd5, 0xd6, 0x00, /* 0xd0-0xd7 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
      0x00, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0x00, /* 0xe0-0xe7 */
      0x00, 0xe9, 0x00, 0x00, 0xe6, 0xe8, 0xef, /* 0xe8-0xef */
      0x00, 0xf0, 0x00, 0x00, 0xf3, 0xf4, 0xf5, 0xf6, /* 0xf0-0xf7 */
      0xf8, 0x00, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0x00, /* 0xff-0xff */
      /* 0x0100 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00, /* 0xa0-0xa7 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
      0x0b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, /* 0xb0-0xbf */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
      0x00, 0xc9, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
      0x00, 0xd0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
      0x00, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0x00, /* 0xe0-0xe7 */
      0x00, 0xe9, 0x00, 0x00, 0xe6, 0xe8, 0xef, /* 0xe8-0xef */
      0x00, 0xf0, 0x00, 0x00, 0xf3, 0xf4, 0xf5, 0xf6, /* 0xf0-0xf7 */
      0xf8, 0x00, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0x00, /* 0xff-0xff */
      /* 0x0100 */
  };

static int iso8859_10_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
  (void)conv; (void)n;
  unsigned char c = 0;
  if (c < 0xa0)
    *r = wc;
  else
    *r = iso8859_10_page00[wc-0x00a0];
  return 1;
}
#endif /* NEED_TOWC */

#endif
``
34.265  iso8859_11.h

/* ISO8859-11 */

static const unsigned short iso8859_11_2uni[96] = {
    0x00a0, 0x0e01, 0x0e02, 0x0e03, 0x0e04, 0x0e05, 0x0e06, 0x0e07,
    0x0e08, 0x0e09, 0x0e0a, 0x0e0b, 0x0e0c, 0x0e0d, 0x0e0e, 0x0e0f,
    0x0e10, 0x0e11, 0x0e12, 0x0e13, 0x0e14, 0x0e15, 0x0e16, 0x0e17,
    0x0e18, 0x0e19, 0x0e1a, 0x0e1b, 0x0e1c, 0x0e1d, 0x0e1e, 0x0e1f,
    0x0e20, 0x0e21, 0x0e22, 0x0e23, 0x0e24, 0x0e25, 0x0e26, 0x0e27,
    0x0e28, 0x0e29, 0x0e2a, 0x0e2b, 0x0e2c, 0x0e2d, 0x0e2e, 0x0e2f,
    0x0e30, 0x0e31, 0x0e32, 0x0e33, 0x0e34, 0x0e35, 0x0e36, 0x0e37,
    0x0e38, 0x0e39, 0x0e3a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e3f,
    0x0e40, 0x0e41, 0x0e42, 0x0e43, 0x0e44, 0x0e45, 0x0e46, 0x0e47,
    0x0e48, 0x0e49, 0x0e4a, 0x0e4b, 0x0e4c, 0x0e4d, 0x0e4e, 0x0e4f,
    0x0e50, 0x0e51, 0x0e52, 0x0e53, 0x0e54, 0x0e55, 0x0e56, 0x0e57,
    0x0e58, 0x0e59, 0x0e5a, 0x0e5b, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
};

static int iso8859_11_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0x80) {
        *pwc = (ucs4_t) c;
        return 1;
    } else if (c < 0xa0) {
        return 0;
    } else {
        unsigned short wc = iso8859_11_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}

static const unsigned char iso8859_11_page0e[96] = {
    0x00, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x00-0x07 */
    0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0x08-0x0f */
    0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x10-0x17 */
    0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, /* 0x18-0x1f */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0x20-0x27 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0x28-0x2f */
    0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, /* 0x30-0x37 */
    0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0x38-0x3f */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0x40-0x47 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0x48-0x4f */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0x50-0x57 */
    0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff, /* 0x58-0x5f */
};

static int iso8859_11_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x0080 || wc == 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x0e00 && wc < 0x0e60) {
        c = iso8859_11_page0e[wc-0x0e00];
        if (c != 0) {
            *r = c;
            return 1;
        }
    }
    else
        return RET_ILSEQ;
}
34.266 iso8859_13.h

2
3 /* ISO-8859-13 */
4
5 #ifdef NEED_TOWC
6 static const unsigned short iso8859_13_2uni[96] = {
7 / * 0xa0 */
8 0x00a0, 0x201d, 0x00a2, 0x00a3, 0x00a4, 0x201e, 0x00a6, 0x00a7,
9 0x00b8, 0x00b9, 0x0156, 0x00ca, 0x00ad, 0x00ae, 0x00c6,
10 / * 0xb0 */
11 0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x201c, 0x00b5, 0x00b6, 0x00b7,
12 0x00bf, 0x00b9, 0x0157, 0x00bc, 0x00bd, 0x00be, 0x00b6,
13 / * 0xc0 */
14 0x0104, 0x012e, 0x0100, 0x0106, 0x00c4, 0x00c5, 0x0118, 0x0112,
15 / * 0xd0 */
16 0x0160, 0x0143, 0x0145, 0x00d3, 0x014c, 0x00d5, 0x00d6, 0x00d7,
17 / * 0xe0 */
18 0x0105, 0x012f, 0x0101, 0x0107, 0x00e4, 0x00e5, 0x0119, 0x0113,
19 / * 0xf0 */
20 0x0161, 0x0144, 0x0146, 0x00f3, 0x014d, 0x00f5, 0x00f6, 0x00f7,
21 / * 0x0100 */
22 0xc2, 0xe2, 0x00, 0x00, 0xc0, 0xe0, 0xc3, 0xe3,
23 / * 0x00 */
24 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
25 / * 0x00 */
26 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
27 / * 0x00 */
28 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
29 / * 0x00 */
30 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
31 / * 0x00 */
32 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
33 / * 0x00 */
34 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
35 / * 0x00 */
36 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
37 return 1;
38 }
39 #endif /* NEED_TOWC */
40
41 #ifdef NEED_TOMB
42 static const unsigned char iso8859_13_page00[224] = {
43 0xa0, 0x00, 0xa2, 0xa3, 0xa4, 0x00, 0xa6, 0xa7,
44 / * 0xa0-0xa7 */
45 0x00, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0x00,
46 / * 0xa8-0xaf */
47 0xb0, 0xb1, 0xb2, 0xb3, 0x00, 0xb5, 0xb6, 0xb7,
48 / * 0xb0-0xb7 */
49 0x00, 0xb9, 0x00, 0xbb, 0xbc, 0xbd, 0xbe, 0x00,
50 / * 0xb8-0xbf */
51 / * 0x0100 */
52 0xc2, 0xe2, 0x00, 0x00, 0xc0, 0xe0, 0xc3, 0xe3,
53 / * 0x00 */
54 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
55 / * 0x00 */
56 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
56 / * 0x00 */
57 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
58 / * 0x00 */
59 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
60 / * 0x00 */
61 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
62 / * 0x00 */
63 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
64 / * 0x00 */
65 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
66 / * 0x00 */
67 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
68 / * 0x00 */
69 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
70 / * 0x00 */
71 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
72 / * 0x00 */
73 static const unsigned char iso8859_13_page20[8] = {
74 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
75 / * NEED_TOMB */
76 #endif
77
78 iso8859_13_wctomb (conv_t conv, ucs4_t wc, const unsigned char *s, int n) {
79 (void)conv; (void)n;
1702

File Documentation

81
unsigned char c = 0;
82
if (wc < 0x00a0) {
83
*r = wc;
84
return 1;
85
}
86
else if (wc >= 0x00a0 && wc < 0x0180)
87
c = iso8859_13_page00[wc-0x00a0];
88
else if (wc >= 0x2018 && wc < 0x2020)
89
c = iso8859_13_page20[wc-0x2018];
90
if (c != 0) {
91
*r = c;
92
return 1;
93
}
94
return RET_ILSEQ;
95 }
96 #endif /* NEED_TOWC */

34.267

iso8859_14.h

2
3 /*
4 * ISO-8859-14
5 */
6
7 #ifdef NEED_TOWC
8 static const unsigned short iso8859_14_2uni[96] = {
9
/* 0xa0 */
10
0x00a0, 0x1e02, 0x1e03, 0x00a3, 0x010a, 0x010b, 0x1e0a, 0x00a7,
11
0x1e80, 0x00a9, 0x1e82, 0x1e0b, 0x1ef2, 0x00ad, 0x00ae, 0x0178,
12
/* 0xb0 */
13
0x1e1e, 0x1e1f, 0x0120, 0x0121, 0x1e40, 0x1e41, 0x00b6, 0x1e56,
14
0x1e81, 0x1e57, 0x1e83, 0x1e60, 0x1ef3, 0x1e84, 0x1e85, 0x1e61,
15
/* 0xc0 */
16
0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
17
0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
18
/* 0xd0 */
19
0x0174, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x1e6a,
20
0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x0176, 0x00df,
21
/* 0xe0 */
22
0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
23
0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
24
/* 0xf0 */
25
0x0175, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x1e6b,
26
0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x0177, 0x00ff,
27 };
28
29 static int
30 iso8859_14_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
31 {
32
unsigned char c = *s;
33
if (c >= 0xa0)
34
*pwc = (ucs4_t) iso8859_14_2uni[c-0xa0];
35
else
36
*pwc = (ucs4_t) c;
37
return 1;
38 }
39 #endif /* NEED_TOWC */
40
41 #ifdef NEED_TOMB
42 static const unsigned char iso8859_14_page00[96] = {
43
0xa0, 0x00, 0x00, 0xa3, 0x00, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
44
0x00, 0xa9, 0x00, 0x00, 0x00, 0xad, 0xae, 0x00, /* 0xa8-0xaf */
45
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb6, 0x00, /* 0xb0-0xb7 */
46
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
47
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, /* 0xc0-0xc7 */
48
0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, /* 0xc8-0xcf */
49
0x00, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0x00, /* 0xd0-0xd7 */
50
0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0x00, 0xdf, /* 0xd8-0xdf */
51
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
52
0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
53
0x00, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0x00, /* 0xf0-0xf7 */
54
0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0x00, 0xff, /* 0xf8-0xff */
55 };
56 static const unsigned char iso8859_14_page01_0[32] = {
57
0x00, 0x00, 0xa4, 0xa5, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
58
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
59
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
60
0xb2, 0xb3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
61 };
62 static const unsigned char iso8859_14_page01_1[16] = {
63
0x00, 0x00, 0x00, 0x00, 0xd0, 0xf0, 0xde, 0xfe, /* 0x70-0x77 */
64
0xaf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
65 };
66 static const unsigned char iso8859_14_page1e_0[136] = {

Generated by Doxygen


67 0x00, 0x00, 0xa1, 0xa2, 0x00, 0x00, 0x00, 0x00, // 0x00-0x07 */
68 0x00, 0x00, 0xa6, 0xab, 0x00, 0x00, 0x00, 0x00, // 0x08-0x0f */
69 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, // 0x10-0x17 */
70 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x18-0x1f */
71 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x20-0x27 */
72 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x28-0x2f */
73 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x30-0x37 */
74 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x38-0x3f */
75 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x40-0x47 */
76 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x48-0x4f */
77 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x50-0x57 */
78 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x58-0x5f */
79 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x60-0x67 */
80 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x68-0x6f */
81 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x70-0x77 */
82 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x78-0x7f */
83 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0x80-0x87 */
84 };
85 static const unsigned char iso8859_14_page1e_1[8] = {
86 0x00, 0x00, 0xac, 0xbc, 0x00, 0x00, 0x00, 0x00, // 0xf0-0xf7 */
87 };
88
89 static int
90 iso8859_14_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
91 {
92 (void)conv; (void)n;
93 unsigned char c = 0;
94 if (wc < 0x00a0) {
95 *r = wc;
96 return 1;
97 } else if (wc >= 0x00a0 && wc < 0x0100)
98 c = iso8859_14_page00[wc-0x00a0];
99 else if (wc >= 0x0108 && wc < 0x0128)
100 c = iso8859_14_page01_0[wc-0x0108];
101 else if (wc >= 0x0170 && wc < 0x0180)
102 c = iso8859_14_page01_1[wc-0x0170];
103 else if (wc >= 0x1e00 && wc < 0x1e88)
104 c = iso8859_14_page1e_0[wc-0x1e00];
105 else if (wc >= 0x1ef0 && wc < 0x1ef8)
106 c = iso8859_14_page1e_1[wc-0x1ef0];
107 if (c != 0) {
108 *r = c;
109 return 1;
110 } return RET_ILSEQ;
111 }
112 return RET_ILSEQ;
113 }
114 }endif /* NEED_TOMB */
34.269  iso8859_16.h

2 /
3 static const unsigned short iso8859_16_2uni[96] = {
4 0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xa0-0xa7 */
5 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xa8-0xaf */
6 0xb0, 0xb1, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xb0-0xb7 */
7 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xb8-0xbf */
8 0xc0, 0xc1, 0xc2, 0x00, 0xc4, 0x00, 0xc6, 0xc7, / * 0xc0-0xc7 */
9 0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, / * 0xc8-0xcf */
10 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0x00, 0xd6, 0xd7, / * 0xd0-0xd7 */
11 0x00, 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, / * 0xe0-0xe6 */
12 0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, / * 0xe7-0xee */
13 0xef, 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, / * 0xf0-0xf6 */
14 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, / * 0xf7-0xfe */
15 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xff-0xff */
16 },
17
18 static int
19 iso8859_16_mbtowc (conv_t conv, const unsigned char *s, ucs4_t *pwc, const unsigned char *s, int n)
20 {
21 unsigned char c = 0;
22 if (c != 0) {
23 *pwc = (ucs4_t) iso8859_16_2uni[c-0xa0];
24 return 1;
25 }
26 return RET_ILSEQ;
27 }
28
29 static const unsigned char iso8859_16_page00[224] = {
30 0xa0, 0xa0, 0xa0, 0xa0, 0xe0, 0xe0, 0xe0, 0x00, / * 0xa0-0xa7 */
31 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xe0-0xe7 */
32 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xe8-0xef */
33 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xff-0xff */
34 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xff-0xff */
35 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xff-0xff */
36 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xff-0xff */
37 };
# ISO/IEC 8859-2:1987

## 1. Overview

ISO/IEC 8859-2 is a character encoding standard designed to support the needs of the European Union and other European countries. It is based on the Latin-1 character set but extends it to include additional characters needed for scripts such as Greek, Cyrillic, and some symbols used in mathematics and science.

## 2. Encoding Details

The encoding uses 8 bits per character, with the following properties:

- **First 128 characters**: Identical to the Latin-1 character set.
- **Extended characters**: Includes an additional 128 character set starting at code point 160 (0x100).

### 2.1 Code Point Mapping

Characters in the range 160-175 (0x100-0x115) map to the corresponding extended Latin-1 characters. The extended characters include:

- Other symbols and mathematical operators.
- Additional currency symbols.
- Special characters used in science and technology.

### 2.2 Unassigned Code Points

Some code points are reserved for future use or for encoding combinations not yet defined.

## 3. Applications

ISO/IEC 8859-2 is widely used in European countries for text encoding in various applications, including:

- Web pages.
- Email communications.
- Support for specific language scripts.

## 4. Implementation Considerations

When implementing applications using ISO/IEC 8859-2, consider the following:

- Use appropriate character sets and encodings for different regions.
- Be aware of the limitations when encoding or decoding characters outside the standard's scope.
- Test thoroughly to ensure compatibility across different systems.

---

Generated by Doxygen
if (c < 0xa0)
*pwc = (ucs4_t) c;
else
*pwc = (ucs4_t) iso8859_2_2uni[c-0xa0];
return 1;
}
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
static const unsigned char iso8859_2_page00[224] = {
0xa0, 0x00, 0x00, 0x00, 0x0a, 0x00, 0x00, 0x0a, / * 0xa0-0xa7 */
0xb0, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0xbf, / * 0xb0-0xbf */
0x00, 0xc1, 0x00, 0x00, 0xc8, 0x00, 0x00, 0xc7, / * 0xc0-0xc7 */
0x80, 0xc9, 0x00, 0xcb, 0xc7, 0xe0, 0xe0, 0xe0, / * 0xc8-0xcf */
0x00, 0xd0, 0x00, 0xe3, 0xf4, 0x00, 0xf6, 0xf7, / * 0xd0-0xdf */
0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x00-0x07 */
0x00, 0xc9, 0x00, 0xc9, 0x00, 0x00, 0x00, 0x00, / * 0x08-0x0f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x10-0x17 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x18-0x1f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x20-0x27 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x28-0x2f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x30-0x37 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x38-0x3f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x40-0x47 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x48-0x4f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x50-0x57 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x58-0x5f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x60-0x67 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x68-0x6f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x70-0x77 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x78-0xdf */

};

static const unsigned char iso8859_2_page02[32] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0b, / * 0xc0-0xc7 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x00-0x07 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x08-0x0f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x10-0x17 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x18-0x1f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x20-0x27 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x28-0x2f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x30-0x37 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x38-0x3f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x40-0x47 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x48-0x4f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x50-0x57 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x58-0x5f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x60-0x67 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x68-0x6f */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x70-0x77 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x78-0x7f */
};
#endif /* NEED_TOMB */

static int
iso8859_2_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{ unsigned char c = 0;
if (wc < 0x00a0) {
*r = wc;
return 1;
}
else if (wc >= 0x00a0 && wc < 0x0180)
c = iso8859_2_page00[wc-0x00a0];
else if (wc >= 0x02c0 && wc < 0x02e0)
c = iso8859_2_page02[wc-0x02c0];
if (c != 0) {
*r = c;
return 1;
}
return RET_ILSEQ;
}

 Generated by Doxygen
/* 0xc0 */
0x00c0, 0x00c1, 0x00c2, 0xfffd, 0x00c4, 0x010a, 0x0108, 0x00c7,
0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
/* 0xd0 */
0xfffd, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x0120, 0x00d6, 0x00d7,
0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x16c, 0x015c, 0x00df,
/* 0xe0 */
0x00e0, 0x00e1, 0x00e2, 0xfffd, 0x00e4, 0x010b, 0x0109, 0x00e7,
0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
/* 0xf0 */
0xfffd, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x0121, 0x00f6, 0x00f7,
0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x016c, 0x015c, 0x00df,
};

static int
iso8859_3_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0) {
        *pwc = (ucs4_t) c;
        return 1;
    }
    else {
        unsigned short wc = iso8859_3_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}
#endif /* NEED_TOWC */

#ifdef NEED_TOMB
static const unsigned char iso8859_3_page00[96] = {
    0xa0, 0x00, 0x00, 0xa3, 0xa4, 0x00, 0x00, 0xa7, /* 0xa0-0xa7 */
    0xa8, 0x00, 0x00, 0x00, 0x00, 0xad, 0x00, 0x00, /* 0xa8-0xaf */
    0xb0, 0x00, 0xb2, 0xb3, 0xb4, 0xb5, 0x00, 0xb7, /* 0xb0-0xb7 */
    0xb8, 0x00, 0x00, 0x00, 0x00, 0xbd, 0x00, 0x00, /* 0xb8-0xbf */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0x00, 0xc7, /* 0xc0-0xc7 */
    0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xd0, 0x00, 0xcf, /* 0xc8-0xcf */
    0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0x00, 0xc6, 0xc7, /* 0xc0-0xc7 */
    0xd0, 0xd1, 0xdb, 0xdc, 0x00, 0x00, 0xdf, /* 0xd0-0xdf */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0x00, 0xe7, /* 0xe0-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
    0x00, 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0x00, 0xf6, /* 0xf0-0xf7 */
    0x00, 0xf9, 0xfa, 0xfb, 0xfc, 0x00, 0x00, /* 0xf8-0xff */
};
static const unsigned char iso8859_3_page01[120] = {
    0xc6, 0xe6, 0xc5, 0xe5, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x10-0x17 */
    0x00, 0x00, 0x00, 0x00, 0xd8, 0xf8, 0x00, 0x00, /* 0x18-0x1f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
};
static const unsigned char iso8859_3_page02[8] = {
    0xa2, 0xff, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
};
#endif /* NEED_TOMB */

static int
iso8859_3_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    if (conv) (void)conv;
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0x0100)
        c = iso8859_3_page00[wc-0x00a0];
    else if (wc >= 0x0108 && wc < 0x0180)
        c = iso8859_3_page01[wc-0x0108];
    else if (wc >= 0x02d8 && wc < 0x02e0)
        c = iso8859_3_page02[wc-0x02d8];
    else {
        c = 0;
        *r = c;
        return 1;
    }
34.272 iso8859_4.h

```c
2
3 /* ISO-8859-4 */
4
5 #ifdef NEED_TOWC
6 static const unsigned short iso8859_4_2uni[96] =
7 / * 0xa0 */
8 0x00a0, 0x0104, 0x0138, 0x0156, 0x00a4, 0x0128, 0x013b, 0x00a7,
9 0x00a8, 0x0160, 0x0112, 0x0122, 0x0166, 0x00ad, 0x017d, 0x00af,
10 / * 0xb0 */
11 0x00b0, 0x0105, 0x02db, 0x0157, 0x00b4, 0x0129, 0x013c, 0x02c7,
12 0x00b8, 0x0161, 0x0113, 0x0123, 0x0167, 0x014a, 0x017e, 0x014b,
13 / * 0xc0 */
14 0x0100, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
15 0x00c8, 0x0162, 0x0114, 0x0124, 0x0168, 0x014c, 0x017f, 0x014d,
16 / * 0xd0 */
17 0x0110, 0x0146, 0x014d, 0x0136, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
18 0x00d8, 0x0172, 0x00da, 0x00db, 0x00dc, 0x0169, 0x016b, 0x02d9,
19 / * 0xe0 */
20 0x0101, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
21 0x00e8, 0x0173, 0x00fa, 0x00fb, 0x00fc, 0x016f, 0x016b, 0x02d9,
22 */
23 static int
24 iso8859_4_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
25 {
26 unsigned char c = *s;
27 if (c < 0xa0)
28 *pwc = (ucs4_t) c;
29 else
30 *pwc = (ucs4_t) iso8859_4_2uni[c-0xa0];
31 return 1;
32 }
33 #endif /* NEED_TOWC */
34
35 #ifdef NEED_TOMB
36 static const unsigned char iso8859_4_page00[224] =
37 / * 0xc0-0xc7 */
38 0xa0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
39 / * 0xc8-0xcf */
40 0x00, 0xc9, 0x00, 0xcb, 0x00, 0xcd, 0xce, 0x00,
41 / * 0xd0-0xdf */
42 0x00, 0xda, 0xdb, 0x00, 0xdc, 0x00, 0xdd, 0xde, 0x00,
43 / * 0xe0-0xef */
44 0x00, 0xe9, 0x00, 0xe6, 0x00, 0xe7, 0x00, 0xe8,
45 / * 0xe8-0xff */
46 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
47 / * 0x0100 */
48 0x00, 0x00, 0x00, 0x00, 0x0a1, 0xb1, 0x00, 0x00,
49 / * 0x00-0x07 */
50 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
51 / * 0x08-0x0f */
52 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
53 / * 0x00-0x07 */
54 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
55 / * 0x08-0x0f */
56 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
57 / * 0x00-0x07 */
58 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
59 / * 0x08-0x0f */
60 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
61 / * 0x00-0x07 */
62 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
63 / * 0x08-0x0f */
64 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
65 / * 0x00-0x07 */
66 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
67 / * 0x08-0x0f */
68 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
69 / * 0x00-0x07 */
70 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
71 / * 0x08-0x0f */
72 */
73 static const unsigned char iso8859_4_page02[32] =
74 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
75 / * 0x00-0x07 */
76 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
77 / * 0x08-0x0f */
78 */
```

Generated by Doxygen

---

102 }
103 return RET_ILSEQ;
104 }
105 #endif /* NEED_TOMB */

```

---

Generated by Doxygen
static int iso8859_5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
(void)conv; (void)n;
unsigned char c = 0;
if (wc < 0x00a0) {
    *r = wc;
    return 1;
} else if (wc >= 0x00a0 && wc < 0x0180)
    c = iso8859_4_page00[wc-0x00a0];
else if (wc >= 0x02c0 && wc < 0x02e0)
    c = iso8859_4_page02[wc-0x02c0];
if (c != 0) {
    *r = c;
    return 1;
} else {
    return RET_ILSEQ;
}
}
#endif /* NEED_TOMB */

static const unsigned short iso8859_5_2uni[96] = {
    0xa0, 0x0401, 0x0402, 0x0403, 0x0404, 0x0405, 0x0406, 0x0407,
    0x0408, 0x0409, 0x040a, 0x00ad, 0x040e, 0x040f,
    0xb0, 0x0410, 0x0411, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416, 0x0417,
    0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e, 0x041f,
    0xc0, 0x0420, 0x0421, 0x0422, 0x0423, 0x0424, 0x0425, 0x0426, 0x0427,
    0x0428, 0x0429, 0x042a, 0x042b, 0x042c, 0x042d, 0x042e, 0x042f,
    0xd0, 0x0430, 0x0431, 0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437,
    0x0438, 0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e, 0x043f,
    0xe0, 0x0440, 0x0441, 0x0442, 0x0443, 0x0444, 0x0445, 0x0446, 0x0447,
    0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d, 0x044e, 0x044f,
    0xf0, 0x0450, 0x0451, 0x0452, 0x0453, 0x0454, 0x0455, 0x0456, 0x0457,
    0x0458, 0x0459, 0x045a, 0x045b, 0x045c, 0x00a7, 0x045e, 0x045f,
};

static int iso8859_5_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) iso8859_5_2uni[c-0xa0];
    return 1;
}
#endif /* NEED_TOWC */

static const unsigned char iso8859_5_page00[16] = {
    0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfd, /* 0xa0-0xa7 */
    0x00, 0x00, 0xad, 0x00, 0x00, / * 0xe0-0x0af */
};

static int iso8859_5_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0)
        *pwc = (ucs4_t) c;
    else
        *pwc = (ucs4_t) iso8859_5_2uni[c-0xa0];
    return 1;
}
#endif /* NEED_TOMB */

static const unsigned char iso8859_5_page04[96] = {
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, /* 0x00-0x07 */
    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, /* 0x08-0x0f */
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, /* 0x10-0x17 */
    0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, /* 0x18-0x1f */
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, /* 0x20-0x27 */
    0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, /* 0x28-0x2f */
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, /* 0x30-0x37 */
    0x38, 0x39, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, /* 0x38-0x3f */
    0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xe0-0xe7 */
    0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xe8-0xef */
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xf0-0xf7 */
    0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff, /* 0xf8-0xff */
};

static int
62 iso8859_5_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
63 {
64  (void)conv; (void)n;
65  unsigned char c = 0;
66  if (wc < 0x00a0) {
67    *r = wc;
68    return 1;
69  }
70  else if (wc >= 0x00a0 && wc < 0x00b0)
71    c = iso8859_5_page00[wc-0x00a0];
72  else if (wc >= 0x0400 && wc < 0x0460)
73    c = iso8859_5_page04[wc-0x0400];
74  else if (wc == 0x2116)
75    c = 0xf0;
76  if (c != 0) {
77    *r = c;
78    return 1;
79  }
80  return RET_ILSEQ;
81 }
82 #endif /* NEED_TOMB */

34.274 iso8859_6.h

2 /
3 /*
4 * ISO-8859-6
5 */
6 #ifdef NEED_TOWC
7 static const unsigned short iso8859_6_2uni[96] = {
8  /* 0xa0 */
9    0x00a0, 0xfffd, 0xfffd, 0xfffd, 0x00a4, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
10    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
11    /* 0xb0 */
12    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
13    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
14    0xfffd, 0xfffd, 0xfffd, 0x061b, 0xfffd, 0xfffd, 0xfffd, 0x061f,
15    /* 0xc0 */
16    0xfffd, 0x0621, 0x0622, 0x0623, 0x0624, 0x0625, 0x0626, 0x0627,
17    0x0628, 0x0629, 0x062a, 0x062b, 0x062c, 0x062d, 0x062e, 0x062f,
18    /* 0xd0 */
19    0x0630, 0x0631, 0x0632, 0x0633, 0x0634, 0x0635, 0x0636, 0x0637,
20    0x0638, 0x0639, 0x063a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
21    /* 0xe0 */
22    0x0640, 0x0641, 0x0642, 0x0643, 0x0644, 0x0645, 0x0646, 0x0647,
23    0x0648, 0x0649, 0x064a, 0x064b, 0x064c, 0x064d, 0x064e, 0x064f,
24    /* 0xf0 */
25    0x0650, 0x0651, 0x0652, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
26    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
27};
28
29 static int
30 iso8859_6_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
31 {
32  unsigned char c = *s;
33  if (c < 0xa0) {
34    *pwc = (ucs4_t) c;
35    return 1;
36  }
37  else if (ucw == 0x00a0) { 
38    *pwc = (ucs4_t) c;
39    return 1;
40  }
41  return RET_ILSEQ;
42 }
43 #endif /* NEED_TOWC */

34.274 iso8859_6.h

2 /
3 /*
4 * ISO-8859-6
5 */
6 #ifdef NEED_TOWC
7 static const unsigned short iso8859_6_2uni[96] = {
8  /* 0xa0 */
9    0x00a0, 0xfffd, 0xfffd, 0xfffd, 0x00a4, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
10    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
11    /* 0xb0 */
12    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
13    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
14    0xfffd, 0xfffd, 0xfffd, 0x061b, 0xfffd, 0xfffd, 0xfffd, 0x061f,
15    /* 0xc0 */
16    0xfffd, 0x0621, 0x0622, 0x0623, 0x0624, 0x0625, 0x0626, 0x0627,
17    0x0628, 0x0629, 0x062a, 0x062b, 0x062c, 0x062d, 0x062e, 0x062f,
18    /* 0xd0 */
19    0x0630, 0x0631, 0x0632, 0x0633, 0x0634, 0x0635, 0x0636, 0x0637,
20    0x0638, 0x0639, 0x063a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
21    /* 0xe0 */
22    0x0640, 0x0641, 0x0642, 0x0643, 0x0644, 0x0645, 0x0646, 0x0647,
23    0x0648, 0x0649, 0x064a, 0x064b, 0x064c, 0x064d, 0x064e, 0x064f,
24    /* 0xf0 */
25    0x0650, 0x0651, 0x0652, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
26    0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
27};
28
29 static int
30 iso8859_6_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
31 {
32  unsigned char c = *s;
33  if (c < 0xa0) {
34    *pwc = (ucs4_t) c;
35    return 1;
36  }
37  else if (ucw == 0x00a0) { 
38    *pwc = (ucs4_t) c;
39    return 1;
40  }
41  return RET_ILSEQ;
42 }
43 #endif /* NEED_TOWC */
```c
static int
iso8859_6_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x00a0) {
        *r = wc;
        return 1;
    } else if (wc >= 0x00a0 && wc < 0x00b0)
        c = iso8859_6_page00[wc-0x00a0];
    else if (wc >= 0x0608 && wc < 0x0658)
        c = iso8859_6_page06[wc-0x0608];
    if (c != 0) {
        *r = c;
        return 1;
    }
}

static int
iso8859_7_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
    unsigned char c = *s;
    if (c < 0xa0) {
        *pwc = (ucs4_t) c;
        return 1;
    } else {
        unsigned short wc = iso8859_7_2uni[c-0xa0];
        if (wc != 0xfffd) {
            *pwc = (ucs4_t) wc;
            return 1;
        }
    }
    return RET_ILSEQ;
}
```

Generated by Doxygen
static const unsigned char iso8859_7_page20[16] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0xaf, 0x00, 0x00, /* 0x10-0x17 */
  0xa1, 0xa2, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
};

static int iso8859_7_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  (void)conv; (void)n;
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00c0)
    c = iso8859_7_page00[wc-0x00a0];
  else if (wc >= 0x0380 && wc < 0x03d0)
    c = iso8859_7_page03[wc-0x0380];
  else if (wc >= 0x2010 && wc < 0x2020)
    c = iso8859_7_page20[wc-0x2010];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

static const unsigned short iso8859_8_2uni[96] = {
  /* 0xa0 */
  0x00a0, 0xfffd, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
  /* 0xb0 */
  0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
  /* 0xc0 */
  0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
  /* 0xd0 */
  0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
  /* 0xe0 */
  0x05d0, 0x05d1, 0x05d2, 0x05d3, 0x05d4, 0x05d5, 0x05d6, 0x05d7,
  /* 0xf0 */
  0x05e0, 0x05e1, 0x05e2, 0x05e3, 0x05e4, 0x05e5, 0x05e6, 0x05e7,
  /* 0x10 */
  0x05e8, 0x05e9, 0x05ea, 0xfffd, 0xfffd, 0x200e, 0x200f, 0xfffd,
};

static int iso8859_8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = *s;
  if (c >= 0xa0) {
    unsigned short wc = iso8859_8_2uni[c-0xa0];
    if (wc != 0xfffd) {
      *pwc = (ucs4_t) wc;
      return 1;
    }
  } else {
    *pwc = (ucs4_t) c;
    return 1;
  }
  return RET_ILSEQ;
}
34.277 iso8859_9.h

```c
#define NEED_TOMB
static const unsigned char iso8859_8_page00[88] = {
  0xa0, 0x00, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, /* 0xa0-0xa7 */
  0xa8, 0xa9, 0x00, 0xab, 0xac, 0xad, 0xae, 0xaf, /* 0xa8-0xaf */
  0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, /* 0xb0-0xb7 */
  0xb8, 0xb9, 0x00, 0xbc, 0xbd, 0xbe, 0x00, /* 0xb8-0xbf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe8-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xf7 */
};
static const unsigned char iso8859_8_page05[32] = {
  0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, /* 0xd0-0xd7 */
  0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, /* 0xd8-0xdf */
  0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, /* 0xe0-0xe7 */
  0xf8, 0xf9, 0xfa, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe8-0xef */
};
static const unsigned char iso8859_8_page20[16] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfd, 0xfe, /* 0x08-0x0f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x17, /* 0x10-0x17 */
};

static int
iso8859_8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00f8)
    c = iso8859_8_page00[wc-0x00a0];
  else if (wc >= 0x05d0 && wc < 0x05f0)
    c = iso8859_8_page05[wc-0x05d0];
  else if (wc >= 0x2008 && wc < 0x2018)
    c = iso8859_8_page20[wc-0x2008];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
#endif /* NEED_TOMB */

#define NEED_TOWC
static const unsigned short iso8859_9_2uni[48] = {
  0x011e, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
  0x00d8, 0x00d9, 0x00ea, 0x00eb, 0x00ed, 0x00ee, 0x00ef,
  0x011f, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00ff, 0x00f6, 0x00f7,
  0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x0131, 0x0132, 0x0133,
};

static int
iso8859_9_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = *s;
  if (c >= 0xd0)
    *pwc = (ucs4_t) iso8859_9_2uni[c-0xd0];
  else
    *pwc = (ucs4_t) c;
  return 1;
}
#endif /* NEED_TOWC */

#define NEED_TOMB
static const unsigned char iso8859_9_page00[48] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
  0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, /* 0x08-0x0f */
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, /* 0x10-0x17 */
  0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, /* 0x18-0x1f */
};
static const unsigned char iso8859_9_page05[32] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
  0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, /* 0x08-0x0f */
  0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, /* 0x10-0x17 */
  0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, /* 0x18-0x1f */
};

static int
iso8859_9_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00f8)
    c = iso8859_9_2uni[wc-0x00a0];
  else if (wc >= 0x05d0 && wc < 0x05f0)
    c = iso8859_9_2uni[wc-0x05d0];
  else if (wc >= 0x2008 && wc < 0x2018)
    c = iso8859_9_2uni[wc-0x2008];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
#endif /* NEED_TOMB */
```

34.277 iso8859_9.h

```c

static int
iso8859_9_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00a0 && wc < 0x00f8)
    c = iso8859_9_2uni[wc-0x00a0];
  else if (wc >= 0x05d0 && wc < 0x05f0)
    c = iso8859_9_2uni[wc-0x05d0];
  else if (wc >= 0x2008 && wc < 0x2018)
    c = iso8859_9_2uni[wc-0x2008];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}
```

Generated by Doxygen
34.278 iso8859_9e.h

2 /
3 */
4 #endif /* NEED_TOMB */
5 */
6 static const unsigned short iso8859_9e_2uni[96] = {
7 /* 0xa0 */
8 0x00a0, 0x017d, 0x00a2, 0x00a3, 0x20ac, 0x00a5, 0x012c, 0x00a7,
9 0x016c, 0x00a9, 0x01e6, 0x00ab, 0x014a, 0x00ad, 0x00ae, 0x01d1,
10 /* 0xb0 */
11 0x00b0, 0x017e, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x012d, 0x00b7,
12 0x016d, 0x00b9, 0x01e7, 0x00bb, 0x014b, 0x00bd, 0x00be, 0x0178,
13 /* 0xc0 */
14 0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x012f, 0x00c7,
15 0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
16 /* 0xd0 */
17 0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00dd,
18 0x019f, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x0130, 0x0131, 0x00df,
19 /* 0xe0 */
20 0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x0259, 0x00e7,
21 0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
22 /* 0xf0 */
23 0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00fd,
24 0x0275, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x0131, 0x013f, 0x00ff,
25 /* */
26 };
static const unsigned char iso8859_9e_page01[136] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x1f, /* 0x18-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x27, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x28, 0x2f, /* 0x28-0x2f */
  0xdd, 0xfd, 0x30, 0x37, /* 0x30-0x37 */
  0x00, 0x00, 0x00, 0x00, 0x40, 0x4f, /* 0x40-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x50, 0x5f, /* 0x50-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x60, 0x6f, /* 0x60-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x70, 0x7f, /* 0x70-0x7f */
  0x00, 0x00, 0x00, 0x00, 0x80, 0x8f, /* 0x80-0x8f */
  0x00, 0x00, 0x00, 0x00, 0x90, 0x9f, /* 0x90-0x9f */
  0x00, 0x00, 0x00, 0x00, 0xa0, 0xa9, /* 0xa0-0xa9 */
  0x00, 0x00, 0x00, 0x00, 0xb0, 0xb9, /* 0xb0-0xb9 */
  0x00, 0x00, 0x00, 0x00, 0xc0, 0xcf, /* 0xc0-0xcf */
  0x00, 0x00, 0x00, 0x00, 0xd0, 0xdf, /* 0xd0-0xdf */
  0x00, 0x00, 0x00, 0x00, 0xe0, 0xef, /* 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0xf0, 0xff, /* 0xf0-0xff */
};

static const unsigned char iso8859_9e_page01_d[24] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x1f, /* 0x18-0x1f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x27, /* 0x20-0x27 */
  0x00, 0x00, 0x00, 0x00, 0x28, 0x2f, /* 0x28-0x2f */
  0x00, 0x00, 0x00, 0x00, 0x30, 0x37, /* 0x30-0x37 */
  0x00, 0x00, 0x00, 0x00, 0x40, 0x4f, /* 0x40-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x50, 0x5f, /* 0x50-0x5f */
  0x00, 0x00, 0x00, 0x00, 0x60, 0x6f, /* 0x60-0x6f */
  0x00, 0x00, 0x00, 0x00, 0x70, 0x7f, /* 0x70-0x7f */
  0x00, 0x00, 0x00, 0x00, 0x80, 0x8f, /* 0x80-0x8f */
  0x00, 0x00, 0x00, 0x00, 0x90, 0x9f, /* 0x90-0x9f */
  0x00, 0x00, 0x00, 0x00, 0xa0, 0xa9, /* 0xa0-0xa9 */
  0x00, 0x00, 0x00, 0x00, 0xb0, 0xb9, /* 0xb0-0xb9 */
  0x00, 0x00, 0x00, 0x00, 0xc0, 0xcf, /* 0xc0-0xcf */
  0x00, 0x00, 0x00, 0x00, 0xd0, 0xdf, /* 0xd0-0xdf */
  0x00, 0x00, 0x00, 0x00, 0xe0, 0xef, /* 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0xf0, 0xff, /* 0xf0-0xff */
};

static int iso8859_9e_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x0100) { c = iso8859_9e_page00[wc-0x00a0];
    *r = c;
    return 1;
  } else if (wc >= 0x0118 && wc < 0x01a0) { c = iso8859_9e_page01[wc-0x0118];
    *r = c;
    return 1;
  } else if (wc >= 0x01d0 && wc < 0x01e8) { c = iso8859_9e_page01_d[wc-0x01d0];
    *r = c;
    return 1;
  } else if (wc == 0x0259) { c = 0xe6;
    *r = c;
    return 1;
  } else if (wc == 0x20ac) { c = 0xa4;
    *r = c;
    return 1;
  } else if (wc == 0x00a0) { c = 0xe6;
    if (c != 0) {
      *r = c;
      return 1;
    }
  } return RET_ILSEQ;
}

static int jisx0201_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = 0;
  if (wc < 0x00a0) {
    *r = wc;
    return 1;
  } else if (wc >= 0x00a0 && wc < 0x0100) { c = iso8859_9e_page00[wc-0x00a0];
    *r = c;
    return 1;
  } else if (wc >= 0x0118 && wc < 0x01a0) { c = iso8859_9e_page01[wc-0x0118];
    *r = c;
    return 1;
  } else if (wc >= 0x01d0 && wc < 0x01e8) { c = iso8859_9e_page01_d[wc-0x01d0];
    *r = c;
    return 1;
  } else if (wc == 0x0259) { c = 0xe6;
    if (c == 0xe6) { c = 0x88; } else if (c == 0x0275) { c = 0x88; }
    *r = c;
    return 1;
  } return RET_ILSEQ;
}

Generated by Doxygen
static int jisx0201_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    (void)conv; (void)n;
    if (wc < 0x0080 && !(wc == 0x005c || wc == 0x007e)) {
        *r = wc;
        return 1;
    }
    if (wc == 0x00a5) {
        *r = 0x5c;
        return 1;
    }
    if (wc == 0x203e) {
        *r = 0x7e;
        return 1;
    }
    if (wc >= 0xff61 && wc < 0xffa0) {
        *r = wc - 0xfec0;
        return 1;
    }
    return RET_ILSEQ;
}
#endif /* NEED_TOMB */

static const unsigned short jisx0208_2uni_page21[690] = {
    0x3000, 0x3001, 0x3002, 0xff0c, 0xff0e, 0x30fb, 0xff1a, 0xff1b,
    0xff1f, 0xff01, 0x309b, 0x309c, 0x00b4, 0xff40, 0x00a8, 0xff3e,
    0xffe3, 0xff3f, 0x30fd, 0x30fe, 0x309d, 0x309e, 0x3003, 0x4edd,
    0x3005, 0x3006, 0x3007, 0x30fc, 0x2015, 0x2010, 0xff0f, 0xff3c,
    0x301c, 0x2016, 0xff5c, 0x2026, 0x2025, 0x2018, 0x2019, 0x201c,
    0x201d, 0xff08, 0xff09, 0x3014, 0x3015, 0xff3b, 0xff3d, 0xff5b,
    0xff5d, 0x3009, 0x300a, 0x300b, 0x300c, 0x300d, 0x300e, 0x3010,
    0x3011, 0x3012, 0x3013, 0x3016, 0x3017, 0x3018, 0x3019, 0x301a,
    0x301b, 0x301c, 0x301d, 0x301e, 0x3020, 0x3021, 0x3022, 0x3023,
    0x3024, 0x3025, 0x3026, 0x3027, 0x3028, 0x3029, 0x302a, 0x302b,
    0x302c, 0x302d, 0x302e, 0x302f, 0x3030, 0x3031, 0x3032, 0x3033,
    0x3034, 0x3035, 0x3036, 0x3037, 0x3038, 0x3039, 0x303a, 0x303b,
    0x303c, 0x303d, 0x303e, 0x303f, 0x3040, 0x3041, 0x3042, 0x3043,
    0x3044, 0x3045, 0x3046, 0x3047, 0x3048, 0x3049, 0x304a, 0x304b,
    0x304c, 0x304d, 0x304e, 0x304f, 0x3050, 0x3051, 0x3052, 0x3053,
    0x3054, 0x3055, 0x3056, 0x3057, 0x3058, 0x3059, 0x305a, 0x305b,
    0x305c, 0x305d, 0x305e, 0x305f, 0x3060,
};
static int
jisx0208_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
unsigned char c1 = (s[0] & 0x7F);
if ((c1 >= 0x21 && c1 <= 0x28) || (c1 >= 0x30 && c1 <= 0x74)) {
    if (n >= 2) {
        unsigned char c2 = (s[1] & 0x7F);
        if (c2 >= 0x21 && c2 < 0x7f) {
            unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
            unsigned short wc = 0xfffd;
            if (i < 1410) {
                if (i < 690)
                    wc = jisx0208_2uni_page21[i];
                else {
                    if (i < 7808)
                        wc = jisx0208_2uni_page22[i];
                    else
                        wc = jisx0208_2uni_page23[i];
                }
            } else
                wc = 0xfffe;
        } else
            wc = 0xfffe;
    } else
        wc = 0xfffe;
}
ifdef NEED_TOMB

static const unsigned short jisx20082charset"#79" = {

0x21A0, 0x21A1, 0x21A2, 0x21A3, 0x21A4, 0x21A5, 0x21A6, 0x21A7,
0x21A8, 0x21A9, 0x21AA, 0x21AB, 0x21AC, 0x21AD, 0x21AE, 0x21AF,
0x21B0, 0x21B1, 0x21B2, 0x21B3, 0x21B4, 0x21B5, 0x21B6, 0x21B7,
0x21B8, 0x21B9, 0x21BA, 0x21BB, 0x21BC, 0x21BD, 0x21BE, 0x21BF,
0x21C0, 0x21C1, 0x21C2, 0x21C3, 0x21C4, 0x21C5, 0x21C6, 0x21C7,
0x21C8, 0x21C9, 0x21CA, 0x21CB, 0x21CC, 0x21CD, 0x21CE, 0x21CF,
0x21D0, 0x21D1, 0x21D2, 0x21D3, 0x21D4, 0x21D5, 0x21D6, 0x21D7,
0x21D8, 0x21D9, 0x21DA, 0x21DB, 0x21DC, 0x21DD, 0x21DE, 0x21DF,
0x21E0, 0x21E1, 0x21E2, 0x21E3, 0x21E4, 0x21E5, 0x21E6, 0x21E7,
0x21E8, 0x21E9, 0x21EA, 0x21EB, 0x21EC, 0x21ED, 0x21EE, 0x21EF,
0x21F0, 0x21F1, 0x21F2, 0x21F3, 0x21F4, 0x21F5, 0x21F6, 0x21F7,
0x21F8, 0x21F9, 0x21FA, 0x21FB, 0x21FC, 0x21FD, 0x21FE, 0x21FF,
0x2200, 0x2201, 0x2202, 0x2203, 0x2204, 0x2205, 0x2206, 0x2207,
0x2208, 0x2209, 0x220A, 0x220B, 0x220C, 0x220D, 0x220E, 0x220F,
0x2210, 0x2211, 0x2212, 0x2213, 0x2214, 0x2215, 0x2216, 0x2217,
0x2218, 0x2219, 0x221A, 0x221B, 0x221C, 0x221D, 0x221E, 0x221F,
0x2220, 0x2221, 0x2222, 0x2223, 0x2224, 0x2225, 0x2226, 0x2227,
0x2228, 0x2229, 0x222A, 0x222B, 0x222C, 0x222D, 0x222E, 0x222F,
0x2230, 0x2231, 0x2232, 0x2233, 0x2234, 0x2235, 0x2236, 0x2237,
0x2238, 0x2239, 0x223A, 0x223B, 0x223C, 0x223D, 0x223E, 0x223F,
0x2240, 0x2241, 0x2242, 0x2243, 0x2244, 0x2245, 0x2246, 0x2247,
0x2248, 0x2249, 0x224A, 0x224B, 0x224C, 0x224D, 0x224E, 0x224F,
0x2250, 0x2251, 0x2252, 0x2253, 0x2254, 0x2255, 0x2256, 0x2257,
0x2258, 0x2259, 0x225A, 0x225B, 0x225C, 0x225D, 0x225E, 0x225F,
0x2260, 0x2261, 0x2262, 0x2263, 0x2264, 0x2265, 0x2266, 0x2267,
0x2268, 0x2269, 0x226A, 0x226B, 0x226C, 0x226D, 0x226E, 0x226F,
0x2270, 0x2271, 0x2272, 0x2273, 0x2274, 0x2275, 0x2276, 0x2277,
0x2278, 0x2279, 0x227A, 0x227B, 0x227C, 0x227D, 0x227E, 0x227F,
0x2280, 0x2281, 0x2282, 0x2283, 0x2284, 0x2285, 0x2286, 0x2287,
0x2288, 0x2289, 0x228A, 0x228B, 0x228C, 0x228D, 0x228E, 0x228F,
0x2290, 0x2291, 0x2292, 0x2293, 0x2294, 0x2295, 0x2296, 0x2297,
0x2298, 0x2299, 0x229A, 0x229B, 0x229C, 0x229D, 0x229E, 0x229F,
0x22A0, 0x22A1, 0x22A2, 0x22A3, 0x22A4, 0x22A5, 0x22A6, 0x22A7,
0x22A8, 0x22A9, 0x22AA, 0x22AB, 0x22AC, 0x22AD, 0x22AE, 0x22AF,
0x22B0, 0x22B1, 0x22B2, 0x22B3, 0x22B4, 0x22B5, 0x22B6, 0x22B7,
0x22B8, 0x22B9, 0x22BA, 0x22BB, 0x22BC, 0x22BD, 0x22BE, 0x22BF,
0x22C0, 0x22C1, 0x22C2, 0x22C3, 0x22C4, 0x22C5, 0x22C6, 0x22C7,
0x22C8, 0x22C9, 0x22CA, 0x22CB, 0x22CC, 0x22CD, 0x22CE, 0x22CF,
0x22D0, 0x22D1, 0x22D2, 0x22D3, 0x22D4, 0x22D5, 0x22D6, 0x22D7,
0x22D8, 0x22D9, 0x22DA, 0x22DB, 0x22DC, 0x22DD, 0x22DE, 0x22DF,
0x22E0, 0x22E1, 0x22E2, 0x22E3, 0x22E4, 0x22E5, 0x22E6, 0x22E7,
0x22E8, 0x22E9, 0x22EA, 0x22EB, 0x22EC, 0x22ED, 0x22EE, 0x22EF,
0x22F0, 0x22F1, 0x22F2, 0x22F3, 0x22F4, 0x22F5, 0x22F6, 0x22F7,
0x22F8, 0x22F9, 0x22FA, 0x22FB, 0x22FC, 0x22FD, 0x22FE, 0x22FF,
static const Summary16 jisx0208_uni2indx_pageff[15] = {
   { (unsigned short) 1 » i) {
    / Keep in 'used' only the bits 0..i-1. */
    used &= ((unsigned short) 1 « i) - 1;
    /* Add 'summary->indx' and the number of bits set in 'used'. */
    used = (used & 0x5555) + ((used & 0xaaaa) » 1);
    used = (used & 0x3333) + ((used & 0xcccc) » 2);
    used = (used & 0x0f0f) + (used » 8);
    c = jisx0208_2charset[summary->indx + used];
    r[0] = (c » 8); r[1] = (c & 0xff);
    return 2;
    } else if (wc == 0x0f00 && wc < 0xfff0) {
    summary = &jisx0208_uni2indx_pageff[wc»4]-0xff0];
    if (summary) {
        unsigned short used = summary->used;
        unsigned int i = wc & 0x0f;
        if (used & ((unsigned short) 1 « i)) {
            /* Keep in 'used' only the bits 0..i-1. */
            used &= ((unsigned short) 1 « i) - 1;
            /* Add 'summary->indx' and the number of bits set in 'used'. */
            used = (used & 0x5555) + ((used & 0xaaaa) » 1);
            used = (used & 0x3333) + ((used & 0xcccc) » 2);
            used = (used & 0x0f0f) + (used » 8);
            c = jisx0208_2charset[summary->indx + used];
            r[0] = (c » 8); r[1] = (c & 0xff);
            return 2;
        } else {
            return RET_ILSEQ;
        }
    } else {
        return RET_TOOSMALL;
    }
}
#endif /* NEED_TOMB */

```c
static const unsigned short jisx0212_2uni_page22[81] = {
  0x22 /* 0x22 */
};
static const unsigned short jisx0212_2uni_page26[188] = {
  0x26 /* 0x26 */
};
static const unsigned short jisx0212_2uni_page29[275] = {
  0x29 /* 0x29 */
};
static const unsigned short jisx0212_2uni_page22[81] = {
  0x22 /* 0x22 */
};
static const unsigned short jisx0212_2uni_page26[188] = {
  0x26 /* 0x26 */
};
static const unsigned short jisx0212_2uni_page29[275] = {
  0x29 /* 0x29 */
};
```
static int jisx0212_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c1 = (s[0] & 0x7F);
  if ((c1 == 0x22) || (c1 >= 0x26 && c1 <= 0x27) || (c1 >= 0x30 && c1 <= 0x6d)) {
    if (n >= 2) {
      unsigned char c2 = (s[1] & 0x7F);
      if (c2 >= 0x21 && c2 < 0x7f) {
        unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
        unsigned short wc = 0xfffd;
        if (i < 470) {
          if (i < 175)
            wc = jisx0212_2uni_page22[i-94];
          else if (i < 752)
            wc = jisx0212_2uni_page26[i-470];
          else if (i < 1410)
            wc = jisx0212_2uni_page29[i-752];
          else {
            if (i < 7211)
              wc = jisx0212_2uni_page30[i-1410];
          }
        }
      }
      if (wc != 0xfffd) {
        *pwc = (ucs4_t) wc;
        return 2;
      }
    }
  }
  return RET_ILSEQ;
}

#endif /* NEED_TOWC */

#ifdef NEED_TOMB

static const unsigned short jisx0212_2charset[6067] = {
  0x2237, 0x2242, 0x2270, 0x2243, 0x2246, 0x224e, 0x224e, 0x2234,
  0x2231, 0x226b, 0x2244, 0x22a2, 0x22a1, 0x22a4, 0x22a2, 0x22a3,
  0x2a29, 0x2a29, 0x2a26, 0x2a32, 0x2a31, 0x2a34, 0x2a33, 0x2a40,
  0x2a3f, 0x2a4f, 0x2a41, 0x2a50, 0x2a51, 0x2a54, 0x2a4a, 0x2a58,
  0x2a53, 0x2a9c, 0x2a63, 0x2a65, 0x2a66, 0x2a72, 0x2a90,
  0x2a9e, 0x2b22, 0x2b21, 0x2b24, 0x2b2a, 0x2b23, 0x2b29, 0x2b41,
  0x2b2e, 0x2b32, 0x2b31, 0x2b34, 0x2b33, 0x2b3f, 0x2b40, 0x2b42,
  0x2b43, 0x2b50, 0x2b55, 0x2b54, 0x2b58, 0x2b53,
  0x2b4c, 0x2b63, 0x2b65, 0x2b64, 0x2b72, 0x2b50, 0x2b73,
  0x2b27, 0x2b27, 0x2b25, 0x2b28, 0x2b28, 0x2b2b, 0x2b2b,
  0x2b2a, 0x2b2f, 0x2b2e, 0x2b2b, 0x2b2b, 0x2b2a, 0x2b2b,
  0x2b29, 0x2b29, 0x2b37, 0x2b36, 0x2b38, 0x2b38, 0x2b38,
  0x2b35, 0x2b35, 0x2b3a, 0x2b3a, 0x2b3b, 0x2b3a, 0x2b3d,
  0x2b3c, 0x2b3e, 0x2b3e, 0x2b29, 0x2b44, 0x2b47, 0x2b47, 0x2b45,
  0x2b4b, 0x2b4d, 0x2b4e, 0x2b4d, 0x2b4e, 0x2b4d, 0x2b4c,
  0x2b4b, 0x2b4b, 0x2b29, 0x2b49, 0x2b49, 0x2b48, 0x2b4d,
};

#endif /* NEED_TOMB */

#define NEED_TOWC

Generated by Doxygen
static const Summary16 jisx0212_uni2indx_page00[70] = {

 0x6d51, 0x6d52, 0x6d53, 0x6d54, 0x6d55, 0x6d56, 0x6d57, 0x6d58,
 0x6d41, 0x6d42, 0x6d43, 0x6d44, 0x6d45, 0x6d46, 0x6d47, 0x6d48,
 0x6d31, 0x6d32, 0x6d33, 0x6d34, 0x6d35, 0x6d36, 0x6d37, 0x6d38,
 0x6d21, 0x6d22, 0x6d23, 0x6d24, 0x6d25, 0x6d26, 0x6d27, 0x6d28,
 0x6c6f, 0x6c70, 0x6c71, 0x6c72, 0x6c73, 0x6c74, 0x6c75, 0x6c76,
 0x6c67, 0x6c68, 0x6c69, 0x6c6a, 0x6c6b, 0x6c6c, 0x6c6d, 0x6c6e,
 0x6c5f, 0x6c60, 0x6c61, 0x6c62, 0x6c63, 0x6c64, 0x6c65, 0x6c66,
 0x6c57, 0x6c58, 0x6c59, 0x6c5a, 0x6c5b, 0x6c5c, 0x6c5d, 0x6c5e,
 0x6c5f, 0x6c60, 0x6c61, 0x6c62, 0x6c63, 0x6c64, 0x6c65, 0x6c66,
 0x6c57, 0x6c58, 0x6c59, 0x6c5a, 0x6c5b, 0x6c5c, 0x6c5d, 0x6c5e,
 0x6c6f, 0x6c70, 0x6c71, 0x6c72, 0x6c73, 0x6c74, 0x6c75, 0x6c76,
 0x6c67, 0x6c68, 0x6c69, 0x6c6a, 0x6c6b, 0x6c6c, 0x6c6d, 0x6c6e,
 0x6c5f, 0x6c60, 0x6c61, 0x6c62, 0x6c63, 0x6c64, 0x6c65, 0x6c66,
 0x6c57, 0x6c58, 0x6c59, 0x6c5a, 0x6c5b, 0x6c5c, 0x6c5d, 0x6c5e,
 0x6c6f, 0x6c70, 0x6c71, 0x6c72, 0x6c73, 0x6c74, 0x6c75, 0x6c76,
 0x6c67, 0x6c68, 0x6c69, 0x6c6a, 0x6c6b, 0x6c6c, 0x6c6d, 0x6c6e,
 0x6c5f, 0x6c60, 0x6c61, 0x6c62, 0x6c63, 0x6c64, 0x6c65, 0x6c66,
 0x6c57, 0x6c58, 0x6c59, 0x6c5a, 0x6c5b, 0x6c5c, 0x6c5d, 0x6c5e,
 0x6c6f, 0x6c70, 0x6c71, 0x6c72, 0x6c73, 0x6c74, 0x6c75, 0x6c76,
 0x6c67, 0x6c68, 0x6c69, 0x6c6a, 0x6c6b, 0x6c6c, 0x6c6d, 0x6c6e,
 0x6c5f, 0x6c60, 0x6c61, 0x6c62, 0x6c63, 0x6c64, 0x6c65, 0x6c66,
static int
jisx0212_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
(void)conv;
if (n >= 2) {
const Summary16 *summary = NULL;
if (wc < 0x0460)
summary = &jisx0212_uni2indx_page00[(wc»4)];
else if (wc >= 0x2100 && wc < 0x2130)
summary = &jisx0212_uni2indx_page21[(wc»4)-0x210];
else if (wc >= 0x4e00 && wc < 0x9fb0)
summary = &jisx0212_uni2indx_page4e[(wc»4)-0x4e0];
if (summary) {
unsigned short used = summary->used;
unsigned int i = wc & 0x0f;
if (used & ((unsigned short) 1 « i)) {
unsigned short c;
/* Keep in 'used' only the bits 0..i-1. */
used &= ((unsigned short) 1 « i) - 1;
/* Add 'summary->indx' and the number of bits set in 'used'. */
used = (used & 0x5555) + ((used & 0xaaaa) » 1);
used = (used & 0x3333) + ((used & 0xcccc) » 2);
used = (used & 0x0f0f) + (used » 8);
c = jisx0212_2charset[summary->indx + used];
r[0] = (c » 8); r[1] = (c & 0xff);
return 2;
}
}
return RET_ILSEQ;
return RET_TOOSMALL;
}
34.282 koi8_c.h

2169 #endif /* NEED_TOMB */

34.282

koi8_c.h

2
3 /*
4 * KOI8-C
5 */
6
7 static const unsigned short koi8_c_2uni[128] = {
8
/* 0x80 */
9
0x0493, 0x0497, 0x049b, 0x049d, 0x04a3, 0x04af, 0x04b1, 0x04b3,
10
0x04b7, 0x04b9, 0x04bb, 0x2580, 0x04d9, 0x04e3, 0x04e9, 0x04ef,
11
/* 0x90 */
12
0x0492, 0x0496, 0x049a, 0x049c, 0x04a2, 0x04ae, 0x04b0, 0x04b2,
13
0x04b6, 0x04b8, 0x04ba, 0x2321, 0x04d8, 0x04e2, 0x04e8, 0x04ee,
14
/* 0xa0 */
15
0x00a0, 0x0452, 0x0453, 0x0451, 0x0454, 0x0455, 0x0456, 0x0457,
16
0x0458, 0x0459, 0x045a, 0x045b, 0x045c, 0x0491, 0x045e, 0x045f,
17
/* 0xb0 */
18
0x2116, 0x0402, 0x0403, 0x0401, 0x0404, 0x0405, 0x0406, 0x0407,
19
0x0486, 0x0409, 0x040a, 0x040b, 0x040c, 0x0490, 0x040e, 0x040f,
20
/* 0xc0 */
21
0x044e, 0x0430, 0x0431, 0x0446, 0x0434, 0x0435, 0x0444, 0x0433,
22
0x0445, 0x0438, 0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e,
23
/* 0xd0 */
24
0x043f, 0x044f, 0x0440, 0x0441, 0x0442, 0x0443, 0x0436, 0x0432,
25
0x044c, 0x044b, 0x0437, 0x0448, 0x044d, 0x0449, 0x0447, 0x044a,
26
/* 0xe0 */
27
0x042e, 0x0410, 0x0411, 0x0426, 0x0414, 0x0415, 0x0424, 0x0413,
28
0x0425, 0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e,
29
/* 0xf0 */
30
0x041f, 0x042f, 0x0420, 0x0421, 0x0422, 0x0423, 0x0416, 0x0412,
31
0x042c, 0x042b, 0x0417, 0x0428, 0x042d, 0x0429, 0x0427, 0x042a,
32 };
33
34 static int
35 koi8_c_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
36 {
37
unsigned char c = *s;
38
if (c < 0x80)
39
*pwc = (ucs4_t) c;
40
else
41
*pwc = (ucs4_t) koi8_c_2uni[c-0x80];
42
return 1;
43 }
44
45 static const unsigned char koi8_c_page00[1] = {
46
0xa0,
/* 0xa0-0xa7 */
47 };
48 static const unsigned char koi8_c_page04[240] = {
49
0x00, 0xb3, 0xb1, 0xb2, 0xb4, 0xb5, 0xb6, 0xb7, /* 0x00-0x07 */
50
0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0x00, 0xbe, 0xbf, /* 0x08-0x0f */
51
0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa, /* 0x10-0x17 */
52
0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0, /* 0x18-0x1f */
53
0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe8, 0xe3, 0xfe, /* 0x20-0x27 */
54
0xfb, 0xfd, 0xff, 0xf9, 0xf8, 0xfc, 0xe0, 0xf1, /* 0x28-0x2f */
55
0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xd6, 0xda, /* 0x30-0x37 */
56
0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, /* 0x38-0x3f */
57
0xd2, 0xd3, 0xd4, 0xd5, 0xc6, 0xc8, 0xc3, 0xde, /* 0x40-0x47 */
58
0xdb, 0xdd, 0xdf, 0xd9, 0xd8, 0xdc, 0xc0, 0xd1, /* 0x48-0x4f */
59
0x00, 0xa3, 0xa1, 0xa2, 0xa4, 0xa5, 0xa6, 0xa7, /* 0x50-0x57 */
60
0xa8, 0xa9, 0xaa, 0xab, 0xac, 0x00, 0xae, 0xaf, /* 0x58-0x5f */
61
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
62
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
63
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
64
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
65
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
66
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
67
0xbd, 0xad, 0x90, 0x80, 0x00, 0x00, 0x91, 0x81, /* 0x90-0x97 */
68
0x00, 0x00, 0x92, 0x82, 0x93, 0x83, 0x00, 0x00, /* 0x98-0x9f */
69
0x00, 0x00, 0x94, 0x84, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
70
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x95, 0x85, /* 0xa8-0xaf */
71
0x96, 0x86, 0x97, 0x87, 0x00, 0x00, 0x98, 0x88, /* 0xb0-0xb7 */
72
0x99, 0x89, 0x9a, 0x8a, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
73
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
74
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
75
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
76
0x9c, 0x8c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
77
0x00, 0x00, 0x9d, 0x8d, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xe7 */
78
0x9e, 0x8e, 0x00, 0x00, 0x00, 0x00, 0x9f, 0x8f, /* 0xe8-0xef */
79 };
80 static const unsigned char koi8_c_page22[1] = {
81
0xb0,
/* 0x16-0x16 */

Generated by Doxygen

1769


82 }
83 84 static int
85 koi8_c_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
86 {
87     unsigned char c = 0;
88     if (wc < 0x0080) {
89         *r = wc;
90         return 1;
91     }
92     else if (wc >= 0x00a0 && wc < 0x00a1)
93         c = koi8_c_page00[wc-0x00a0];
94     else if (wc >= 0x0400 && wc < 0x04ef)
95         c = koi8_c_page04[wc-0x0400];
96     else if (wc >= 0x2216 && wc < 0x2217)
97         c = koi8_c_page22[wc-0x2216];
98     if (c != 0) {
99         *r = c;
100        return 1;
101    }
102    return RET_ILSEQ;
103 }

34.283 koi8_r.h
1 /* $XFree86: xc/lib/X11/lcUniConv/koi8_r.h,v 1.3 2000/11/29 17:40:34 dawes Exp $ */
2
3 /*KOI8-R*/
4
5 /* Specification: RFC 1489*/
6
7 /*ifdef NEED_TOWC*/
10 static const unsigned short koi8_r_2uni[128] = {
11     /* 0x80 */
12     0x2500, 0x2502, 0x250c, 0x2510, 0x2514, 0x2518, 0x251c, 0x2524,
13     0x252c, 0x2534, 0x253c, 0x2580, 0x2584, 0x2588, 0x258c, 0x2590,
14     /* 0x90 */
15     0x2591, 0x2592, 0x2593, 0x2320, 0x25a0, 0x2219, 0x221a, 0x2248,
16     0x2264, 0x2265, 0x00a0, 0x2321, 0x00b0, 0x00b2, 0x00b7, 0x00f7,
17     /* 0xa0 */
18     0x2550, 0x2551, 0x2552, 0x2553, 0x2554, 0x2555, 0x2556,
19     0x2557, 0x2558, 0x2559, 0x255a, 0x255b, 0x255c, 0x255d, 0x255e,
20     /* 0xb0 */
21     0x255f, 0x2560, 0x2561, 0x2565, 0x2566, 0x2568, 0x2569,
22     0x256a, 0x256b, 0x256c, 0x256d, 0x256e, 0x256f,
23     /* 0xc0 */
24     0x0400, 0x0401, 0x0403, 0x0404, 0x0405, 0x0406, 0x0407,
25     0x0408, 0x0409, 0x040a, 0x040b, 0x040c, 0x040d, 0x040e,
26     /* 0xd0 */
27     0x0410, 0x0411, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416,
28     0x0417, 0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d,
29     /* 0xe0 */
30     0x041e, 0x041f, 0x0420, 0x0421, 0x0422, 0x0423, 0x0424,
31     0x0425, 0x0426, 0x0427, 0x0428, 0x0429, 0x042a,
32     /* 0xf0 */
33     0x042b, 0x042c, 0x042d, 0x042e, 0x042f, 0x0430, 0x0431,
34     0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437, 0x0438,
35     0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e,
36     /* 0x00 */
37     0x0440, 0x0441, 0x0442, 0x0443, 0x0444, 0x0445, 0x0446,
38     0x0447, 0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d,
39     /* 0x01 */
40     0x044e, 0x044f, 0x0450, 0x0451, 0x0452, 0x0453, 0x0454,
41     0x0455, 0x0456, 0x0457, 0x0458, 0x0459, 0x045a, 0x045b,
42     0x045c, 0x045d, 0x045e, 0x045f, 0x0460, 0x0461, 0x0462,
43     /* 0x02 */
44     0x0463, 0x0464, 0x0465, 0x0466, 0x0467, 0x0468, 0x0469,
45     0x046a, 0x046b, 0x046c, 0x046d, 0x046e, 0x046f, 0x0470,
46     /* 0x03 */
47     0x0471, 0x0472, 0x0473, 0x0474, 0x0475, 0x0476, 0x0477,
48     0x0478, 0x0479, 0x047a, 0x047b, 0x047c, 0x047d, 0x047e,
49     /* 0x04 */
50     0x0480, 0x0481, 0x0482, 0x0483, 0x0484, 0x0485, 0x0486,
51     0x0487, 0x0488, 0x0489, 0x048a, 0x048b, 0x048c, 0x048d,
52     /* 0x05 */
53     0x048e, 0x048f, 0x0490, 0x0491, 0x0492, 0x0493, 0x0494,
54     0x0495, 0x0496, 0x0497, 0x0498, 0x0499, 0x049a, 0x049b,
55     /* 0x06 */
56     0x049c, 0x049d, 0x049e, 0x049f, 0x04a0, 0x04a1, 0x04a2,
57     0x04a3, 0x04a4, 0x04a5, 0x04a6, 0x04a7, 0x04a8, 0x04a9,
58     /* 0x07 */
59     0x04aa, 0x04ab, 0x04ac, 0x04ad, 0x04ae, 0x04af, 0x04b0,
60     0x04b1, 0x04b2, 0x04b3, 0x04b4, 0x04b5, 0x04b6, 0x04b7,
61     /* 0x08 */
62     0x04b8, 0x04b9, 0x04ba, 0x04bb, 0x04bc, 0x04bd, 0x04be,
63     0x04bf, 0x04c0, 0x04c1, 0x04c2, 0x04c3, 0x04c4, 0x04c5,
64     /* 0x09 */
65     0x04c6, 0x04c7, 0x04c8, 0x04c9, 0x04ca, 0x04cb, 0x04cc,
66     0x04cd, 0x04ce, 0x04cf, 0x04d0, 0x04d1, 0x04d2, 0x04d3,
34.284 koi8_u.h

1771

61
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x9f, /* 0xf0-0xf7 */
62 };
63 static const unsigned char koi8_r_page04[88] = {
64
0x00, 0xb3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
65
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
66
0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa, /* 0x10-0x17 */
67
0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0, /* 0x18-0x1f */
68
0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe8, 0xe3, 0xfe, /* 0x20-0x27 */
69
0xfb, 0xfd, 0xff, 0xf9, 0xf8, 0xfc, 0xe0, 0xf1, /* 0x28-0x2f */
70
0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xd6, 0xda, /* 0x30-0x37 */
71
0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, /* 0x38-0x3f */
72
0xd2, 0xd3, 0xd4, 0xd5, 0xc6, 0xc8, 0xc3, 0xde, /* 0x40-0x47 */
73
0xdb, 0xdd, 0xdf, 0xd9, 0xd8, 0xdc, 0xc0, 0xd1, /* 0x48-0x4f */
74
0x00, 0xa3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
75 };
76 static const unsigned char koi8_r_page22[80] = {
77
0x00, 0x95, 0x96, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
78
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
79
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
80
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
81
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
82
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
83
0x97, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
84
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x50-0x57 */
85
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
86
0x00, 0x00, 0x00, 0x00, 0x98, 0x99, 0x00, 0x00, /* 0x60-0x67 */
87 };
88 static const unsigned char koi8_r_page23[8] = {
89
0x93, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
90 };
91 static const unsigned char koi8_r_page25[168] = {
92
0x80, 0x00, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
93
0x00, 0x00, 0x00, 0x00, 0x82, 0x00, 0x00, 0x00, /* 0x08-0x0f */
94
0x83, 0x00, 0x00, 0x00, 0x84, 0x00, 0x00, 0x00, /* 0x10-0x17 */
95
0x85, 0x00, 0x00, 0x00, 0x86, 0x00, 0x00, 0x00, /* 0x18-0x1f */
96
0x00, 0x00, 0x00, 0x00, 0x87, 0x00, 0x00, 0x00, /* 0x20-0x27 */
97
0x00, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, 0x00, /* 0x28-0x2f */
98
0x00, 0x00, 0x00, 0x00, 0x89, 0x00, 0x00, 0x00, /* 0x30-0x37 */
99
0x00, 0x00, 0x00, 0x00, 0x8a, 0x00, 0x00, 0x00, /* 0x38-0x3f */
100
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */
101
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
102
0xa0, 0xa1, 0xa2, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8, /* 0x50-0x57 */
103
0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, 0xb0, /* 0x58-0x5f */
104
0xb1, 0xb2, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, /* 0x60-0x67 */
105
0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0x00, 0x00, 0x00, /* 0x68-0x6f */
106
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
107
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
108
0x8b, 0x00, 0x00, 0x00, 0x8c, 0x00, 0x00, 0x00, /* 0x80-0x87 */
109
0x8d, 0x00, 0x00, 0x00, 0x8e, 0x00, 0x00, 0x00, /* 0x88-0x8f */
110
0x8f, 0x90, 0x91, 0x92, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
111
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x98-0x9f */
112
0x94, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
113 };
114
115 static int
116 koi8_r_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
117 {
118
(void)conv; (void)n;
119
unsigned char c = 0;
120
if (wc < 0x0080) {
121
*r = wc;
122
return 1;
123
}
124
else if (wc >= 0x00a0 && wc < 0x00f8)
125
c = koi8_r_page00[wc-0x00a0];
126
else if (wc >= 0x0400 && wc < 0x0458)
127
c = koi8_r_page04[wc-0x0400];
128
else if (wc >= 0x2218 && wc < 0x2268)
129
c = koi8_r_page22[wc-0x2218];
130
else if (wc >= 0x2320 && wc < 0x2328)
131
c = koi8_r_page23[wc-0x2320];
132
else if (wc >= 0x2500 && wc < 0x25a8)
133
c = koi8_r_page25[wc-0x2500];
134
if (c != 0) {
135
*r = c;
136
return 1;
137
}
138
return RET_ILSEQ;
139 }
140 #endif /* NEED_TOMB */

34.284

koi8_u.h

1 /* $XFree86:
2

xc/lib/X11/lcUniConv/koi8_u.h,v 1.3 2000/11/29 17:40:34 dawes Exp $ */

Generated by Doxygen


1772

File Documentation

3 /*
4 * KOI8-U
5 */
6
7 /* Specification: RFC 2319 */
8 #ifdef NEED_TOWC
9 static const unsigned short koi8_u_2uni[128] = {
10
/* 0x80 */
11
0x2500, 0x2502, 0x250c, 0x2510, 0x2514, 0x2518, 0x251c, 0x2524,
12
0x252c, 0x2534, 0x253c, 0x2580, 0x2584, 0x2588, 0x258c, 0x2590,
13
/* 0x90 */
14
0x2591, 0x2592, 0x2593, 0x2320, 0x25a0, 0x2219, 0x221a, 0x2248,
15
0x2264, 0x2265, 0x00a0, 0x2321, 0x00b0, 0x00b2, 0x00b7, 0x00f7,
16
/* 0xa0 */
17
0x2550, 0x2551, 0x2552, 0x0451, 0x0454, 0x2554, 0x0456, 0x0457,
18
0x2557, 0x2558, 0x2559, 0x255a, 0x255b, 0x0491, 0x255d, 0x255e,
19
/* 0xb0 */
20
0x255f, 0x2560, 0x2561, 0x0401, 0x0404, 0x2563, 0x0406, 0x0407,
21
0x2566, 0x2567, 0x2568, 0x2569, 0x256a, 0x0490, 0x256c, 0x00a9,
22
/* 0xc0 */
23
0x044e, 0x0430, 0x0431, 0x0446, 0x0434, 0x0435, 0x0444, 0x0433,
24
0x0445, 0x0438, 0x0439, 0x043a, 0x043b, 0x043c, 0x043d, 0x043e,
25
/* 0xd0 */
26
0x043f, 0x044f, 0x0440, 0x0441, 0x0442, 0x0443, 0x0436, 0x0432,
27
0x044c, 0x044b, 0x0437, 0x0448, 0x044d, 0x0449, 0x0447, 0x044a,
28
/* 0xe0 */
29
0x042e, 0x0410, 0x0411, 0x0426, 0x0414, 0x0415, 0x0424, 0x0413,
30
0x0425, 0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e,
31
/* 0xf0 */
32
0x041f, 0x042f, 0x0420, 0x0421, 0x0422, 0x0423, 0x0416, 0x0412,
33
0x042c, 0x042b, 0x0417, 0x0428, 0x042d, 0x0429, 0x0427, 0x042a,
34 };
35
36 static int
37 koi8_u_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
38 {
39
unsigned char c = *s;
40
if (c < 0x80)
41
*pwc = (ucs4_t) c;
42
else
43
*pwc = (ucs4_t) koi8_u_2uni[c-0x80];
44
return 1;
45 }
46 #endif /* NEED_TOWC */
47
48 #ifdef NEED_TOMB
49 static const unsigned char koi8_u_page00[88] = {
50
0x9a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
51
0x00, 0xbf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
52
0x9c, 0x00, 0x9d, 0x00, 0x00, 0x00, 0x00, 0x9e, /* 0xb0-0xb7 */
53
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
54
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
55
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
56
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xd7 */
57
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd8-0xdf */
58
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xe7 */
59
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe8-0xef */
60
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x9f, /* 0xf0-0xf7 */
61 };
62 static const unsigned char koi8_u_page04[152] = {
63
0x00, 0xb3, 0x00, 0x00, 0xb4, 0x00, 0xb6, 0xb7, /* 0x00-0x07 */
64
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x08-0x0f */
65
0xe1, 0xe2, 0xf7, 0xe7, 0xe4, 0xe5, 0xf6, 0xfa, /* 0x10-0x17 */
66
0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0, /* 0x18-0x1f */
67
0xf2, 0xf3, 0xf4, 0xf5, 0xe6, 0xe8, 0xe3, 0xfe, /* 0x20-0x27 */
68
0xfb, 0xfd, 0xff, 0xf9, 0xf8, 0xfc, 0xe0, 0xf1, /* 0x28-0x2f */
69
0xc1, 0xc2, 0xd7, 0xc7, 0xc4, 0xc5, 0xd6, 0xda, /* 0x30-0x37 */
70
0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, /* 0x38-0x3f */
71
0xd2, 0xd3, 0xd4, 0xd5, 0xc6, 0xc8, 0xc3, 0xde, /* 0x40-0x47 */
72
0xdb, 0xdd, 0xdf, 0xd9, 0xd8, 0xdc, 0xc0, 0xd1, /* 0x48-0x4f */
73
0x00, 0xa3, 0x00, 0x00, 0xa4, 0x00, 0xa6, 0xa7, /* 0x50-0x57 */
74
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x58-0x5f */
75
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x60-0x67 */
76
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
77
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
78
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
79
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
80
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x88-0x8f */
81
0xbd, 0xad, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
82 };
83 static const unsigned char koi8_u_page22[80] = {
84
0x00, 0x95, 0x96, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
85
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
86
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
87
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
88
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
89
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x40-0x47 */

Generated by Doxygen


static const unsigned char koi8_u_page23[8] = {
    0x93, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x48-0x4f */
}

static const unsigned char koi8_u_page25[168] = {
    0x80, 0x00, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x00-0x07 */
    0x00, 0x00, 0x00, 0x00, 0x82, 0x00, 0x00, 0x00, /* 0x08-0x0f */
    0x83, 0x00, 0x00, 0x00, 0x84, 0x00, 0x00, 0x00, /* 0x10-0x17 */
    0x85, 0x00, 0x00, 0x00, 0x86, 0x00, 0x00, 0x00, /* 0x18-0x1f */
    0x00, 0x00, 0x00, 0x00, 0x87, 0x00, 0x00, 0x00, /* 0x20-0x27 */
    0x00, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, 0x00, /* 0x28-0x2f */
    0x00, 0x00, 0x00, 0x00, 0x89, 0x00, 0x00, 0x00, /* 0x30-0x37 */
    0x00, 0x00, 0x00, 0x00, 0x8a, 0x00, 0x00, 0x00, /* 0x38-0x3f */
    0x00, 0x00, 0x00, 0x00, 0x8b, 0x00, 0x00, 0x00, /* 0x40-0x47 */
    0x00, 0x00, 0x00, 0x00, 0x8c, 0x00, 0x00, 0x00, /* 0x48-0x4f */
    0xa0, 0xa1, 0xa2, 0x00, 0xa5, 0x00, 0x00, 0xa8, /* 0x50-0x57 */
    0xa9, 0xaa, 0xab, 0xac, 0x00, 0xae, 0xaf, 0xb0, /* 0x58-0x5f */
    0xb1, 0xb2, 0x00, 0xb5, 0x00, 0x00, 0xb8, 0xb9, /* 0x60-0x67 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x68-0x6f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x78-0x7f */
    0x8b, 0x00, 0x00, 0x00, 0x8c, 0x00, 0x00, 0x00, /* 0x80-0x87 */
    0x8d, 0x00, 0x00, 0x00, 0x8e, 0x00, 0x00, 0x00, /* 0x88-0x8f */
    0x00, 0x00, 0x00, 0x00, 0x8f, 0x00, 0x00, 0x00, /* 0x90-0x97 */
    0x00, 0x00, 0x00, 0x00, 0x90, 0x00, 0x00, 0x00, /* 0x98-0x9f */
    0x94, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
    0x9b, 0x00, 0x00, 0x00, 0x9c, 0x00, 0x00, 0x00, /* 0xa8-0xaf */
    0x00, 0x00, 0x00, 0x00, 0x9d, 0x00, 0x00, 0x00, /* 0xb0-0xb7 */
    0x00, 0x00, 0x00, 0x00, 0x9e, 0x00, 0x00, 0x00, /* 0xb8-0xbf */
    0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc0-0xc7 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xc8-0xcf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xd0-0xdf */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xe0-0xef */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf0-0xff */
};

static int koi8_u_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) {
    (void)conv; (void)n;
    unsigned char c = 0;
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    }
    else if (wc >= 0x00a0 && wc < 0x00f8) {
        c = koi8_u_page00[wc-0x00a0];
    }
    else if (wc >= 0x0400 && wc < 0x0498) {
        c = koi8_u_page04[wc-0x0400];
    }
    else if (wc >= 0x2218 && wc < 0x2268) {
        c = koi8_u_page22[wc-0x2218];
    }
    else if (wc >= 0x2320 && wc < 0x2328) {
        c = koi8_u_page23[wc-0x2320];
    }
    else if (wc >= 0x2500 && wc < 0x25a8) {
        c = koi8_u_page25[wc-0x2500];
    }
    if (c != 0) {
        *r = c;
        return 1;
    }
    return RET_ILSEQ;
}

#endif /* NEED_TOMB */
static const unsigned short ksc5601_2uni_page4a[4888] = {
    0xd78c, 0xd790, 0xd798, 0xd799, 0xd79b, 0xd79d,
    0xd76c, 0xd770, 0xd774, 0xd77c, 0xd77d, 0xd781, 0xd788, 0xd789,
    0xd757, 0xd758, 0xd759, 0xd760, 0xd761, 0xd763, 0xd765, 0xd769,
    0xd73c, 0xd744, 0xd747, 0xd749, 0xd750, 0xd751, 0xd754, 0xd756,
    0xd6fc, 0xd6fd, 0xd700, 0xd704, 0xd711, 0xd718, 0xd719, 0xd71c,
    0xd5f5, 0xd5f7, 0xd5f9, 0xd5fa, 0xd5fb, 0xd5fc, 0xd5fd, 0xd5fe,
    0xd590, 0xd5a5, 0xd5b8, 0xd5bc, 0xd5c8, 0xd5d0, 0xd5d2, 0xd5d8,
    0xd575, 0xd578, 0xd57c, 0xd584, 0xd585, 0xd587, 0xd588, 0xd589,
    0xd590, 0xd593, 0xd595, 0xd596, 0xd598, 0xd59e, 0xd5a5, 0xd5a6,
    0xd5f5, 0xd5f7, 0xd5f9, 0xd600, 0xd601, 0xd604, 0xd608, 0xd610,
    0xd611, 0xd613, 0xd615, 0xd61c, 0xd620,
    0x5b63, 0x5c46, 0x60b8, 0x6212, 0x6229, 0x624a,
    0x757a, 0x7586, 0x758f, 0x7597, 0x7598, 0x759b, 0x759e, 0x759f,
    0x75a7, 0x75a9, 0x75aa, 0x75ac, 0x75b7, 0x75b9, 0x75ba, 0x75b7,
    0x75b9, 0x75ba, 0x75b7, 0x75b9, 0x75ba, 0x75b7, 0x75b9, 0x75ba,
#ifdef NEED_TOMB

```c

if (c2 >= 0x21 && c2 < 0x7F) {
  unsigned int i = 94 * (c1 - 0x21) + (c2 - 0x21);
  unsigned short wc = 0xfffd;
}
#endif /* NEED_TOMB */
```
static const Summary1 ksc5601_uni2indx_page00[100] = {
   0x0000, /*
   0, 0,0000, { 0, 0,0000 }, { 0, 0,0000 }, { 0, 0,0000 },
   0, 0,0000, { 0, 0,0000 }, { 0, 0,0000 },
   0, 0,0000, { 0, 0,0000 }, { 0, 0,0000 },
   0, 0,0000, { 0, 0,0000 }, { 0, 0,0000 },
   { 21, 0xc181 }, 26, 0x0040, 21, 0x4181, */
   0x0010 /*
   31, 0x0000, { 31, 0x0000 }, { 32, 0x0000 }, { 34, 0x0100 },
   39, 0x0000, { 40, 0x0000 }, { 47, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   0x0020 /*
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 49, 0x0000 }, { 49, 0x0000 },
   49, 0x0000, { 50, 0x2f01 }, { 56, 0x0000 },
   0x0030 /*
   56, 0x0000, { 56, 0x0000 }, { 56, 0x0000 },
   56, 0x0000, { 56, 0x0000 }, { 56, 0x0000 },
   56, 0x0000, { 56, 0x0000 }, { 56, 0x0000 },
   95, 0x03fb, { 104, 0x0000 }, { 104, 0x0000 },
   0x0040 /*
   104, 0x0000, { 105, 0xffff }, { 121, 0xffff },
   137, 0xffff, { 169, 0x0000 },
   0x0050, */
   static const Summary1 ksc5601_uni2indx_page20[103] = {
   0x2000 /*
   170, 0x0000, { 170, 0x3320 }, { 175, 0x0063 }, { 179, 0x080d },
   183, 0x0000, { 183, 0x0000 }, { 183, 0x0000 },
   185, 0x0000, { 189, 0x0000 }, { 189, 0x0000 },
   189, 0x0000, { 189, 0x0000 }, { 189, 0x0000 },
   0x2100 /*
   189, 0x2008, { 191, 0x0048 }, { 193, 0x0846 }, { 197, 0x7818 },
   197, 0x0000, { 203, 0x03ff }, { 213, 0x03ff },
   223, 0x0000, { 233, 0x03ff }, { 233, 0x0000 },
   233, 0x0000, { 233, 0x0000 }, { 235, 0x0000 },
   0x2200 /*
   235, 0x0000, { 246, 0x5f11 }, { 255, 0x0330 },
   259, 0x0004, { 260, 0x0c33 }, { 266, 0x0000 },
   270, 0x0020, { 271, 0x0020 }, { 272, 0x0000 },
   272, 0x0000, { 272, 0x0000 }, { 272, 0x0000 },
   0x2300 /*
   272, 0x0000, { 272, 0x0000 }, { 273, 0x0000 },
   273, 0x0000, { 273, 0x0000 }, { 273, 0x0000 },
   273, 0x0000, { 273, 0x0000 }, { 273, 0x0000 },
   0x2400 /*
   273, 0x0000, { 273, 0x0000 }, { 273, 0x0000 },
   273, 0x0000, { 273, 0x0000 }, { 288, 0x00ff },
   303, 0x0000, { 307, 0xffff }, { 323, 0xffff },
   329, 0x0000, { 329, 0xffff }, { 345, 0xffff },
   0x2500 */
"}
Generated by Doxygen
summary = ksc5601_uni2indx_pagef9[(wc>>4)+0xf90];
} else if (wc >= 0xff00 && wc < 0xfff0)
summary = ksc5601_uni2indx_pageff[(wc>>4)+0xff0];
if (summary) {
unsigned short used = summary->used;
unsigned int i = wc & 0x0f;
if (used & ((unsigned short) 1 « i)) {
unsigned short c;
/ * Keep in 'used' only the bits 0..i-1. */
used &= ((unsigned short) 1 « i) - 1;
/ * Add 'summary->indx' and the number of bits set in 'used'. */
used = (used & 0x5555) + ((used & 0xaaa) >> 1);
used = (used & 0xaaaa) + (used & 0x0ff0) >> 4;
used = (used & 0x0ff0) + (used & 0x00f0) >> 8;
c = ksc5601_2charset[summary->indx + used];
r[0] = (c >> 8); r[1] = (c & 0xff);
return 2;
}
return RET_ILSEQ;
return RET_TOOSMALL;
#endif /* NEED_TOMB */

mulelao.c

static const unsigned short mulelao_2uni[96] = {
/ * 0xa0 */
0x00a0, 0x0e81, 0x0e82, 0xfffd, 0x0e84, 0xfffd, 0x0e87,
0x0e88, 0xfffd, 0x0e8a, 0xfffd, 0x0e8d, 0xfffd, 0x0e8f,
/ * 0xb0 */
0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e94, 0x0e95, 0x0e96, 0x0e97,
0x0e98, 0x0e99, 0x0e9a, 0x0e9b, 0x0e9c, 0x0e9d, 0x0e9e, 0x0e9f,
/ * 0xc0 */
0xfffd, 0x0ea1, 0x0ea2, 0x0ea3, 0xfffd, 0x0ea5, 0xfffd, 0x0ea7,
0xfffd, 0x0ea9, 0x0eab, 0x0eac, 0xfffd, 0x0ead, 0x0eae, 0x0eaf,
/ * 0xd0 */
0x0eb0, 0x0eb1, 0x0eb2, 0x0eb3, 0x0eb4, 0x0eb5, 0x0eb6, 0x0eb7,
0x0eb8, 0x0eb9, 0x0ebc, 0x0ebd, 0x0ebe, 0xfffd, 0x0ebf,
/ * 0xe0 */
0x0ec0, 0x0ec1, 0x0ec2, 0x0ec3, 0x0ec4, 0xfffd, 0x0ec6, 0xfffd,
0x0ec8, 0x0ec9, 0x0eca, 0x0ecb, 0x0ecc, 0x0ecd, 0xfffd, 0xfffd,
/ * 0xf0 */
0x0ef0, 0x0ef1, 0x0ef2, 0x0ef3, 0xfffd, 0x0ef5, 0xfffd, 0x0ef7,
0x0ef8, 0x0ef9, 0x0efc, 0x0efd, 0x0efc, 0x0efb, 0x0efb,
};

static int
mulelao_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
unsigned char c = *s;
if (c < 0xa0) {
*pwc = (ucs4_t) c;
return 1;
}
else {
unsigned short wc = mulelao_2uni[c-0xa0];
if (wc != 0xfffd) {
*pwc = (ucs4_t) wc;
return 1;
}
return RET_ILSEQ;
}

mulelao.h

#define mulelao_2charset(summary->indx + used)
61 static int
62 mulelao_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
63 {
64 unsigned char c = 0;
65 if (wc < 0x00a0) {
66 *r = wc;
67 return 1;
68 }
69 else if (wc == 0x00a0)
70 c = 0xa0;
71 else if (wc >= 0x0e80 && wc < 0x0ee0)
72 c = mulelao_page0e[wc-0x0e80];
73 if (c != 0) {
74 *r = c;
75 return 1;
76 }
77 return RET_ILSEQ;
78 }

34.287 tatar_cyr.h

2
3 /*
4 * TATAR-CYR
5 */
6
7 static const unsigned short tatar_cyr_2uni[128] = {
8 / * 0x80 */
9 0x04d8, 0x0403, 0x201a, 0x0453, 0x201e, 0x2026, 0x2020, 0x2021,
10 0x040c, 0x0403, 0x2039, 0x044e, 0x044f, 0x0442, 0x044a,
11 / * 0x90 */
12 0x04d9, 0x2018, 0x2019, 0x201c, 0x201d, 0x2022, 0x2023, 0x2024,
13 0x98, 0x2122, 0x0496, 0x04a2, 0x04ba,
14 / * 0xa0 */
15 0x00a0, 0x040e, 0x045e, 0x0408, 0x00a4, 0x0490, 0x00a6, 0x00a7,
16 0x00a9, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
17 / * 0xb0 */
18 0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
19 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
20 / * 0xc0 */
21 0x04c0, 0x04c1, 0x04c2, 0x04c3, 0x04c4, 0x04c5, 0x04c6, 0x04c7,
22 0x04c8, 0x04c9, 0x04ca, 0x04cb, 0x04cc, 0x04cd, 0x04ce, 0x04cf,
23 / * 0xd0 */
24 0x04d0, 0x04d1, 0x04d2, 0x04d3, 0x04d4, 0x04d5, 0x04d6, 0x04d7,
25 0x04d8, 0x04d9, 0x04da, 0x04db, 0x04dc, 0x04dd, 0x04de, 0x04df,
26 / * 0xe0 */
27 0x04e0, 0x04e1, 0x04e2, 0x04e3, 0x04e4, 0x04e5, 0x04e6, 0x04e7,
28 0x04e8, 0x04e9, 0x04ea, 0x04eb, 0x04ec, 0x04ed, 0x04ee, 0x04ef,
29 / * 0xf0 */
30 0x04f0, 0x04f1, 0x04f2, 0x04f3, 0x04f4, 0x04f5, 0x04f6, 0x04f7,
31 0x04f8, 0x04f9, 0x04fa, 0x04fb, 0x04fc, 0x04fd, 0x04fe, 0x04ff,
32 }
33
34 static int
35 tatar_cyr_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
36 {
37 unsigned char c = 0;
38 if (c < 0x80)
39 *pwc = (ucs4_t) c;
40 return 1;
41 }
42
43 static const unsigned char tatar_cyr_page00[32] = {
44 0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
45 / * 0x00a8 */
46 0x00a9, 0x00a0, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
47 / * 0x00b0 */
48 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
49 / * 0x00c0 */
50 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
51 / * 0x00d0 */
52 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
53 / * 0x00e0 */
54 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
55 / * 0x00f0 */
56 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
57 / * 0x0100 */
58 Generated by Doxygen
```c
static const unsigned char tatar_cyr_page20[48] = {
    0x00, 0x00, 0x00, 0x96, 0x97, 0x00, 0x00, 0x00, /* 0x10-0x17 */
    0x91, 0x92, 0x82, 0x00, 0x93, 0x94, 0x84, 0x00, /* 0x18-0x1f */
    0x86, 0x87, 0x95, 0x00, 0x00, 0x00, 0x85, 0x00, /* 0x20-0x27 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
    0x89, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
    0x00, 0x8b, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x38-0x3f */
};

static const unsigned char tatar_cyr_page21[24] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xb9, 0x00, /* 0x10-0x17 */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x18-0x1f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x20-0x27 */
    0x96, 0x97, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x28-0x2f */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x30-0x37 */
};

static const unsigned char tatar_cyr_page22[1] = {
    0xb0, /* 0x16-0x16 */
};

static int
tatar_cyr_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
    unsigned char c = 0;
    if (wc < 0x0080) {
        *r = wc;
        return 1;
    } else if (wc >= 0x00a0 && wc < 0x00bc)
        c = tatar_cyr_page00[wc-0x00a0];
    else if (wc >= 0x0400 && wc < 0x04ef)
        c = tatar_cyr_page04[wc-0x0400];
    else if (wc >= 0x2010 && wc < 0x203b)
        c = tatar_cyr_page20[wc-0x2010];
    else if (wc == 0x20ac)
        c = 0x88;
    else if (wc >= 0x2110 && wc < 0x2123)
        c = tatar_cyr_page21[wc-0x2110];
    if (c != 0) {
        *r = c;
        return 1;
    } return RET_ILSEQ;
}
```

34.288 tcvn.h

2
4 /* 
5 * TCVN-5712
6 */

```c
static const unsigned short tcvn_2uni_1[32] = {
    0x0000, 0x00da, 0x1ee4, 0x0003, 0x1eea, 0x1eec, 0x1eee, 0x0007,
    0x0008, 0x00c3, 0x00da, 0x00c1, 0x00ea, 0x00ec, 0x000f,
    0x0000, 0x00da, 0x1ee4, 0x0003, 0x1eea, 0x1eec, 0x1eee, 0x0007,
    0x00c3, 0x00ca, 0x00c1, 0x00ec, 0x000f, 0x0000, 0x00da, 0x00c1,
    0x00ea, 0x00ec, 0x000f, /* 0x10-0x17 */
    0x00c3, 0x00da, 0x00c1, 0x00ec, 0x000f, /* 0x18-0x1f */
    0x00c3, 0x00da, 0x00c1, 0x00ec, 0x000f, /* 0x20-0x27 */
    0x00c3, 0x00da, 0x00c1, 0x00ec, 0x000f, /* 0x28-0x2f */
    0x00c3, 0x00da, 0x00c1, 0x00ec, 0x000f, /* 0x30-0x37 */
};

static const unsigned short tcvn_2uni_2[128] = {
    0x00c3, 0x00da, 0x00c1, 0x00ec, 0x000f, /* 0x38-0x3f */
};
```

Generated by Doxygen
static int tcvn_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
{
  unsigned char c = *s;
  if (c < 0x20)
    *pwc = (ucs4_t) tcvn_2uni_1[c];
  else if (c < 0x80)
    *pwc = (ucs4_t) c;
  else
    *pwc = (ucs4_t) tcvn_2uni_2[c-0x80];
  return 1;
}

static const unsigned char tcvn_page00[96+184] = {
  0xa0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xa0-0xa7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xa8-0xaf */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xb0-0xb7 */
  0x80, 0x83, 0xa2, 0x82, 0x00, 0x00, 0x00, 0x00, / * 0xc0-0xc7 */
  0x87, 0x8a, 0xa3, 0x00, 0x8d, 0x90, 0x00, 0x00, / * 0xc8-0xcf */
  0x00, 0x00, 0x92, 0x95, 0xa4, 0x94, 0x00, 0x00, / * 0xd0-0xd7 */
  0x00, 0x9d, 0x01, 0x00, 0x00, 0x16, 0x00, 0x00, / * 0xd8-0xdf */
  0xdb, 0x8a, 0xa9, 0xab, 0x00, 0x00, 0x00, 0x00, / * 0xe0-0xef */
  0xb5, 0xb8, 0xa9, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xf0-0xf7 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x00-0x07 */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x08-0x0f */
  0x8f, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x28-0x2f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x30-0x3f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x40-0x4f */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0x50-0x5f */
  0x84, 0xb9, 0x81, 0xb6, 0xc4, 0xca, 0xc1, 0xc7, / * 0xa0-0xa7 */
  0xc2, 0xc8, 0xc3, 0xc9, 0x86, 0x85, 0x64, 0xcb, / * 0xc0-0xc7 */
  0x89, 0xce, 0xc5, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xc8-0xcf */
  0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xe0-0xef */
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, / * 0xf0-0xff */
};

Generated by Doxygen
108 0x97, 0x99, 0x9b, 0xed, 0x98, 0xea, 0x99, 0xeb, /* 0x88-0x9f */
109 0x9a, 0x9c, 0xea, 0x02, 0xf4, 0xe9, 0xff, /* 0xe0-0xe7 */
110 0x9a, 0x9c, 0xea, 0x02, 0xf4, 0xe9, 0xff, /* 0xe0-0xe7 */
111 0x11, 0xf8, 0x04, 0xf5, 0x05, 0xf6, 0x06, 0xf7, /* 0xe8-0xef */
112 0x12, 0xf9, 0x13, 0xfa, 0x17, 0xfe, 0x14, 0xfb, /* 0xf0-0xf7 */
113 0x15, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xf8-0xff */
114 
115 static int
tcvn_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
116 {
117     unsigned char c = 0;
118     if (wc < 0x0080 && (wc >= 0x0020 || (0x00fe0076 & (1 << wc)) == 0)) {
119         *r = wc;
120         return 1;
121     } else if (wc >= 0x00a0 && wc < 0x01b8)
122         c = tcvn_page00[wc-0x00a0];
123     else if (wc >= 0x0300 && wc < 0x0328)
124         c = tcvn_page03[wc-0x0300];
125     else if (wc >= 0x1ea0 && wc < 0x1f00)
126         c = tcvn_page1e[wc-0x1ea0];
127     if (c != 0) {
128         *r = c;
129         return 1;
130     }
131     return RET_ILSEQ;
132 }
133 }
134 
34.289 tis620.h
1 /* $XFree86: xc/lib/X11/lcUniConv/tis620.h,v 1.4 2001/02/09 00:02:54 dawes Exp $ */
2
3 #ifndef _TIS620_H_
4 #define _TIS620_H_
5 #endif
6
7 static const unsigned short tis620_2uni[96] = {
8     /* 0xa0 */
9         0xfffd, 0x0e01, 0x0e02, 0x0e03, 0x0e04, 0x0e05, 0x0e06, 0x0e07,
10         0x0e08, 0x0e09, 0x0e0a, 0x0e0b, 0x0e0c, 0x0e0d, 0x0e0e, 0x0e0f,
11     /* 0xb0 */
12         0x0e10, 0x0e11, 0x0e12, 0x0e13, 0x0e14, 0x0e15, 0x0e16, 0x0e17,
13         0x0e18, 0x0e19, 0x0e1a, 0x0e1b, 0x0e1c, 0x0e1d, 0x0e1e, 0x0e1f,
14     /* 0xc0 */
15         0x0e20, 0x0e21, 0x0e22, 0x0e23, 0x0e24, 0x0e25, 0x0e26, 0x0e27,
16         0x0e28, 0x0e29, 0x0e2a, 0x0e2b, 0x0e2c, 0x0e2d, 0x0e2e, 0x0e2f,
17     /* 0xd0 */
18         0x0e30, 0x0e31, 0x0e32, 0x0e33, 0x0e34, 0x0e35, 0x0e36, 0x0e37,
19         0x0e38, 0x0e39, 0x0e3a, 0xfffd, 0xfffd, 0xfffd, 0xfffd, 0x0e3f,
20     /* 0xe0 */
21         0x0e40, 0x0e41, 0x0e42, 0x0e43, 0x0e44, 0x0e45, 0x0e46, 0x0e47,
22         0x0e48, 0x0e49, 0x0e4a, 0x0e4b, 0x0e4c, 0x0e4d, 0x0e4e, 0x0e4f,
23     /* 0xf0 */
24         0x0e50, 0x0e51, 0x0e52, 0x0e53, 0x0e54, 0x0e55, 0x0e56, 0x0e57,
25         0x0e58, 0x0e59, 0x0e5a, 0x0e5b, 0xfffd, 0xfffd, 0xfffd, 0xfffd,
26     }
27
28 static int
tis620_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
29 {
30     unsigned char c = *s;
31     if (c < 0x80) {
32         *pwc = (ucs4_t) c;
33         return 1;
34     } else if (c < 0xa0) {
35         return 1;
36     } else {
37         unsigned short wc = tis620_2uni[c-0xa0];
38         if (wc != 0xfffd) {
39             *pwc = (ucs4_t) wc;
40             return 1;
41         }
42     }
43     return RET_ILSEQ;
44 }
45
46 static const unsigned char tis620_page0e[96] = {
47     /* 0xa0 */
48         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, /* 0x00-0x07 */
49         0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, /* 0x08-0x0f */
50         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, /* 0x10-0x17 */
51         0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, /* 0x18-0x1f */
52         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, /* 0x20-0x27 */
53         0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, /* 0x28-0x2f */
54         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, /* 0x30-0x37 */
58 Generated by Doxygen
34.290 ucs2be.h

1 /*
2 * UCS-2BE = UCS-2 big endian
3 */
5
6 static int
7 ucs2be_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
8 {
9     unsigned char c = 0;
10     if (n >= 2) {
11         if (s[0] >= 0xd8 && s[0] < 0xe0) {
12             return RET_ILSEQ;
13         } else if (wc >= 0x0e00 && wc < 0x0e60)
14             c = tis620_page0e[wc-0x0e00];
15         if (c != 0) {
16             *r = c;
17             return 1;
18         }
19     } else if (wc >= 0x0e00 && wc < 0x0e60)
20         c = tis620_page0e[wc-0x0e00];
21     if (c != 0) {
22         *r = c;
23         return 1;
24     }
25     return RET_TOOFEW;
26 }
27
28 static int
29 ucs2be_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
30 {
31     unsigned char c = 0;
32     if (wc < 0x10000 && !(wc >= 0xd800 && wc < 0xe000)) {
33         if (n >= 2) {
34             r[0] = (unsigned char) (wc >> 8);
35             r[1] = (unsigned char) wc;
36             return 2;
37         } else
38             return RET_TOOFEW;
39     } else if (c != 0) {
40         *r = c;
41         return 1;
42     }
43     return RET_ILSEQ;
44 }

34.291 utf8.h

1 /*
2 * UTF-8
3 */
5
6 static int
7 utf8_mbtowc (conv_t conv, ucs4_t *pwc, const unsigned char *s, int n)
8 {
9     if (n >= 2) {
10         if (s[0] >= 0x80) {
11             *pwc = c;
12             return 1;
13         } else if (c < 0xc2) {
14             return RET_TOOFEW(0);
15         } else if (c < 0xe0) {
16             if (n < 2)
17                 return RET_ILSEQ;
18         }
19         return RET_TOOFEW(0);
20     }
21     return RET_ILSEQ;
22 }
23
24 static int
25 utf8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
26 {
27     unsigned char c = 0;
28     if (wc < 0x10000 && !(wc >= 0xd800 && wc < 0xe000)) {
29         if (n >= 2) {
30             r[0] = (unsigned char) (wc >> 8);
31             r[1] = (unsigned char) wc;
32             return 2;
33         } else
34             return RET_TOOFEW;
35     } else if (c != 0) {
36         *r = c;
37         return 1;
38     }
39     return RET_ILSEQ;
40 }

Generated by Doxygen
if (s[1] ^ 0x80) < 0x40)
    return RET_ILSEQ;

*pwc = (ucs4_t) (c & 0x1f) « 6
      | (ucs4_t) (s[1] ^ 0x80);
return 2;
} else if (c < 0xf0) {
    if (n < 3)
      return RET_TOOFEW(0);
    if (((s[1] ^ 0x80) < 0x40
         && (s[2] ^ 0x80) < 0x40
         && (c >= 0xe1 || s[1] >= 0xa0))
      return RET_ILSEQ;
    *pwc = (ucs4_t) (c & 0x0f) « 12
      | (ucs4_t) (s[1] ^ 0x80) « 6
      | (ucs4_t) (s[2] ^ 0x80);
    return 3;
} else if (c < 0xf8) {
    if (n < 4)
      return RET_TOOFEW(0);
    if (((s[1] ^ 0x80) < 0x40
         && (s[2] ^ 0x80) < 0x40
         && (s[3] ^ 0x80) < 0x40
         && (c >= 0xf1 || s[1] >= 0x90))
      return RET_ILSEQ;
    *pwc = (ucs4_t) (c & 0x07) « 18
      | (ucs4_t) (s[1] ^ 0x80) « 12
      | (ucs4_t) (s[2] ^ 0x80) « 6
      | (ucs4_t) (s[3] ^ 0x80);
    return 4;
} else if (c < 0xfc) {
    if (n < 5)
      return RET_TOOFEW(0);
    if (((s[1] ^ 0x80) < 0x40
         && (s[2] ^ 0x80) < 0x40
         && (s[3] ^ 0x80) < 0x40
         && (s[4] ^ 0x80) < 0x40
         && (c >= 0xf9 || s[1] >= 0x88))
      return RET_ILSEQ;
    *pwc = (ucs4_t) (c & 0x03) « 24
      | (ucs4_t) (s[1] ^ 0x80) « 18
      | (ucs4_t) (s[2] ^ 0x80) « 12
      | (ucs4_t) (s[3] ^ 0x80) « 6
      | (ucs4_t) (s[4] ^ 0x80);
    return 5;
} else if (c < 0xfe) {
    if (n < 6)
      return RET_TOOFEW(0);
    if (((s[1] ^ 0x80) < 0x40
         && (s[2] ^ 0x80) < 0x40
         && (s[3] ^ 0x80) < 0x40
         && (s[4] ^ 0x80) < 0x40
         && (s[5] ^ 0x80) < 0x40
         && (c >= 0xfd || s[1] >= 0x84))
      return RET_ILSEQ;
    *pwc = (ucs4_t) (c & 0x01) « 30
      | (ucs4_t) (s[1] ^ 0x80) « 24
      | (ucs4_t) (s[2] ^ 0x80) « 18
      | (ucs4_t) (s[3] ^ 0x80) « 12
      | (ucs4_t) (s[4] ^ 0x80) « 6
      | (ucs4_t) (s[5] ^ 0x80);
    return 6;
} else
  return RET_ILSEQ;
}

static int utf8_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n) /* n == 0 is acceptable */
{
  int count;
  if (wc < 0x80)
    count = 1;
  else if (wc < 0x800)
    count = 2;
  else if (wc < 0x10000)
    count = 3;
  else if (wc < 0x20000)
    count = 4;
  else if (wc < 0x40000)
    count = 5;
  else if (wc <= 0x7fffffff)
    count = 6;
  else
    return RET_ILSEQ;
  if (n < count)
    return RET_TOOFEW();
  switch (count) /* note: code falls through cases! */
  {
    case 6: r[5] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x4000000;
    break;
    case 5: r[4] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x2000000;
    break;
    case 4: r[3] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x1000000;
    break;
    case 3: r[2] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x800000;
    break;
    case 2: r[1] = 0x80 | (wc & 0x3f); wc = wc >> 6; wc |= 0x0c0000;
    break;
    case 1: r[0] = wc;
  }
static const unsigned short viscii_2uni_1[32] = {
    /* 0x00 */
    0x0000, 0x0001, 0x1eb2, 0x0003, 0x0004, 0x1eb4, 0x1eaa, 0x0007,
    0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
    /* 0x10 */
    0x0010, 0x0011, 0x0012, 0x0013, 0x1ef6, 0x0015, 0x0016, 0x0017,
    0x0018, 0x1ef8, 0x001a, 0x001b, 0x001c, 0x001d, 0x1ef4, 0x001f,
};

static const unsigned short viscii_2uni_2[128] = {
    /* 0x80 */
    0x1ea0, 0x1eae, 0x1eb0, 0x1eb6, 0x1ea4, 0x1ea6, 0x1ea8, 0x1eac,
    0x1ebc, 0x1eb8, 0x1eba, 0x0150, 0x0151, 0x0152, 0x0153, 0x0154,
    /* 0x90 */
    0x0155, 0x0156, 0x0157, 0x0158, 0x0159, 0x015a, 0x015b, 0x015c,
    0x015d, 0x015e, 0x015f, 0x0160, 0x0161, 0x0162, 0x0163, 0x0164,
    /* 0xa0 */
    0x0165, 0x0166, 0x0167, 0x0168, 0x0169, 0x016a, 0x016b, 0x016c,
    0x016d, 0x016e, 0x016f, 0x0170, 0x0171, 0x0172, 0x0173, 0x0174,
    /* 0xb0 */
    0x0175, 0x0176, 0x0177, 0x0178, 0x0179, 0x017a, 0x017b, 0x017c,
    0x017d, 0x017e, 0x017f, 0x0180, 0x0181, 0x0182, 0x0183, 0x0184,
    /* 0xc0 */
    0x0185, 0x0186, 0x0187, 0x0188, 0x0189, 0x018a, 0x018b, 0x018c,
    0x018d, 0x018e, 0x018f, 0x0190, 0x0191, 0x0192, 0x0193, 0x0194,
    /* 0xd0 */
    0x0195, 0x0196, 0x0197, 0x0198, 0x0199, 0x019a, 0x019b, 0x019c,
    0x019d, 0x019e, 0x019f, 0x01a0, 0x01a1, 0x01a2, 0x01a3, 0x01a4,
    /* 0xe0 */
    0x01a5, 0x01a6, 0x01a7, 0x01a8, 0x01a9, 0x01aa, 0x01ab, 0x01ac,
    0x01ad, 0x01ae, 0x01af, 0x01b0, 0x01b1, 0x01b2, 0x01b3, 0x01b4,
    /* 0xf0 */
    0x01b5, 0x01b6, 0x01b7, 0x01b8, 0x01b9, 0x01ba, 0x01bb, 0x01bc,
    0x01bd, 0x01be, 0x01bf, 0x01c0, 0x01c1, 0x01c2, 0x01c3, 0x01c4,
    /* 0x0100 */
    0x01c5, 0x01c6, 0x01c7, 0x01c8, 0x01c9, 0x01ca, 0x01cb, 0x01cc,
    0x01cd, 0x01ce, 0x01cf, 0x01d0, 0x01d1, 0x01d2, 0x01d3, 0x01d4,
    0x01d5, 0x01d6, 0x01d7, 0x01d8, 0x01d9, 0x01da, 0x01db, 0x01dc,
    0x01dd, 0x01de, 0x01df, 0x01e0, 0x01e1, 0x01e2, 0x01e3, 0x01e4,
    0x01e5, 0x01e6, 0x01e7, 0x01e8, 0x01e9, 0x01ea, 0x01eb, 0x01ec,
    0x01ed, 0x01ee, 0x01ef, 0x01f0, 0x01f1, 0x01f2, 0x01f3, 0x01f4,
    0x01f5, 0x01f6, 0x01f7, 0x01f8, 0x01f9, 0x01fa, 0x01fb, 0x01fc,
    0x01fd, 0x01fe, 0x01ff, 0x0200, 0x0201, 0x0202, 0x0203, 0x0204,
    0x0205, 0x0206, 0x0207, 0x0208, 0x0209, 0x020a, 0x020b, 0x020c,
    0x020d, 0x020e, 0x020f, 0x0210, 0x0211, 0x0212, 0x0213, 0x0214,
    0x0215, 0x0216, 0x0217, 0x0218, 0x0219, 0x021a, 0x021b, 0x021c,
    0x021d, 0x021e, 0x021f, 0x0220, 0x0221, 0x0222, 0x0223, 0x0224,
    0x0225, 0x0226, 0x0227, 0x0228, 0x0229, 0x022a, 0x022b, 0x022c,
    0x022d, 0x022e, 0x022f, 0x0230, 0x0231, 0x0232, 0x0233, 0x0234,
    0x0235, 0x0236, 0x0237, 0x0238, 0x0239, 0x023a, 0x023b, 0x023c,
    0x023d, 0x023e, 0x023f, 0x0240, 0x0241, 0x0242, 0x0243, 0x0244,
    0x0245, 0x0246, 0x0247, 0x0248, 0x0249, 0x024a, 0x024b, 0x024c,
    0x024d, 0x024e, 0x024f, 0x0250, 0x0251, 0x0252, 0x0253, 0x0254,
    0x0255, 0x0256, 0x0257, 0x0258, 0x0259, 0x025a, 0x025b, 0x025c,
    0x025d, 0x025e, 0x025f, 0x0260, 0x0261, 0x0262, 0x0263, 0x0264,
static const unsigned char viscii_page00[96] = {
  0x80, 0xd5, 0xc4, 0xe4, 0x84, 0xa4, 0x85, 0xa5, /* 0xa0-0xa7 */
  0x86, 0xa6, 0x06, 0xe7, 0x87, 0xa7, 0x81, 0xa1, /* 0x8e-0xaf */
  0x82, 0xa2, 0x02, 0xe8, 0x88, 0xa8, 0x83, 0xa3, /* 0xb0-0xb7 */
  0x89, 0xa9, 0xc9, 0xe9, 0x89, 0xa9, 0x8a, 0xa5, /* 0xb8-0xbf */
  0x8b, 0x9b, 0x0b, 0xc6, 0x8b, 0x9b, 0x8c, 0x9c, /* 0xc0-0xc7 */
  0x8d, 0x9d, 0x0d, 0xc7, 0x8d, 0x9d, 0x8e, 0x9e, /* 0xc8-0xcf */
  0x8f, 0x9f, 0x0f, 0xc6, 0x8f, 0x9f, 0x90, 0xa0, /* 0xd0-0xdf */
  0xc3, 0xe3, 0x93, 0xb3, 0xc4, 0xe4, 0x94, 0xb4, /* 0xe0-0xef */
  0xc5, 0xe5, 0x95, 0xb5, 0xc6, 0xe6, 0x96, 0xb6, /* 0xf0-0xff */
  0x19, 0xdb, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x70-0x77 */
  0x78, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x88, 0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
  0x98, 0x9f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0xa0-0xa7 */
  0xa8, 0xaf, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x80-0x87 */
  0x88, 0x8f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0x90-0x97 */
};

static int
viscii_wctomb (conv_t conv, unsigned char *r, ucs4_t wc, int n)
{
  unsigned char c = 0;
  if (wc < 0x0080 && (wc >= 0x0020 || (0x42100064 & (1 « wc)) == 0)) {
    *r = wc;
    return 1;
  }
  else if (wc >= 0x00c0 && wc < 0x01b8)
    c = viscii_page00[wc-0x00c0];
  else if (wc >= 0x1ea0 && wc < 0x1f00)
    c = viscii_page1e[wc-0x1ea0];
  if (c != 0) {
    *r = c;
    return 1;
  }
  return RET_ILSEQ;
}

34.293  Ximint.h
34.294  Xlibint.h
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FL_DIAMOND_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_DIAMOND_UP_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_EMBOSSED_LABEL</td>
<td>Enumerations.H, 1191</td>
<td></td>
</tr>
<tr>
<td>_FL_ENGRAVED_LABEL</td>
<td>Enumerations.H, 1191</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_DOWN_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_ROUND_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GLEAM_THIN_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_DOWN_FRAME</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_ROUND_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_ROUND_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_THIN_DOWN_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_GTK_THIN_UP_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_ICON_LABEL</td>
<td>Enumerations.H, 1191</td>
<td></td>
</tr>
<tr>
<td>_FL_IMAGE_LABEL</td>
<td>Enumerations.H, 1191</td>
<td></td>
</tr>
<tr>
<td>_FL_MULTI_LABEL</td>
<td>Enumerations.H, 1191</td>
<td></td>
</tr>
<tr>
<td>_FL_OFLAT_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_OSHADOW_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_OVAL_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_OVAL_FRAME</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_BUTTON_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_DOWN_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_ROUND_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_ROUND_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_THIN_DOWN_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_THIN_DOWN_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_THIN_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_THIN_UP_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_UP_BOX</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_OXY_UP_FRAME</td>
<td>Enumerations.H, 1185</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_ROUND_UP_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_DOWN_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_THIN_UP_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_BOX</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
<tr>
<td>_FL_PLASTIC_UP_FRAME</td>
<td>Enumerations.H, 1184</td>
<td></td>
</tr>
</tbody>
</table>
add_idle
   FI, 409
add_key_binding
   FI_Text_Editor, 969
add_modify_callback
   FI_Text_Buffer, 915
add_scheme_name
   FI_Scheme, 799
add_system_handler
   Events handling functions, 277
add_timeout
   FI, 409
   FI_Timeout, 987
add_vertex
   FI_File_Icon, 554, 555
addPlugin
   FI_Plugin_Manager, 739
address
   FI_Text_Buffer, 915
align
   FI_Widget, 1099
alloc_size
   FI_Flex, 566
ALWAYS_ON
   FI_Browser, 476
angle1
   FI_Dial, 535
animate
   FI_GIF_Image, 580
ansi
   FI_Simple_Terminal, 830, 831
ansi_show_unknown
   FI_Simple_Terminal, 831, 832
api_version
   FI, 409
append
   FI_Input, 644
   FI_Simple_Terminal, 832
   FI_String, 851, 852
   FI_Text_Buffer, 915
appendfile
   FI_Text_Buffer, 916
apply_undo
   FI_Input, 644
arg
   FI, 410
args
   FI, 410
argument
   FI_Menu_Item, 702
   FI_Widget, 1099, 1100
armscii_8.h, 1538
array
   FI_Group, 597
   FI_RGB_Image, 794
   FI_Table, 886
arrow_widths
   FI_Counter, 531
as_double_window
   FI_Double_Window, 538
   FI_Window, 1142
as_gl_window
   FI_Gl_Window, 584
   FI_Widget, 1100
as_group
   FI_Group, 597
   FI_Widget, 1100
as_overlay_window
   FI_Overlay_Window, 724
   FI_Window, 1142
as_shared_image
   FI_Image, 625
   FI_Shared_Image, 819
as_svg_image
   FI_RGB_Image, 792
   FI_SVG_Image, 868
as_window
   FI_Widget, 1101
   FI_Window, 1142
ascii.h, 1539
assign
   FI_String, 852
at
   FI_String, 853
atclose
   Windows handling functions, 274
ATTR_BGCOLOR
   FI_Text_Display, 936
ATTR_BGCOLOR_EXT
   FI_Text_Display, 936
ATTR_BGCOLOR_EXT_
   FI_Text_Display, 936
ATTR_GRAMMAR
   FI_Text_Display, 936
ATTR_LINES_MASK
   FI_Text_Display, 936
ATTR_SPELLING
   FI_Text_Display, 936
ATTR_STRIKE_THROUGH
   FI_Text_Display, 936
ATTR_UNDERLINE
   FI_Text_Display, 936
autosize
   FI_Chart, 499
awake
   Multithreading support functions, 347
b
   FI_Color_Chooser, 523
   FI_Rect, 784
B0
   FI_Paged_Device, 730
B1
   FI_Paged_Device, 730
B10
   FI_Paged_Device, 730
B2  
Fl_Paged_Device, 730
box
Fl_Widget, 1102
box_border_radius_max
Fl, 411, 412
box_color
Fl, 412
box_dh
Fl, 412
box_dw
Fl, 412
box_dx
Fl, 412
box_dy
Fl, 413
box_shadow_width
Fl, 413
BROWSE_DIRECTORY
Fl_Native_File_Chooser, 718
BROWSE_FILE
Fl_Native_File_Chooser, 718
BROWSE_MULTI_DIRECTORY
Fl_Native_File_Chooser, 718
BROWSE_MULTI_FILE
Fl_Native_File_Chooser, 718
BROWSE_SAVE_DIRECTORY
Fl_Native_File_Chooser, 718
BROWSE_SAVE_FILE
Fl_Native_File_Chooser, 718
buffer
Fl_Text_Display, 937, 938
buffer_modified_cb
Fl_Text_Display, 938
buffer_predelete_cb
Fl_Text_Display, 938
byte_at
Fl_Text_Buffer, 916
C_LOCALE
Fl_Preferences, 756
c_str
Fl_String, 853
cairo_autolink_context
Fl_Native_File_Chooser, 718
cairo_autolink_context
Fl_Native_File_Chooser, 718
cairo_cc
Fl_Native_File_Chooser, 718
cairo_flush
Fl_Native_File_Chooser, 718
cairo_make_current
Fl_Native_File_Chooser, 718
calc_dimensions
Fl_Tree, 1012
calc_item_height
Fl_Tree_Item, 1050
calc_last_char
  Fl_Text_Display, 939

calc_line_starts
  Fl_Text_Display, 939

calc_tree
  Fl_Tree, 1013

callback
  Fl_Menu_Item, 702, 703
  Fl_Table, 886
  Fl_Widget, 1103, 1104

Callback Function Typedefs, 271
  Fl_Event_Dispatch, 272
  Fl_Timeout_Handler, 272

callback_col
  Fl_Table, 887

callback_context
  Fl_Table, 887

callback_item
  Fl_Tree, 1013

callback_reason
  Events handling functions, 278
  Fl_Tree, 1013, 1014

callback_row
  Fl_Table, 887

can_do
  Fl_Gl_Window, 584

can_do_overlay
  Fl_Gl_Window, 584

can_redo
  Fl_Input_, 645
  Fl_Text_Buffer, 916

can_undo
  Fl_Input_, 645
  Fl_Text_Buffer, 916
canUndo
  Fl_Text_Buffer, 916
canvas
  Fl_Anim_GIF_Image, 433
canvas_h
  Fl_Anim_GIF_Image, 434
canvas_w
  Fl_Anim_GIF_Image, 434
capacity
  Fl_String, 853

CARET_CURSOR
  Fl_Text_Display, 936
case.h, 1475
  cc
    Fl_Cairo_State, 494
cgdebug.h, 1431

CHANGED
  Fl_Widget, 1097
changed
  Fl_Widget, 1104
char_at
  Fl_Text_Buffer, 916
check
  Fl, 413
  Fl_Menu_Item, 703
checkbox
  Fl_Menu_Item, 703
checked
  Fl_Menu_Item, 703
child
  Fl_Group, 598
  Fl_Table, 887
  Fl_Tree_Item, 1050
children
  Fl_Table, 888
clear
  Fl_Browser, 455
  Fl_Button, 491
  Fl_Group, 598
  Fl_Menu_, 682
  Fl_Simple_Terminal, 832
  Fl_Sys_Menu_Bar, 875
  Fl_Table, 888
  Fl_Table_Row, 898
  Fl_Tree, 1014
  Fl_Tree_Item_Array, 1066

clear_active
  Fl_Widget, 1104
clear_border
  Fl_Widget, 1143
clear_changed
  Fl_Widget, 1105
clear_children
  Fl_Tree, 1014
clear_damage
  Fl_Widget, 1105
clear_modal_states
  Fl_Window, 1143
clear_output
  Fl_Widget, 1105
clear_rect
  Fl_Text_Display, 939
clear_submenu
  Fl_Menu_, 682
  Fl_Sys_Menu_Bar, 875
clear_visible
  Fl_Widget, 1105
clear_visible_focus
  Fl_Widget, 1105
clear_widget_pointer
    Safe widget deletion support functions, 349
client_area
  Fl Tabs, 905

CLIP_CHILDREN
  Fl_Widget, 1097
clip_children
  Fl_Group, 598, 599
clipboard_contains
    Selection & Clipboard functions, 290
close
  Fl_EPS_File_Surface, 541
  Fl_SVG_File_Surface, 864
<table>
<thead>
<tr>
<th>COLUMN</th>
<th>Fl_Flex, 564</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_char</td>
<td>Fl_Browser, 455</td>
</tr>
<tr>
<td>column_widths</td>
<td>Fl_Browser, 455</td>
</tr>
</tbody>
</table>

Common Dialog Classes and Functions, 372
- error, 388
- fatal, 388
- fl_alert, 374
- fl_ask, 374
- fl_beep, 375
- fl_choice, 375
- fl_choice_n, 376
- fl_color_chooser, 377, 378
- fl_dir_chooser, 379
- fl_file_chooser, 380
- fl_file_chooser_callback, 380
- fl_file_chooser_ok_label, 381
- fl_input, 381
- fl_input_str, 381, 382
- fl_message, 382
- fl_message_hotspot, 383
- fl_message_icon, 383
- fl_message_icon_label, 384
- fl_message_position, 384, 385
- fl_message_title, 386
- fl_message_title_default, 386
- fl_password, 386
- fl_password_str, 387
- warning, 388

compare
- Fl_Shared_Image, 820

compose
- Events handling functions, 278
- compose_reset
  - Events handling functions, 279

connectorstyle
- Fl_Tree, 1015

contains
- Fl_Widget, 1107

context
- Fl_Gl_Window, 585

CONTEXT_CELL
- Fl_Table, 886

CONTEXT_COL_HEADER
- Fl_Table, 886

CONTEXT_ENDPAGE
- Fl_Table, 886

CONTEXT_NONE
- Fl_Table, 886

CONTEXT_RC_RESIZE
- Fl_Table, 886

CONTEXT_ROW_HEADER
- Fl_Table, 886

CONTEXT_STARTPAGE
- Fl_Table, 886

CONTEXT_TABLE
| 1822 | INDEX |

<table>
<thead>
<tr>
<th>Fl_Flex, 1014, 1015</th>
</tr>
</thead>
<tbody>
<tr>
<td>closedeicon</td>
</tr>
<tr>
<td>Fl_Tree_Prefs, 1071</td>
</tr>
<tr>
<td>closeicon</td>
</tr>
<tr>
<td>Fl_Tree, 1015</td>
</tr>
<tr>
<td>Fl_Tree_Prefs, 1071</td>
</tr>
<tr>
<td>col_header</td>
</tr>
<tr>
<td>Fl_Table, 888</td>
</tr>
<tr>
<td>col_resize</td>
</tr>
<tr>
<td>Fl_Table, 888</td>
</tr>
<tr>
<td>col_resize_min</td>
</tr>
<tr>
<td>Fl_Table, 888</td>
</tr>
<tr>
<td>col_to_x</td>
</tr>
<tr>
<td>Fl_Text_Display, 939</td>
</tr>
<tr>
<td>col_width</td>
</tr>
<tr>
<td>Fl_Table, 888</td>
</tr>
<tr>
<td>col_width_all</td>
</tr>
<tr>
<td>Fl_Table, 889</td>
</tr>
<tr>
<td>color</td>
</tr>
<tr>
<td>Fl_Tooltip, 997</td>
</tr>
<tr>
<td>Fl_Widget, 1106</td>
</tr>
<tr>
<td>Color &amp; Font functions, 299</td>
</tr>
<tr>
<td>fl_color, 301</td>
</tr>
<tr>
<td>fl_color_average, 301</td>
</tr>
<tr>
<td>fl_contrast, 302</td>
</tr>
<tr>
<td>fl_contrast_function, 303</td>
</tr>
<tr>
<td>fl_contrast_level, 303, 304</td>
</tr>
<tr>
<td>fl_contrast_mode, 305</td>
</tr>
<tr>
<td>fl_font, 306</td>
</tr>
<tr>
<td>fl_height, 306</td>
</tr>
<tr>
<td>fl_latin1_to_local, 306</td>
</tr>
<tr>
<td>fl_lightness, 307</td>
</tr>
<tr>
<td>fl_local_to_latin1, 307</td>
</tr>
<tr>
<td>fl_local_to_mac_roman, 307</td>
</tr>
<tr>
<td>fl_luminance, 309</td>
</tr>
<tr>
<td>fl_mac_roman_to_local, 309</td>
</tr>
<tr>
<td>fl_show_colormap, 310</td>
</tr>
<tr>
<td>fl_size, 310</td>
</tr>
<tr>
<td>fl_text_extents, 310, 311</td>
</tr>
<tr>
<td>fl_width, 311</td>
</tr>
<tr>
<td>free_color, 311</td>
</tr>
<tr>
<td>get_color, 311, 312</td>
</tr>
<tr>
<td>get_font, 312</td>
</tr>
<tr>
<td>get_font_name, 312</td>
</tr>
<tr>
<td>get_font_sizes, 312</td>
</tr>
<tr>
<td>set_color, 313</td>
</tr>
<tr>
<td>set_font, 313</td>
</tr>
<tr>
<td>set_fonts, 314</td>
</tr>
<tr>
<td>color2</td>
</tr>
<tr>
<td>Fl_Widget, 1106, 1107</td>
</tr>
<tr>
<td>color_average</td>
</tr>
<tr>
<td>Fl_Anim_GIF_Image, 434</td>
</tr>
<tr>
<td>Fl_Image, 625</td>
</tr>
<tr>
<td>Fl_Pixmap, 735</td>
</tr>
<tr>
<td>Fl_RGB_Image, 792</td>
</tr>
<tr>
<td>Fl_Shared_Image, 820</td>
</tr>
<tr>
<td>Fl_SVG_Image, 869</td>
</tr>
<tr>
<td>Fl_Tiled_Image, 984</td>
</tr>
</tbody>
</table>

Generated by Doxygen
INDEX

FL_Table, 886
context_valid
  FL_Gl_Window, 585
COPYED_LABEL
  FL_Widget, 1097
COPYED_TOOLTIP
  FL_Widget, 1097
copy
  FL_Anim_GIF_Image, 434
  FL_Bitmap, 444
  FL_Image, 626
  FL_Input_, 645
  FL_Menu_, 683
  FL_Pixmap, 735
  FL_RGB_Image, 792
  FL_Shared_Image, 820
  FL_SVG_Image, 869
  FL_Text_Buffer, 917
  FL_Tiled_Image, 984
Selection & Clipboard functions, 290
copy_cuts
  FL_Input_, 645
copy_label
  FL_Widget, 1107
copy_tooltip
  FL_Widget, 1107
CORE
  FL_Preferences, 756
CORE_READ_OK
  FL_Preferences, 771
CORE_SYSTEM
  FL_Preferences, 756
CORE_SYSTEM_L
  FL_Preferences, 756
CORE_USER
  FL_Preferences, 756
CORE_USER_L
  FL_Preferences, 756
CORE_WRITE_OK
  FL_Preferences, 771
count
  FL_Image, 627
  FL_Native_File_Chooser, 718
count_displayed_characters
  FL_Text_Buffer, 917
count_lines
  FL_Text_Buffer, 917
  FL_Text_Display, 940
cp1133.h, 1589
cp1251.h, 1590
cp1255.h, 1591
cp1256.h, 1593
cp936ext.h, 1594
create_window_menu
  FL_Sys_Menu_Bar, 875
current
  FL_Group, 599
  FL_Timeout, 987
  FL_Tooltip, 998
  FL_Window, 1143
current_
  FL_Window, 1159
current_style
  FL_Simple_Terminal, 832
current_style_index
  FL_Simple_Terminal, 833
current_timeout
  FL_Timeout, 992
cursor
  FL_Window, 1144
cursor2rowcol
  FL_Table, 889
cursor_color
  FL_Input_, 646
  FL_Text_Display, 940
  FL_Value_Input, 1080
cursor_style
  FL_Text_Display, 940
custom_application_menu_items
  FL_Mac_App_Menu, 676
cut
  FL_Input_, 646, 647
d
  FL_Image, 627
damage
  FL_Widget, 1108
damage_zone
  FL_Table, 889
data
  FL_Browser, 456
  FL_Image, 627
  FL_String, 853
deactivate
  FL_Menu_Item, 703
  FL_Tree_Item, 1050
  FL_Widget, 1109
debug
  FL_String, 854
DEBUG_FLAG
  FL_Anim_GIF_Image, 432
decorated_h
  FL_Window, 1144
decorated_w
  FL_Window, 1145
default_atclose
  Windows handling functions, 273
default_callback
  FL_Widget, 1109
default_cursor
  FL_Window, 1145
default_icon
  FL_Window, 1145
default_icons
  FL_Window, 1146
default_size_range
  FL_Window, 1147
default_value
  Fl_Shortcut_Button, 825
default_xclass
  Fl_Window, 1147, 1148
deimage
  Fl_Widget, 1109, 1110
DEIMAGE_BOUND
  Fl_Widget, 1097
deimage_bound
  Fl_Widget, 1110
delay
  Fl_Anim_GIF_Image, 435
  Fl_Tooltip, 998
delete_child
  Fl_Group, 599
  Fl_Scroll, 806
delete_entry
  Fl_Preferences, 759
delete_group
  Fl_Preferences, 760
delete_widget
  Safe widget deletion support functions, 349
deleted
  Fl_Widget_Tracker, 1136
deleting
  Fl_Browser_, 477
deparent
  Fl_Tree_Item, 1051
  Fl_Tree_Item_Array, 1066
depth
  Fl_Tree_Item, 1051
desaturate
  Fl_Anim_GIF_Image, 435
  Fl_Image, 628
  Fl_Pixmap, 735
  Fl_RGB_Image, 793
  Fl_Shared_Image, 821
  Fl_SVG_Image, 870
  Fl_Tiled_Image, 985
deselect
  Fl_Browser_, 477
  Fl_Tree, 1016
deselect_all
  Fl_Group, 1017
  Fl_Tree_Item, 1051
DIM_CURSOR
  Fl_Text_Display, 936
dingbats_.h, 1495
direction
  Fl_Timer, 994
directory
  Fl_Native_File_Chooser, 718
dirty
  Fl_Preferences, 760
disable
  Fl_Tooltip, 998
disable_im
  Events handling functions, 279
display
  Fl, 413
  Fl_Browser, 456
  Fl_Browser_, 477
  Fl_Tree, 1017
display_insert
  Fl_Text_Display, 941
displayed
  Fl_Browser, 456
  Fl_Browser_, 477
  Fl_Tree, 1017
dnd
  Selection & Clipboard functions, 291
dnd_text_ops
  Fl, 414
do_callback
  Fl_Menu_Item, 703, 704
  Fl_Table, 889
  Fl_Widget, 1110, 1111
do_widget_deletion
  Safe widget deletion support functions, 350
DONT_RESIZE_CANVAS
  Fl_Anim_GIF_Image, 432
DONT_SET_AS_IMAGE
  Fl_Anim_GIF_Image, 432
DONT_START
  Fl_Anim_GIF_Image, 432
down_box
  Fl_Button, 491
  Fl_File_Input, 559
  Fl_Menu_, 683
draw
  Fl_Adjuster, 428
  Fl_Anim_GIF_Image, 435
  Fl_Bitmap, 445
  Fl_Box, 449
  Fl_Browser_, 478
  Fl_Button, 491
  Fl_Cairo_Window, 496
  Fl_Chart, 500
  Fl_Choice, 512
  Fl_Clock_Output, 518, 519
  Fl.Counter, 531
  Fl.Dial, 535
  Fl_File_Icon, 555
  Fl_File_Input, 559
  Fl_FormsBitmap, 573
  Fl_FormsPixmap, 574
  Fl_FormsText, 575
  Fl_Free, 577
  Fl_GL_Window, 585
  Fl_Glut_Window, 592
  Fl_Group, 600
  Fl_Help_View, 615
  Fl_Image, 628
  Fl_Input, 639
  Fl_Label, 672
  Fl_Light_Button, 674
INDEX 1825

FL_Menu_Bar, 692
FL_Menu_Button, 696
FL_Pack, 727
FL_Pixmap, 736
FL_Positioner, 743
FL_Progress, 780
FL_Return_Button, 788
FL_Roller, 796
FL_Scroll, 806
FL_Scrollbar, 812
FL_Shortcut_Image, 821
FL_Shortcut_Button, 826
FL_Simple_Terminal, 833
FL_Slider, 841
FL_Spinner, 845
FL_SVG_Image, 870
FL_Sys_Menu_Bar, 876
FL_Table, 889
FL_Tab, 905
FL_Text_Display, 941
FL_Tiled_Image, 985
FL_Timer, 994
FL_Tree, 1018
FL_Tree_Item, 1051
FL_Value_Input, 1080
FL_Value_Output, 1084
FL_Value_Slider, 1087
FL_Widget, 1112
FL_Widget_Surface, 1133
FL_Window, 1148
FL_Wizard, 1160
draw_barchart
FL_Chart, 500
draw_begin
FL_Gl_Window, 586
draw_box_active
FL, 414
draw_cell
FL_Table, 889
draw_child
FL_Group, 600
draw_children
FL_Group, 600
draw_cursor
FL_Text_Display, 941
draw_decorated_window
FL_Widget_Surface, 1134
draw_empty
FL_Image, 628
draw_end
FL_Gl_Window, 586
draw_focus
FL_Widget, 1112
draw_GL_text_with_textures
FL, 414
draw_horbarchart
FL_Chart, 500
draw_horizontal_connector
FL_Tree_Item, 1052
draw_item_content
FL_Tree_Item, 1052
draw_label
FL_Widget, 1113
draw_line_numbers
FL_Text_Display, 941
draw_linechart
FL_Chart, 501
draw_overlay
FL_Glut_Window, 593
FL_Overlay_Window, 724
draw_piechart
FL_Chart, 501
draw_range
FL_Text_Display, 942
draw_scaled
FL_Image, 629
draw_string
FL_Text_Display, 942
draw_text
FL_Text_Display, 942
draw_vertical_connector
FL_Tree_Item, 1053
draw_vline
FL_Text_Display, 943
drawbgcolor
FL_Tree_Item, 1053
drawfgcolor
FL_Tree_Item, 1053
Drawing functions, 314
fl_add_symbol, 320
fl_antialias, 320
fl_arc, 320, 321
fl_begin_complex_polygon, 322
fl_begin_offscreen, 322
fl_begin_points, 322
fl_can_do_alpha_blending, 322
FL_CAP_FLAT, 319
FL_CAP_ROUND, 319
FL_CAP_SQUARE, 319
fl_capture_window, 322
fl_circle, 323
fl_clip, 323
fl_clip_box, 324
fl_clip_region, 324, 325
fl_copy_offscreen, 325
fl_create_offscreen, 325
fl_cursor, 326
fl_curve, 326
FL_DASH, 319
FL_DASHDOT, 319
FL_DASHDOTDOT, 319
fl_delete_offscreen, 326
FL_DOT, 319
fl_draw, 327
fl_draw_arrow, 328

Generated by Doxygen
fl_draw_box, 328
fl_draw_check, 328
fl_draw_circle, 329
fl_draw_image, 329, 330
fl_draw_image_mono, 330, 331
fl_draw_pixmap, 331
fl_draw_symbol, 332
fl_expand_text, 332
fl_focus_rect, 332
fl_frame, 332
fl_frame2, 333
fl_gap, 333
FL_JOIN_BEVEL, 319
FL_JOIN_MITER, 319
fl_line_style, 333
fl_load_matrix, 334
fl_measure, 334
fl_measure_pixmap, 335
fl_mult_matrix, 335
fl_not_clipped, 336
fl_old_shortcut, 336
fl_overlay_clear, 337
fl_overlay_rect, 337
fl_override_scale, 338
fl_pie, 338
fl_polygon, 339
fl_pop_clip, 339
fl_push_clip, 339
fl_push_matrix, 340
fl_read_image, 340
fl_rect, 340
fl_rectf, 340, 341
fl_rescale_offscreen, 341
fl_reset_spot, 341
fl_restore_scale, 341
fl_rotate, 342
fl_rounded_rect, 342
fl_rounded_rectf, 342
fl_scale, 342
fl_scroll, 343
fl_set_spot, 343
fl_set_status, 343
fl_shortcut_label, 344
FL_SOLID, 319
fl_transform_dx, 345
fl_transform_dy, 345
fl_transform_x, 345
fl_transform_y, 345
fl_transformed_vertex, 346
fl_translate, 346
fl_vertex, 346
drawtext
FI_String, 854
empty_vlines
FI_Text_Display, 943
enable
FI_Tooltip, 998
enable_im
Events handling functions, 279
enabled
FI_Tooltip, 998
end
FI_Flex, 566
FI_Group, 600
FI_Text_Selection, 976
end_current
FI_PostScript_File_Device, 747
FI_Surface_Device, 861
end_job
FI_Paged_Device, 731
FI_PostScript_File_Device, 748
FI_Printer, 775
end_page
FI_Paged_Device, 731
FI_PostScript_File_Device, 748
FI_Printer, 776
enforce_history_lines
FI_Simple_Terminal, 833
enforce_stay_at_bottom
FI_Simple_Terminal, 833
enter_area
FI_Tooltip, 998
entries
FI_Preferences, 760
current
FI_Preferences, 760
ten_exists
FI_Preferences, 761
Enumerations.H, 1169, 1194
_FL_DIAMOND_DOWN_BOX, 1184
_FL_DIAMOND_UP_BOX, 1184
_FL_EMBOSSED_LABEL, 1191
_FL_ENGRAVED_LABEL, 1191
_FL_GLEAM_DOWN_BOX, 1185
_FL_GLEAM_DOWN_FRAME, 1185
_FL_GLEAM_ROUND_DOWN_BOX, 1185
_FL_GLEAM_ROUND_UP_BOX, 1185
_FL_GLEAM_THIN_DOWN_BOX, 1185
_FL_GLEAM_THIN_DOWN_FRAME, 1185
_FL_GLEAM_THIN_UP_BOX, 1185
_FL_GLEAM_THIN_UP_FRAME, 1185
_FL_GTK_DOWN_BOX, 1184
_FL_GTK_DOWN_FRAME, 1184
_FL_GTK_ROUND_DOWN_BOX, 1185
_FL_GTK_ROUND_UP_BOX, 1185
_FL_GTK_THIN_DOWN_BOX, 1184
_FL_GTK_THIN_DOWN_FRAME, 1185
_FL_GTK_THIN_UP_BOX, 1184
_FL_GTK_THIN_UP_FRAME, 1185
_FL_GTK_UP_BOX, 1184

Generated by Doxygen
FL_FRAME, 1193
FL_FREE_BOXTYPE, 1185
FL_FREE_LABELTYPE, 1191
FL_FULLSCREEN, 1190
fl_gray_ramp, 1194
FL_HIDE, 1190
FL_IMAGE_LABEL, 1181
FL_KEYBOARD, 1189
FL_KEYDOWN, 1189
FL_KEYUP, 1189
FL_Labeltype, 1190
FL_LEAVE, 1188
FL_MAJOR_VERSION, 1181
FL_MINOR_VERSION, 1181
FL_MOUSEWHEEL, 1190
FL_MOVE, 1189
FL_MULTI_LABEL, 1190
FL_NO_BOX, 1183
FL_NO_EVENT, 1188
FL_NO_LABEL, 1191
FL_NORMAL_LABEL, 1191
FL_NORMAL_SIZE, 1194
FL_ORIENTATION, 1191
FL_PATCH_VERSION, 1181
FL_PUSH, 1188
FL_READ, 1183
FL_REASON_CANCELLED, 1186
FL_REASON_CHANGED, 1186
FL_REASON_CLOSED, 1186
FL_REASON_DESELECTED, 1186
FL_REASON_DRAGGED, 1186
FL_REASON_GOT_FOCUS, 1186
FL_REASON_LOST_FOCUS, 1186
FL_REASON_OPENED, 1186
FL_REASON_RELEASED, 1186
FL_REASON_RESELECTED, 1186
FL_REASON_SELECTED, 1186
FL_REASON_UNKNOWN, 1186
FL_RELEASE, 1188
FL_SCREEN_CONFIGURATION_CHANGED, 1190
FL_SELECTIONCLEAR, 1190
FL_SHORTCUT, 1189
FL_SHOW, 1190
FL_SYMBOL_LABEL, 1181
FL_THIN_DOWN_BOX, 1184
FL_THIN_DOWN_FRAME, 1184
FL_THIN_UP_BOX, 1184
FL_THIN_UP_FRAME, 1184
FL_UNFOCUS, 1188
FL_UP_BOX, 1184
FL_UP_FRAME, 1184
FL_VERSION, 1182
FL_When, 1192
FL_WHEN_CHANGED, 1192
FL_WHEN_CLOSED, 1192
FL_WHEN_ENTER_KEY, 1192
FL_WHEN_ENTER_KEY_ALWAYS, 1192
FL_WHEN_ENTER_KEY_CHANGED, 1192
FL_WHEN_NEVER, 1192
FL_WHEN_NOT_CHANGED, 1192
FL_WHEN_RELEASE, 1192
FL_WHEN_RELEASE_ALWAYS, 1192
FL_WRITE, 1183
FL_ZOOM_EVENT, 1190
FL_ZOOM_GESTURE, 1190
erase
FL_String, 854
errmsg
FL_File_Browser, 544
FL_Native_File_Chooser, 718
error
Common Dialog Classes and Functions, 388
errorcolor
FL_File_Input, 559, 560
ERRORS_TO_CP1252
Unicode and UTF-8 functions, 355
ERRORS_TO_ISO8859_1
Unicode and UTF-8 functions, 355
event
Events handling functions, 279
event_button
Events handling functions, 279
event_button1
Events handling functions, 280
event_button2
Events handling functions, 280
event_button3
Events handling functions, 280
event_buttons
Events handling functions, 280
event_clicks
Events handling functions, 280
event_clipboard
Events handling functions, 281
event_clipboard_type
Events handling functions, 281
event_dispatch
Events handling functions, 281
event_dx
Events handling functions, 281
event_dy
Events handling functions, 282
event_inside
Events handling functions, 282
event_is_click
Events handling functions, 283

- event_key
- event_length
- event_original_key
- event_state
- event_text
- event_x_root
- event_y_root

Events handling functions, 274

- add_handler, 277
- add_system_handler, 277
- belowmouse, 278
- callback_reason, 278
- compose, 278
- compose_reset, 279
- disable_im, 279
- enable_im, 279
- event, 279
- event_button, 279
- event_button1, 280
- event_button2, 280
- event_button3, 280
- event_buttons, 280
- event_clicks, 280
- event_clipboard, 281
- event_clipboard_type, 281
- event_dispatch, 281
- event_dx, 281
- event_dy, 282
- event_inside, 282
- event_is_click, 283
- event_key, 283
- event_length, 284
- event_original_key, 284
- event_state, 284
- event_text, 284
- event_x_root, 285
- event_y_root, 285

file_callback_reason_names, 288

fl_eventnames, 288

focus, 285

get_key, 285

get_mouse, 286

handle, 286

handle_, 286

pushed, 287

remove_handler, 287

remove_system_handler, 287

testShortcut, 288

executive

- Fl_Paged_Device, 730

exists

- Fl_Widget_Tracker, 1136

extend_range_for_styles

- Fl_Text_Display, 943

extend_selection

- Fl_Tree, 1018

extend_selection_dir

- Fl_Tree, 1018

fail

- Fl_Image, 629

fastarrow.h, 1434

def

Common Dialog Classes and Functions, 388

File names and URI utility functions, 388

- fl_decode_uri, 389
- Fl_File_Sort_F, 389
- fl_filename_absolute, 390
- fl_filename_expand, 390
- fl_filename_ext, 391
- fl_filename_free_list, 391
- fl_filename_isdir, 391
- fl_filename_list, 391
- fl_filename_match, 392
- fl_filename_name, 393
- fl_filename_relative, 393
- fl_filename_setext, 394
- fl_open_uri, 394

file_access

- Fl_Preferences, 761

file_encoding_warning_message

- Fl_Text_Buffer, 926

filename

- Fl_Native_File_Chooser, 719
- Fl_Preferences, 761

filename.H, 1202, 1203

filetype

- Fl_File_Browser, 544

filter

- Fl_File_Browser, 544
- Fl_File_Chooser, 551

- Fl_Native_File_Chooser, 719

filter_value

- Fl_Native_File_Chooser, 719

find

- Fl_File_Icon, 555
- Fl_Group, 600
- Fl_Help_View, 615
- Fl_Shared_Image, 822

find_cell

- Fl_Table, 891

find_chil

- Fl_Tree_Item, 1053, 1054

find_child_item

- Fl_Tree_Item, 1054

find_clicked

- Fl_Tree, 1019
- Fl_Tree_Item, 1054
find_index
    Fl_Menu_, 683, 684
find_item
    Fl_Browser_, 478
    Fl_Menu_, 685
    Fl_Tree, 1019
    Fl_Tree_Item, 1055
find_line
    Fl_Browser, 457
find_line_end
    Fl_Text_Display, 943
find_shortcut
    Fl_Menu_Item, 704
find_wrap_range
    Fl_Text_Display, 944
find_x
    Fl_Text_Display, 944
findchar_backward
    Fl_Text_Buffer, 917
findchar_forward
    Fl_Text_Buffer, 918
first
    Fl_Tree, 1020
first_selected_item
    Fl_Tree, 1020
first_timeout
    Fl_Timeout, 992
first_visible
    Fl_Tree, 1020
first_visible_item
    Fl_Tree, 1021
first_window
    Windows handling functions, 273
fix_scrollbar_order
    Fl_Scroll, 806
fixed
    Fl_Flex, 567
Fl, 397
    abi_check, 407
    abi_version, 408
    add_check, 408
    add_fd, 408
    add_idle, 409
    add_timeout, 409
    api_version, 409
    arg, 410
    args, 410
    background, 411
    background2, 411
    box_border_radius_max, 411, 412
    box_color, 412
    box_dh, 412
    box_dw, 412
    box_dx, 412
    box_dy, 413
    box_shadow_width, 413
    check, 413
    display, 413
dnd_text_ops, 414
    draw_box_active, 414
    draw_GL_text_with_textures, 414
    Fl_Option, 407
    flush, 415
    get_system_colors, 415
    gl_visual, 415
    has_timeout, 415
    help, 426
    hide_all_windows, 416
    idle, 426
    is_scheme, 416
    menu_linespacing, 416, 417
    now, 417
    option, 417, 418
    OPTION_ARROW_FOCUS, 407
    OPTION_DND_TEXT, 407
    OPTION_FNFC_USES_GTK, 407
    OPTION_FNFC_USES_ZENITY, 407
    OPTION_LAST, 407
    OPTION_PRINTER_USES_GTK, 407
    OPTION_SHOW_SCALING, 407
    OPTION_SHOW_TOOLTIP, 407
    OPTION_VISIBLE_FOCUS, 407
    own_colormap, 418
    program_should_quit, 419
    readqueue, 419
    ready, 419
    release, 419
    reload_scheme, 420
    remove_check, 420
    remove_timeout, 420
    repeat_timeout, 420
    run, 421
    scheme, 421
    scrollbar_size, 422
    seconds_between, 422
    seconds_since, 423
    set_box_color, 423
    set_idle, 423
    ticks_between, 423
    ticks_since, 424
    use_high_res_GL, 424
    version, 424
    visible_focus, 425
    visual, 425
    wait, 425
Fl_cxx, 1434
    fl_disable_wayland, 1435
    fl_find, 1435
    fl_open_display, 1435
Fl.H, 1204, 1206
FL_ABI_VERSION
    Enumerations.H, 1180
fl_access
    Unicode and UTF-8 functions, 356
FL_ACTIVATE
    Enumerations.H, 1189
fl__add_symbol
  Drawing functions, 320
Fl_Adjuster, 427
draw, 428
Fl_Adjuster, 428
handle, 428
soft, 428, 429
value_damage, 429
Fl_Adjuster.H, 1212
fl_alert
  Common Dialog Classes and Functions, 374
FL_ALIGN_LEFT
  Enumerations.H, 1194
FL_ALIGN_TOP
  Enumerations.H, 1194
Fl_Anim_GIF_Image, 429
  ~Fl_Anim_GIF_Image, 433
canvas, 433
canvas_h, 434
canvas_w, 434
color_average, 434
copy, 434
DEBUG_FLAG, 432
delay, 435
desaturate, 435
DONT_RESIZE_CANVAS, 432
DONT_SET_AS_IMAGE, 432
DONT_START, 432
draw, 435
Fl_Anim_GIF_Image, 432, 433
Flags, 432
frame, 436
frame_count, 436
frame_h, 437
frame_uncache, 437
frame_w, 437
frame_x, 438
frame_y, 438
frames, 438
image, 438
is_animated, 439
load, 439
LOG_FLAG, 432
loop, 442
min_delay, 442
name, 439
next, 439
on_extension_data, 440
on_frame_data, 440
OPTIMIZE_MEMORY, 432
playing, 440
resize, 440
speed, 441
start, 441
stop, 441
uncache, 441
valid, 441
Fl_Anim_GIF_Image.H, 1213
fl__antialias
  Drawing functions, 320
FL_API_VERSION
  Enumerations.H, 1181
fl_arc
  Drawing functions, 320, 321
fl_arc.hxx, 1436
FL_ARROW_CHOICE
  Enumerations.H, 1183
FL_ARROW_DOUBLE
  Enumerations.H, 1183
FL_ARROW_RETURN
  Enumerations.H, 1183
FL_ARROW_SINGLE
  Enumerations.H, 1183
Fl_Arrow_Type
  Enumerations.H, 1183
fl_ask
  Common Dialog Classes and Functions, 374
fl_ask.hxx, 1436
fl_ask.H, 1214, 1217
Fl_Beep, 1216
FL_BEEP_DEFAULT, 1216
FL_BEEP_ERROR, 1216
FL_BEEP_MESSAGE, 1216
FL_BEEP_NOTIFICATION, 1216
FL_BEEP_PASSWORD, 1216
FL_BEEP_QUESTION, 1216
fl_message_position, 1216
fl_attr.h, 1218, 1219
  __fl_attr, 1218
  FL_DEPRECATED, 1218
Fl_Beep
  fl_ask.H, 1216
fl_beep
  Common Dialog Classes and Functions, 375
FL_BEEP_DEFAULT
  fl_ask.H, 1216
FL_BEEP_ERROR
  fl_ask.H, 1216
FL_BEEP_MESSAGE
  fl_ask.H, 1216
FL_BEEP_NOTIFICATION
  fl_ask.H, 1216
FL_BEEP_PASSWORD
  fl_ask.H, 1216
FL_BEEP_QUESTION
  fl_ask.H, 1216
Fl_begin_complex_polygon
  Drawing functions, 322
fl_begin_offscreen
  Drawing functions, 322
fl_begin_points
  Drawing functions, 322
Fl_Bitmap, 442
copy, 444
draw, 445
Fl_Bitmap, 443, 444
label, 445
uncache, 446
Fl_Bitmap.H, 1221
Fl_BMP_Image, 446
Fl_BMP_Image, 446, 447
Fl_BMP_Image.H, 1221
FL_BORDER_BOX
Enumerations.H, 1184
FL_BORDER_FRAME
Enumerations.H, 1184
Fl_Box, 448
draw, 449
Fl_Box, 448
handle, 449
fl_box
Enumerations.H, 1192
Fl_Box.H, 1222
Fl_Boxtype
Enumerations.H, 1183
fl_boxtype.cxx, 1438
fl_internal_boxtype, 1439
fl_rectbound, 1439
Fl_Browser, 449
__remove, 454
add, 454
bottomline, 454
clear, 455
column_char, 455
column_widths, 455
data, 456
display, 456
displayed, 456
find_line, 457
Fl_Browser, 453
format_char, 457, 458
full_height, 458
hide, 458, 459
icon, 459
incr_height, 459
insert, 460
item_at, 460
item_draw, 461
item_first, 461
item_height, 461
item_last, 462
item_next, 462
item_prev, 462
item_select, 464
item_selected, 464
item_swap, 464
item_text, 465
item_width, 465
lineno, 465
lineposition, 466
load, 466
make_visible, 466
middeline, 467
move, 467
remove, 467
remove_icon, 468
select, 468
selected, 468
show, 468, 469
size, 469
swap, 469
text, 470
textsize, 470
topleft, 470, 471
value, 471
visible, 471
Fl_Browser.H, 1223
Fl_Browser_. 472
ALWAYS_ON, 476
bbox, 476
BOTH, 476
BOTH_ALWAYS, 476
deleting, 477
deselect, 477
display, 477
displayed, 477
draw, 478
find_item, 478
Fl_Browser_. 476
full_height, 478
full_width, 478
handle, 478
hasScrollbar, 479
HORIZONTAL, 476
HORIZONTAL_ALWAYS, 476
hposition, 479
hscrollbar, 488
incr_height, 480
inserting, 480
item_at, 480
item_draw, 480
item_first, 481
item_height, 481
item_last, 481
item_next, 481
item_prev, 481
item_quick_height, 482
item_select, 482
item_selected, 482
item_swap, 483
item_text, 483
item_width, 483
leftedge, 483
new_list, 484
position, 484
redraw_line, 484
redraw_lines, 484
replacing, 484
resize, 485
scrollbar, 488
scrollbar_left, 485
scrollbar_right, 485
FL_CLOSE
   Enumerations.H, 1189

fl_close_fd
   Unicode and UTF-8 functions, 358

fl_cmap
   fl_color.cxx, 1443
   fl_cmap.h, 1439

fl_color
   Color & Font functions, 301

fl_color.cxx, 1443
   fl_cmap, 1443

fl_color_average
   Color & Font functions, 301

Fl_Color_Chooser, 521
   b, 523
   Fl_Color_Chooser, 522
   g, 523
   handle, 523
   hsv, 524
   hsv2rgb, 524
   hue, 524
   mode, 524, 525
   r, 525
   rgb, 525
   rgb2hsv, 525
   saturation, 526
   value, 526

fl_color_chooser
   Common Dialog Classes and Functions, 377, 378

Fl_Color_Chooser.H, 1235, 1236

Fl_color_cube
   Enumerations.H, 1192

Fl-compose.cxx, 1443

fl_config.h, 1237

fl_contrast
   Color & Font functions, 302

fl_contrast.cxx, 1443

FL_CONTRAST_CIELAB
   Enumerations.H, 1186

FL_CONTRAST_CUSTOM
   Enumerations.H, 1186

Fl_Contrast_Function
   Enumerations.H, 1182

fl_contrast_function
   Color & Font functions, 303

FL_CONTRAST_LAST
   Enumerations.H, 1186

FL_CONTRAST_LEGACY
   Enumerations.H, 1186

fl_contrast_level
   Color & Font functions, 303, 304

Fl_Contrast_Mode
   Enumerations.H, 1186

fl_contrast_mode
   Color & Font functions, 305

FL_CONTRAST_NONE
   Enumerations.H, 1186

fl_copy_offscreen
   Drawing functions, 325

Fl_Copy_Surface, 526
   Fl_Copy_Surface, 527
   is_current, 527
   origin, 528
   printable_rect, 528
   set_current, 528
   translate, 529
   untranslate, 529

Fl_Copy_Surface.H, 1238

Fl_Counter, 529
   arrow_widths, 531
   draw, 531
   Fl_Counter, 531
   handle, 531
   lstep, 532
   step, 532

Fl_Counter.H, 1239

fl_create_offscreen
   Drawing functions, 325

Fl_Cursor
   Enumerations.H, 1186

fl_cursor
   Drawing functions, 326

FL_CURSOR_ARROW
   Enumerations.H, 1186

FL_CURSOR_CROSS
   Enumerations.H, 1186

FL_CURSOR_DEFAULT
   Enumerations.H, 1186

FL_CURSOR_E
   Enumerations.H, 1187

FL_CURSOR_HAND
   Enumerations.H, 1186

FL_CURSOR_HELP
   Enumerations.H, 1186

FL_CURSOR_INSERT
   Enumerations.H, 1186

FL_CURSOR_MOVE
   Enumerations.H, 1186

FL_CURSOR_N
   Enumerations.H, 1186

FL_CURSOR_NE
   Enumerations.H, 1187

FL_CURSOR_NESW
   Enumerations.H, 1186

FL_CURSOR_NONE
   Enumerations.H, 1187

FL_CURSOR_NS
   Enumerations.H, 1186

FL_CURSOR_NW
   Enumerations.H, 1187

FL_CURSOR_NWSE
   Enumerations.H, 1186

FL_CURSOR_S
   Enumerations.H, 1187

FL_CURSOR_SE
   Enumerations.H, 1187

Generated by Doxygen
INDEX 1835

FL_CURSOR_SW
Enumerations.H, 1187

FL_CURSOR_W
Enumerations.H, 1187

FL_CURSOR_WAIT
Enumerations.H, 1186

FL_CURSOR_WE
Enumerations.H, 1186

fl_curve
Drawing functions, 326

dl curve.cxx, 1444

Fl_Damage
Enumerations.H, 1187

FL_DAMAGE_ALL
Enumerations.H, 1187

FL DAMAGE_CHILD
Enumerations.H, 1187

FL DAMAGE_EXPOSE
Enumerations.H, 1187

FL DAMAGE_OVERLAY
Enumerations.H, 1187

FL DAMAGE_SCROLL
Enumerations.H, 1187

FL DAMAGE_USER1
Enumerations.H, 1187

FL DAMAGE_USER2
Enumerations.H, 1187

FL_DASH
Drawing functions, 319

FL DASHDOT
Drawing functions, 319

FL DASHDOTDOT
Drawing functions, 319

FL_DEACTIVATE
Enumerations.H, 1189

fl_decode_uri
File names and URI utility functions, 389

fl_define_FL_EMBOSSED_LABEL
Enumerations.H, 1193

fl_define_FL_ENGRAVED_LABEL
Enumerations.H, 1193

fl_define_FL_ICON_LABEL
Enumerations.H, 1193

fl_define_FL_IMAGE_LABEL
Enumerations.H, 1193

fl_define_FL_MULTI_LABEL
Enumerations.H, 1193

fl_define_FL_SHADOW_LABEL
Enumerations.H, 1193

fl_delete_offscreen
Drawing functions, 326

FL DEPRECATED
fl attr.h, 1218

Fl_Device.H, 1240

Fl_Device_Plugin, 533

rectangle_capture, 533

Fl_Dial, 534

angle1, 535
draw, 535

Fl_Dial, 535

handle, 536

Fl_Dial.H, 1241

fl dir_choose

Common Dialog Classes and Functions, 379

fl disable_wayland

Fl.cxx, 1435

Fl_Display_Device, 536

FL_DND_DRAG
Enumerations.H, 1190

FL_DND_ENTER
Enumerations.H, 1190

FL_DND_LEAVE
Enumerations.H, 1190

FL_DND_RELEASE
Enumerations.H, 1190

FL_DOT
Drawing functions, 319

Fl_Double_Window, 537

~Fl_Double_Window, 538

as double_window, 538

flush, 538

hide, 538

resize, 538

show, 539

Fl_Double_Window.cxx, 1444

Fl_Double_Window.H, 1241

fl down

Enumerations.H, 1193

FL_DOWN_BOX
Enumerations.H, 1184

FL_DOWN_FRAME
Enumerations.H, 1184

Fl dragging

Enumerations.H, 1188

fl draw

Drawing functions, 327

fl draw.H, 1242, 1248

fl draw arrow

Drawing functions, 328

fl draw box

Drawing functions, 328

fl draw check

Drawing functions, 328

fl draw circle

Drawing functions, 329

fl draw image

Drawing functions, 329, 330

fl draw image mono

Drawing functions, 330, 331

fl draw pixmap

Drawing functions, 331

fl draw symbol

Drawing functions, 332

FL_EMBOSSED_BOX
Enumerations.H, 1184

FL EMBOSSED FRAME

Generated by Doxygen
<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL_End</td>
<td>539</td>
</tr>
<tr>
<td>FL_ENGRAVED_BOX</td>
<td>Enumerations.H, 1184</td>
</tr>
<tr>
<td>FL_ENGRAVED_FRAME</td>
<td>Enumerations.H, 1184</td>
</tr>
<tr>
<td>FL_ENTER</td>
<td>Enumerations.H, 1188</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>540</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>close, 541</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>origin, 542</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>printable_rect, 542</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>translate, 542</td>
</tr>
<tr>
<td>FL_EPS_File_Surface</td>
<td>untranslate, 543</td>
</tr>
<tr>
<td>FL_EVENT</td>
<td>Enumerations.H, 1187</td>
</tr>
<tr>
<td>fl_eventnames</td>
<td>Events handling functions, 288</td>
</tr>
<tr>
<td>FL_EXCEPT</td>
<td>Enumerations.H, 1183</td>
</tr>
<tr>
<td>fl_expand_text</td>
<td>Drawing functions, 332</td>
</tr>
<tr>
<td>FL_Export.H</td>
<td>1253</td>
</tr>
<tr>
<td>FL_File_Browser</td>
<td>543</td>
</tr>
<tr>
<td>FL_File_Browser</td>
<td>errmsg, 544</td>
</tr>
<tr>
<td>FL_File_Browser</td>
<td>filetype, 544</td>
</tr>
<tr>
<td>FL_File_Browser</td>
<td>filter, 544, 545</td>
</tr>
<tr>
<td>FL_File_Browser</td>
<td>load, 545</td>
</tr>
<tr>
<td>FL_File_Browser.H</td>
<td>1254</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>545</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>add_extra, 551</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>filter, 551</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>load, 550</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>isize, 551</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>preview, 552</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>showHiddenButton, 552</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>shown, 552</td>
</tr>
<tr>
<td>FL_File_Chooser</td>
<td>value, 552</td>
</tr>
<tr>
<td>fl_file_chooser</td>
<td>Common Dialog Classes and Functions, 380</td>
</tr>
<tr>
<td>FL_File_Chooser.H</td>
<td>1255</td>
</tr>
<tr>
<td>fl_file_chooser_callback</td>
<td>Common Dialog Classes and Functions, 380</td>
</tr>
<tr>
<td>fl_file_chooser_callback</td>
<td>Common Dialog Classes and Functions, 381</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>552</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>add, 554</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>add_color, 554</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>add_vertex, 554, 555</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>draw, 555</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>find, 555</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>label, 556</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>labeltype, 556</td>
</tr>
<tr>
<td>FL_file_chooser</td>
<td>alloc_size, 566</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>COLUMN, 564</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>end, 566</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>fixed, 567</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>gap, 568</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>HORIZONTAL, 564</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>horizontal, 568</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>layout, 568</td>
</tr>
<tr>
<td>FL_File_Icon</td>
<td>margin, 568–570</td>
</tr>
</tbody>
</table>
on_remove, 570
resize, 570
ROW, 564
spacing, 571
VERTICAL, 564
Fl_Flex.H, 1260
Fl_Float_Input, 571
Fl_Float_Input.H, 1261
FL_FOCUS
Enumerations.H, 1188
fl_focus_rect
Drawing functions, 332
fl_font
Color & Font functions, 306
fl_fontnames
Events handling functions, 288
Fl_Fontsize
Enumerations.H, 1182
fl_fopen
Unicode and UTF-8 functions, 358
Fl_FormsBitmap, 572
draw, 573
set, 573
Fl_FormsBitmap.H, 1262
Fl_FormsPixmap, 573
draw, 574
Fi_FormsPixmap, 574
Pixmap, 575
set, 575
Fl_FormsPixmap.H, 1262
Fl_FormsText, 575
draw, 575
fl_frame
Drawing functions, 332
Enumerations.H, 1193
fl_frame2
Drawing functions, 333
Fl_Free, 576
draw, 577
Fl_Free.H, 1263
FL_FREE_BOXTYPE
Enumerations.H, 1185
FL_FREE_LABELTYPE
Enumerations.H, 1191
FL_FULLSCREEN
Enumerations.H, 1190
fl_gap
Drawing functions, 333
fl_getcwd
Unicode and UTF-8 functions, 359
fl_getenv
Unicode and UTF-8 functions, 359
Fl_GIF_image, 578
animate, 580
Fl_GIF_image.H, 1263
Fl_GIF_image::GIF_FRAME, 1163
Fl_GIF_image::GIF_FRAME::CPAL, 397
Fl_Gl_Choice, 581
Fl_Gl_Choice.H, 1445
Fl_Gl_Window, 581
as_gi_window, 584
can_do, 584
can_do_overlay, 584
can_do_valid, 584
draw, 585
draw_begin, 586
draw_end, 586
Fl_Gl_Window.H, 1264
Fl_Gl_Window.Driver.H, 1445
Fl_Gl_Window_Font, 590
Fl_Gl_Window_Font.H, 1264
Fl_Gl_Window_Strip, 591
Fl_Gl_Window_Vertex, 591
Fl_Gl_Window_H, 1445
fi_gray_ramp
Enumerations.H, 1194
Fl_Group, 593
~Fl_Group, 596
array, 597
as_group, 597
begin, 597
bounds, 598
child, 598
clear, 598
clip_children, 598, 599
current, 599
delete_child, 599
draw, 600
draw_child, 600
draw_children, 600
end, 600
find, 600
Fl_Group, 596
focus, 601
handle, 601
init_sizes, 601
insert, 602
on_insert, 602
on_move, 602
on_remove, 603
remove, 603, 604
resizable, 604
resize, 605
sizes, 606
update_child, 606
Fl_Group.H, 1271
fl_height
  Color & Font functions, 306
Fl_Help_Block, 607
Fl_Help_Dialog, 607
    load, 608
    show, 609
    textsize, 609
    value, 609
Fl_Help_Dialog.H, 1272
Fl_Help_Font_Stack, 609
Fl_Help_Font_Style, 610
Fl_Help_Font_Stack.H, 1272
Fl_Help_Font_Style.H, 1272
Fl_Help_Link, 610
Fl_Help_Target, 611
Fl_Help_View, 611
  ~Fl_Help_View, 614
draw, 615
find, 615
handle, 615
leftline, 615
link, 616
load, 616
resize, 616
scrollbar_size, 616, 617
topleft, 617
value, 618
Fl_Help_View.H, 1273
FL_HIDE
  Enumerations.H, 1190
Fl_Hold_Browser, 618
  ~Fl_Hold_Browser, 619
Fl_Hold_Browser.H, 1275
Fl_Hor_Fill_Slider, 619
Fl_Hor_Fill_Slider.H, 1277
Fl_Hor_Nice_Slider, 619
Fl_Hor_Nice_Slider.H, 1277
Fl_Hor_Slider, 620
Fl_Hor_Slider.H, 1278
Fl_Hor_Value_Slider, 621
Fl_Hor_Value_Slider.H, 1278
Fl ICO Image, 621
  ~Fl ICO Image, 622
Fl ICO Image.H, 1278
Fl ICO Image::IconDirEntry, 1163
Fl Image, 622
  as_shared_image, 625
color_average, 625
copy, 626
count, 627
d, 627
data, 627
desaturate, 628
draw, 628
draw_empty, 628
draw_scaled, 629
fail, 629
Fl Image, 625
  h, 629, 630
inactive, 630
label, 630
id, 630
release, 631
RGB scaling, 631
scale, 631
scaling_algorithm, 632
uncache, 632
w, 632
Fl Image.H, 1279, 1280
  Fl RGB Scaling, 1279
  FL RGB_SCALING_BILINEAR, 1280
  FL RGB_SCALING_NEAREST, 1280
FL_IMAGE_LABEL
  Enumerations.H, 1181
Fl Image Reader, 632
Fl Image Reader.h, 1447
Fl Image Surface, 633
  Fl Image Surface, 634
  highres_image, 635
  image, 635
  is_current, 635
  offscreen, 635
  origin, 635
  printable_rect, 636
  rescale, 636
  set_current, 636
  translate, 637
  untranslate, 637
Fl Image Surface.H, 1282
Fl Input, 637
draw, 639
Fl Input, 639
handle, 640
handle_key, 640
fl_input
  Common Dialog Classes and Functions, 381
Fl Input.H, 1282
Fl Input.H, 1281
~Fl Input, 640
append, 644
apply_undo, 644
can_redo, 645
INDEX

can_undo, 645

copy, 645
copy_cuts, 645
cursor_color, 646
cut, 646, 647
drawtext, 647
Fl_Input_, 644
handle_mouse, 648
handletext, 648
index, 648
input_type, 648
insert, 649
insert_position, 649, 650
line_end, 650
line_start, 650
mark, 651
maximum_size, 651
position, 651, 652
readonly, 652
redo, 652
replace, 653
resize, 653
shortcut, 654
size, 654
static_value, 655
tab_nav, 655, 656
textcolor, 656
textfont, 656, 657
textsize, 657
undo, 657
up_down_position, 657
value, 658
word_end, 659
word_start, 659
wrap, 659, 660
Fl_Input_.H, 1283
Fl_Input_Choice, 660
add, 663
Fl_Input_Choice, 663
inp_x, 663
input, 663
menu_x, 663
menubutton, 664
resize, 664
update_menubutton, 664
value, 664, 665
Fl_Input_Choice.H, 1286
fl_input_str

Common Dialog Classes and Functions, 381, 382

Fl_Int_Input, 665
Fl_Int_Input., 666
Fl_Int_Input.H, 1288
Fl_Int_Vector, 666
back, 667
empty, 667
operator=, 667
operator[], 667, 668
pop_back, 668

size, 668
Fl_Int_Vector.H, 1288, 1289
fl_internal_boxtype
   fl_boxtype.cxx, 1439
fi_inptr_t
   platform_types.h, 1424
FL_JOIN_BEVEL
   Drawing functions, 319
FL_JOIN_MITER
   Drawing functions, 319
FL_JOIN_ROUND
   Drawing functions, 319
Fl_JPEG_Image, 668
   Fl_JPEG_Image, 669
Fl_JPEG_Image.H, 1290
Fl_Kdialog_Native_File_Chooser_Driver, 671
FL_KEYBOARD
   Enumerations.H, 1189
FL_KEYDOWN
   Enumerations.H, 1189
FL_KEYUP
   Enumerations.H, 1189
Fl_Label, 671
draw, 672
measure, 672
type, 673
FL_Labeltype
   Enumerations.H, 1190
fl_latin1_to_local
   Color & Font functions, 306
FL_LEAVE
   Enumerations.H, 1188
Fl_Light_Button, 673
draw, 674
Fl_Light_Button, 674
handle, 674
Fl_Light_Button.H, 1290
fl_lightness
   Color & Font functions, 307
Fl_Line_Dial, 675
Fl_Line_Dial.H, 1291
fl_line_style
   Drawing functions, 333
fl_load_matrix
   Drawing functions, 334
fl_local_to_latin1
   Color & Font functions, 307
fl_local_to_mac_roman
   Color & Font functions, 307
fl_luminance
   Color & Font functions, 309
Fl_Mac_App_Menu, 675
custom_application_menu_items, 676
print, 676
fl_mac_os_version
   Mac OS X-specific symbols, 372
fl_mac_roman_to_local
   Color & Font functions, 309
**FL_ORIENTATION**
- Enumerations.H, 1191

**FL_ORIENTATION_NONE**
- Enumerations.H, 1191

**FL_ORIENTATION_NW**
- Enumerations.H, 1191

**FL_ORIENTATION_RIGHT**
- Enumerations.H, 1191

**FL_ORIENTATION_SE**
- Enumerations.H, 1192

**FL_ORIENTATION_SW**
- Enumerations.H, 1191

**FL_ORIENTATION_UP**
- Enumerations.H, 1191

**FL_ORIENTATION**
- Enumerations.H, 1191

**Fl_Output**
- 721
  - Fl_Output, 722
  - Fl_Output.H, 1303

**fl_overlay_clear**
- Drawing functions, 337

**fl_overlay_rect**
- Drawing functions, 337

**Fl_Overlay_Window**
- 722
  - as_overlay_window, 724
draw_overlay, 724
  - Fl_Overlay_Window.H, 1303
  flush, 724
hide, 724
redraw_overlay, 724
resize, 724
show, 725

**Fl_Overlay_Window.H**
- 1303

**fl_overlay_scale**
- Drawing functions, 338

**fl_oxy.h**
- 1451

**Fl_Pack**
- 725
draw, 727
  - Fl_Pack.H, 1304
horizontal, 727
resize, 727

**Fl_Pack.H**
- 1304

**Fl_Paged_Device**
- 728
  - A0, 729
A1, 730
A2, 730
A3, 730
A4, 730
A5, 730
A6, 730
A7, 730
A8, 730
A9, 730
B0, 730
B1, 730
B10, 730
B2, 730
B3, 730
B4, 730
B5, 730

**B6, 730**
**B7, 730**
**B8, 730**
**B9, 730**
begin_job, 730
begin_page, 731
end_job, 731
end_page, 731
EXECUTIVE, 730
FOLIO, 730
LANDSCAPE, 730
LEDGER, 730
LEGAL, 730
LETTER, 730
margins, 732
ORIENTATION, 730
Page_Format, 729
Page_Layout, 730
PORTRAIT, 730
REVERSED, 730
rotate, 732
scale, 732
start_job, 733
start_page, 733
TABLOID, 730

**Fl_Paged_Device.cxx**
- 1451

**Fl_Paged_Device.H**
- 1304, 1305

**Fl_Paged_Device::page_format**
- 1166

**fl_password**
- Common Dialog Classes and Functions, 386

**fl_password_str**
- Common Dialog Classes and Functions, 387

**FL_PASTE**
- Enumerations.H, 1190

**FL_PATCH_VERSION**
- Enumerations.H, 1181

**fl_pie**
- Drawing functions, 338

**Fl_Pixmap**
- 733
color_average, 735
copy, 735
desaturate, 735
draw, 736
Fl_Pixmap, 734
label, 736
uncache, 736

**Fl_Pixmap.H**
- 1306

**Fl_Png_Image**
- 739

**Fl_Png_Image.H**
- 1307

**Fl_Plugin**
- 737
Fl_Plugin, 737
Fl_Plugin.H, 1307

**Fl_Plugin_Manager**
- 738
~Fl_Plugin_Manager, 738
addPlugin, 739
load, 739
removePlugin, 739

**Fl_Png_Image**
- 739
Fl_Png_Image, 740
Fl_Png_Image.H, 1307
FL_REASON_CANCELLED
Enumerations.H, 1186

FL_REASON_CHANGED
Enumerations.H, 1186

FL_REASON_CLOSED
Enumerations.H, 1186

FL_REASON_DESELECTED
Enumerations.H, 1186

FL_REASON_DRAGGED
Enumerations.H, 1186

FL_REASON_GOT_FOCUS
Enumerations.H, 1186

FL_REASON_LOST_FOCUS
Enumerations.H, 1186

FL_REASON_OPENED
Enumerations.H, 1186

FL_REASON_RELEASED
Enumerations.H, 1186

FL_REASON_RESELECTED
Enumerations.H, 1186

FL_REASONSelected
Enumerations.H, 1186

FL_REASON_UNKNOWN
Enumerations.H, 1186

Fl_Rect, 783
  b, 784
  Fl_Rect, 784
  inset, 784, 785
  r, 785

Fl_rect
  Drawing functions, 340
  fl_rect.cxx, 1451

Fl_Rect.H, 1317

fl_rectbound
  fl_boxtype.cxx, 1439

Fl_rectf
  Drawing functions, 340, 341

Fl_Region
  platform_types.h, 1424

fl_register_images
  Fl_Shared_Image.H, 1326

FL_RELEASE
  Enumerations.H, 1188

fl_rename
  Unicode and UTF-8 functions, 362

Fl_Repeat_Button, 786
  Fl_Repeat_Button, 786
  handle, 787

Fl_Repeat_Button.H, 1318

fl_rescale_offscreen
  Drawing functions, 341

FL_RESERVED_TYPE
  Fl_Widget.H, 1386

fl_reset_spot
  Drawing functions, 341

fl_restore_scale
  Drawing functions, 341

Fl_Return_Button, 787
  draw, 788
  Fl_Return_Button, 788
  handle, 788

Fl_Return_Button.H, 1318

Fl_RGB_Image, 789
  array, 794
  as_svg_image, 792
  color_average, 792
  copy, 792
  desaturate, 793
  draw, 793
  Fl_RGB_Image, 790–792
  label, 793, 794
  max_size, 794
  normalize, 794
  uncache, 794

Fl_RGB_Image.H, 1319

Fl_RGB_Scaling
  Fl_Image.H, 1279

FL_RGB_SCALING_BILINEAR
  Fl_Image.H, 1280

FL_RGB_SCALING_NEAREST
  Fl_Image.H, 1280

fl_rmdir
  Unicode and UTF-8 functions, 362

Fl_Roller, 795
  draw, 796
  Fl_Roller, 796
  handle, 796

Fl_Roller.H, 1319

fl_rotate
  Drawing functions, 342

Fl_Round_Button, 797
  Fl_Round_Button, 797

Fl_Round_Button.H, 1320

Fl_Round_Clock, 798
  Fl_Round_Clock, 798

Fl_Round_Clock.H, 1320

fl_rounded_rect
  Drawing functions, 342

fl_rounded_rectf
  Drawing functions, 342

fl_scale
  Drawing functions, 342

Fl_Scheme, 799
  add_scheme_name, 799
  names, 800
  num_schemes, 800

Fl_Scheme.H, 1320

Fl_Scheme_Choice, 801
  Fl_Scheme_Choice, 801
  handle, 802
  init_value, 802
  scheme_cb_, 802

Fl_Scheme_Choice.H, 1321

FL_SCREEN_CONFIGURATION_CHANGED
  Enumerations.H, 1190
INDEX

Fl_Screen_Driver.H, 1451
Fl_Scroll, 802
~Fl_Scroll, 805
bbox, 805
delete_child, 806
draw, 806
fix_scrollbar_order, 806
Fl_Scroll, 805
handle, 807
on_insert, 807
on_move, 808
recalc_scrollbars, 808
resize, 808
scroll_to, 809
scrollbar_size, 809
xposition, 810
yposition, 810
Fl_Scroll
Drawing functions, 343
Fl_Scroll.H, 1322
Fl_Scroll::Fl_Region_LRTB, 785
Fl_Scroll::Fl_Region_XYWH, 786
Fl_Scroll::Fl_Scrollbar_Data, 813
Fl_Scroll::ScrollInfo, 1166
Fl_Scrollbar, 810
draw, 812
Fl_Scrollbar, 811
handle, 812
linesize, 812
value, 812, 813
Fl_Scrollbar.H, 1323
Fl_Secret_Input, 814
Fl_Secret_Input, 814
handle, 815
Fl_Secret_Input.H, 1324
Fl_Select_Browser, 815
Fl_Select_Browser, 816
FL_SELECTIONCLEAR
Enumerations.H, 1190
fi_set_spot
Drawing functions, 343
fi_set_status
Drawing functions, 343
Fl_Shared_Handler
Fl_Shared_Image.H, 1325
Fl_Shared_Image, 816
~Fl_Shared_Image, 819
add, 819
add_handler, 819
as_shared_image, 819
color_average, 820
compare, 820
copy, 820
desaturate, 821
draw, 821
find, 822
Fl_Shared_Image, 819
get, 822
image, 823
original, 823
refcount, 823
release, 823
uncache, 824
Fl_Shared_Image.H, 1325, 1326
fi_register_images, 1326
Fl_Shared_Handler, 1325
FL_SHORTCUT
Enumerations.H, 1189
Fl_Shortcut
fl_types.h, 1377
Fl_Shortcut_Button, 824
default_value, 825
draw, 826
Fl_Shortcut_Button, 825
handle, 826
value, 826
Fl_Shortcut_Button.H, 1327
fl_shortcut_label
Drawing functions, 344
FL_SHOW
Enumerations.H, 1190
fl_show_colormap
Color & Font functions, 310
fl_show_colormap.H, 1328
fl_show_input.H, 1328
Fl_Simple_Counter, 826
Fl_Simple_Counter.H, 1329
Fl_Simple_Terminal, 827
ansi, 830, 831
ansi_show_unknown, 831, 832
append, 832
backspace_buffer, 832
clear, 832
current_style, 832
current_style_index, 833
draw, 833
enforce_history_lines, 833
enforce_stay_at_bottom, 833
handle_backspace, 834
history_lines, 834
normal_style_index, 834
printf, 835
remove_lines, 835
stay_at_bottom, 835, 836
style_table, 836
style_table_size, 837
text, 837
vprintf, 838
Fl_Simple_Terminal.H, 1329
Fl_Single_Window, 838
show, 839
Fl_Single_Window.H, 1331
fl_size
Color & Font functions, 310
Fl_Slider, 839

Generated by Doxygen
printf, 921
remove, 922
remove_, 922
replace, 922
rewind_lines, 922
savefile, 923
search_backward, 923
search_forward, 924
secondary_selection_text, 924
selection_text, 924
skip_displayed_characters, 924
tab_distance, 924
text, 924, 925
text_range, 925
transcoding_warning_action, 926
undo, 925
vprintf, 925
word_end, 926
word_start, 926
Fl_Text_Buffer.H, 1348
Fl_Text_Display, 927
∼Fl_Text_Display, 937
absolute_top_line_number, 937
ATTR_BGCOLOR, 936
ATTR_BGCOLOR_EXT, 936
ATTR_BGCOLOR_EXT_, 936
ATTR_GRAMMAR, 936
ATTR_LINES_MASK, 936
ATTR_SPELLING, 936
ATTR_STRIKE_THROUGH, 936
BLOCK_CURSOR, 936
buffer, 937, 938
buffer_modified_cb, 938
buffer_predelete_cb, 938
calc_last_char, 939
calc_line_starts, 939
CARET_CURSOR, 936
clear_rect, 939
col_to_x, 939
count_lines, 940
cursor_color, 940
cursor_style, 940
DIM_CURSOR, 936
display_insert, 941
draw, 941
draw_cursor, 941
draw_line_numbers, 941
draw_range, 942
draw_string, 942
draw_text, 942
draw_vline, 943
empty_vlines, 943
extend_range_for_styles, 943
find_line_end, 943
find_wrap_range, 944
find_x, 944
Fl_Text_Display, 936
generate_absolute_top_line_number, 945
grammar_underline_color, 945
handle, 945
handle_vline, 945
HEAVY_CURSOR, 936
highlight_data, 946
in_selection, 947
insert, 947
insert_position, 947, 948
line_end, 948
line_start, 948
linenumber_align, 949
linenumber_bgc, 949
linenumber_fgcolor, 949
linenumber_font, 949
linenumber_format, 949
linenumber_size, 950
linenumber_width, 950
longest_vline, 950
maintain_absolute_top_line_number, 950
maintaining_absolute_top_line_number, 951
measure_deleted_lines, 951
measure_proportional_character, 951
move_down, 952
move_left, 952
move_right, 952
move_up, 952
NORMAL_CURSOR, 936
offset_line_starts, 952
overstrike, 953
position_style, 953
position_to_line, 953
position_to_linecol, 954
position_to_xy, 954
redisplay_range, 955
reset_absolute_top_line_number, 955
resize, 955
rewind_lines, 955
scroll, 956
scroll_, 956
scroll_timer_cb, 956
scrollbar_align, 957
scrollbar_size, 957
scrollbar_width, 958
secondary_selection_color, 958
shortcut, 958
show_cursor, 959
show_insert_position, 959
SIMPLE_CURSOR, 936
skip_lines, 959
spelling_underline_color, 959, 960
string_width, 960
style_buffer, 960
textcolor, 960, 961
textfont, 961
textsize, 961
update_h_scrollbar, 962
INDEX

update_line_starts, 962
update_v_scrollbar, 962
vline_length, 962
word_end, 963
word_start, 963
WRAP_AT_BOUNDS, 936
WRAP_AT_COLUMN, 936
WRAP_AT_PIXEL, 936
wrap_mode, 963
WRAP_NONE, 936
wrap_uses_character, 964
wrapped_column, 964
wrapped_line_counter, 965
wrapped_row, 965
x_to_col, 966
xy_to_position, 966
xy_to_rowcol, 966

Fl_Text_Display.H, 1352
Fl_Text_Display::Style_Table_Entry, 1167
Fl_Text_Editor, 967
add_key_binding, 969
global_key_bindings, 975
handle, 969
insert_mode, 970
ekf_backspace, 970
kf_c_s_move, 970
kf_copy, 970
kf_ctrl_move, 970
kf_cut, 971
kf_default, 971
kf_delete, 971
kf_down, 971
kf_end, 971
kf_enter, 971
kf_home, 971
kf_ignore, 971
kf_insert, 972
kf_left, 972
kf_m_s_move, 972
kf_meta_move, 972
kf_move, 972
kf_page_down, 973
kf_page_up, 973
kf_paste, 973
kf_redo, 973
kf_right, 973
kf_select_all, 973
kf_shift_move, 973
kf_undo, 974
kf_up, 974
remove_key_binding, 974
tab_nav, 974
Fl_Text_Editor.H, 1357
Fl_Text_Editor::Key_Binding, 1164
fl_text_extents
   Color & Font functions, 310, 311
Fl_Text_Selection, 975
derect, 977
length, 977
position, 977
selected, 977, 978
set, 978
start, 978
update, 979
FL_THIN_DOWN_BOX
   Enumerations.H, 1184
FL_THIN_DOWN_FRAME
   Enumerations.H, 1184
FL_THIN_UP_BOX
   Enumerations.H, 1184
FL_THIN_UP_FRAME
   Enumerations.H, 1184
Fl_Tile, 979
Fl_Tile, 981
handle, 981
move_intersection, 982
position, 982
resize, 982
Fl_Tile.H, 1358
Fl_Tiled_Image, 982
color_average, 984
copy, 984
desaturate, 985
draw, 985
Fl_Tiled_Image, 983
Fl_Tiled_Image.H, 1359
Fl_Timeout, 985
add_timeout, 987
current, 987
current_timeout, 992
elapse_timeouts, 987
first_timeout, 992
get, 988
has_timeout, 988
insert, 989
make_current, 989
release, 989
remove_timeout, 989
repeat_timeout, 991
time_to_wait, 991
Fl_Timeout.cxx, 1459
Fl_Timeout.h, 1459, 1460
Fl_Timeout_Handler
   Callback Function Typedefs, 272
Fl_Timer, 992
direction, 994
draw, 994
Fl_Timer, 993
handle, 994
suspended, 994
Fl_Timer.H, 1359
Fl_Timestamp
   platform_types.h, 1424
Fl_Toggle_Button, 995

Generated by Doxygen
Fl_Toggle_Button, 995
Fl_Toggle_Button.H, 1360
Fl_Toggle_Light_Button.H, 1360
Fl_Toggle_Round_Button.H, 1361
Fl_Tooltip, 996
color, 997
current, 998
delay, 998
disable, 998
enable, 998
enabled, 998
enter_area, 998
font, 999
hidedelay, 999
hoverdelay, 999
margin_height, 999
margin_width, 1000
size, 1000
textcolor, 1000
wrap_width, 1000
Fl_Tooltip.H, 1361
fl_transform_dx
Drawing functions, 345
fl_transform_dy
Drawing functions, 345
fl_transform_x
Drawing functions, 345
fl_transform_y
Drawing functions, 345
fl_transformed_vertex
Drawing functions, 346
fl_translate
Drawing functions, 346
Fl_Tree, 1001
add, 1011, 1012
calc_dimensions, 1012
calc_tree, 1013
callback_item, 1013
callback_reason, 1013, 1014
clear, 1014
clear_children, 1014
close, 1014, 1015
closeicon, 1015
correctorstyle, 1015
deselect, 1016
deselect_all, 1017
display, 1017
displayed, 1017
draw, 1018
extend_selection, 1018
extend_selection_dir, 1018
find_clicked, 1019
find_item, 1019
first, 1020
first_selected_item, 1020
first_visible, 1020
first_visible_item, 1021
get_selected_items, 1021
handle, 1021
hposition, 1022
insert, 1022
insert_above, 1023
is_close, 1023, 1024
is_hscroll_visible, 1024
is_open, 1024, 1025
is_scrollbar, 1025
is_selected, 1025, 1026
is_vscroll_visible, 1026
item_clicked, 1026
item_draw_mode, 1027
item_labelbgcolor, 1027
item_labelfgcolor, 1027
item_labelfont, 1028
item_labelsize, 1028
item_pathname, 1028
item_reselect_mode, 1028
last, 1029
last_selected_item, 1029
last_visible, 1029
last_visible_item, 1029
load, 1030
next, 1030
next_item, 1030
next_selected_item, 1031
next_visible_item, 1032
open, 1033
open_toggle, 1034
openicon, 1034
prev, 1034
recalc_tree, 1035
remove, 1035
resize, 1035
root, 1036
root_label, 1036
scrollbar_size, 1036
select, 1037
select_all, 1038
select_only, 1038
select_toggle, 1039
selectbox, 1039
selectmode, 1039
set_item_focus, 1040
show_item, 1040
show_item_bottom, 1040
show_item_middle, 1041
show_item_top, 1041
show_self, 1041
showcollapse, 1041
showroot, 1042
sortorder, 1042
usericon, 1042
vposition, 1042, 1043
Fl_Tree.H, 1362, 1363
Fl_Tree_Reason, 1363
FL_TREE_REASON_CLOSED, 1363
FL_TREE_REASON_DESELECTED, 1363
Generated by Doxygen
INDEX

FL_TREE_REASON_DRAGGED, 1363
FL_TREE_REASON_NONE, 1363
FL_TREE_REASON_OPENED, 1363
FL_TREE_REASON_RESELECTED, 1363
FL_TREE_REASON_SELECTED, 1363

Fl_Tree_Connector
Fl_Tree_Prefs.H, 1372
FL_TREE_CONNECTOR_DOTTED
Fl_Tree_Prefs.H, 1372
FL_TREE_CONNECTOR_NONE
Fl_Tree_Prefs.H, 1372
FL_TREE_CONNECTOR_SOLID
Fl_Tree_Prefs.H, 1372

Fl_Tree_Item, 1043
activate, 1049
add, 1049, 1050
calc_item_height, 1050
child, 1050
deactivate, 1050
deparent, 1051
depath, 1051
deselect_all, 1051
draw, 1051
draw_horizontal_connector, 1052
draw_item_content, 1052
draw_vertical_connector, 1053
drawbgcolor, 1053
drawfgcolor, 1053
find_child, 1053, 1054
find_child_item, 1054
find_clicked, 1054
find_item, 1055
Fl_Tree_Item, 1048
hide_widgets, 1055
insert, 1055
insert_above, 1055
label, 1056
label_h, 1056
label_w, 1056
label_x, 1056
label_y, 1056
labelbgcolor, 1056, 1057
move, 1057
move_above, 1058
move_below, 1058
move_into, 1058
next, 1058
next_displayed, 1059
next_sibling, 1059
next_visible, 1059
parent, 1059
prefs, 1059
prev, 1060
prev_displayed, 1060
prev_sibling, 1060
prev_visible, 1060
recalc_tree, 1060
remove_child, 1061
reparent, 1061
replace, 1061
replace_child, 1062
select, 1062
select_all, 1062
show_self, 1063
show_widgets, 1063
swap_children, 1063
tree, 1063, 1064
update_prev_next, 1064
userdeicon, 1064
usericon, 1064
visible_r, 1065

Fl_Tree_Item.H, 1366
Fl_Tree_Item_Array, 1065
add, 1066
clear, 1066
deparent, 1066
Fl_Tree_Item_Array, 1066
insert, 1067
manage_item_destroy, 1067
move, 1067
remove, 1067
reparent, 1068
replace, 1068

Fl_Tree_Item_Array.H, 1370
FL_TREE_ITEM_DRAW_DEFAULT
Fl_Tree_Prefs.H, 1372
FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET
Fl_Tree_Prefs.H, 1372
Fl_Tree_Item_Draw_Mode
Fl_Tree_Prefs.H, 1372
FL_TREE_ITEM_HEIGHT_FROM_WIDGET
Fl_Tree_Prefs.H, 1372
Fl_Tree_Item_Reselect_Mode
Fl_Tree_Prefs.H, 1372
Fl_Tree_Prefs, 1068
closedeicon, 1071
closeicon, 1071
item_draw_mode, 1071
item_labelbgcolor, 1071, 1072
marginbottom, 1072
opendeicon, 1072
openicon, 1072
selectmode, 1072
showcollapse, 1072
showroot, 1073
sortorder, 1073
userdeicon, 1073

Fl_Tree_Prefs.H, 1371, 1373
Fl_Tree_Connector, 1372
FL_TREE_CONNECTOR_DOTTED, 1372
FL_TREE_CONNECTOR_NONE, 1372
FL_TREE_CONNECTOR_SOLID, 1372
FL_TREE_ITEM_DRAW_DEFAULT, 1372
FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET, 1372
Fl_Tree_Item_Draw_Mode, 1372
image_bound, 1116
INACTIVE, 1097
inside, 1116
INVISIBLE, 1097
is_label_copied, 1116
label, 1116, 1117
label_shortcut, 1117
labelcolor, 1118
labelfont, 1118
labelsize, 1119
labelfont, 1119
MAC_USE_ACCENTS_MENU, 1097
measure_label, 1120
MENU_WINDOW, 1097
MODAL, 1097
NEEDS_KEYBOARD, 1097
needs_keyboard, 1120
NO_OVERLAY, 1097
NOBORDER, 1097
NON_MODAL, 1097
OUTPUT, 1097
output, 1120
OVERRIDE, 1097
parent, 1121
position, 1121
redraw, 1121
redraw_label, 1121
resize, 1122
selection_color, 1122
set_active, 1123
set_changed, 1123
set_output, 1123
set_visible, 1123
set_visible_focus, 1123
SHORTCUT_LABEL, 1097
shortcut_label, 1123, 1124
show, 1124
size, 1124
take_focus, 1125
takeevents, 1125
test_shortcut, 1125
tooltip, 1126
TOOLTIP_WINDOW, 1097
top_window, 1126
top_window_offset, 1127
type, 1127
user_data, 1127
USERFLAG1, 1097
USERFLAG2, 1097
USERFLAG3, 1097
visible, 1129
VISIBLE_FOCUS, 1097
visible_focus, 1129
visible_r, 1129
w, 1130
when, 1130
window, 1131
x, 1131
y, 1131, 1132
FL_Widget.H, 1386
FL_WIDGET_SURFACE, 1097
draw, 1133
draw_decorated_window, 1134
FL_Widget_Surface, 1133
origin, 1134
print_window_part, 1134
printable_rect, 1135
translate, 1135
untranslate, 1135
FL_Widget_Surface.H, 1391
FL_Widget_Tracker, 1135
deleted, 1136
exists, 1136
widget, 1136
fl_width
Color & Font functions, 311
FL_Window, 1137
~FL_Window, 1142
as_double_window, 1142
as_overlay_window, 1142
as_window, 1142
border, 1143
clear_border, 1143
clear_modal_states, 1143
current, 1143
current_, 1159
cursor, 1144
decorated_h, 1144
decorated_w, 1145
default_cursor, 1145
default_icon, 1145
default_icons, 1146
default_size_range, 1147
default_xclass, 1147, 1148
draw, 1148
FL_Window, 1141
flush, 1148
force_position, 1148, 1149
free_icons, 1149
free_position, 1149
fullscreen, 1149
fullscreen_screens, 1149
handle, 1150
hide, 1150
hotspot, 1150
icon, 1151
iconize, 1151
icons, 1152
is_resizable, 1152
make_current, 1153
modal, 1153
os_id, 1153
resize, 1153
screen_num, 1154
set_menu_window, 1154
set_visible_focus, 1123
set_visible, 1123
set_visible_focus, 1123
SHORTCUT_LABEL, 1097
shortcut_label, 1123, 1124
show, 1124
size, 1124
take_focus, 1125
takeevents, 1125
test_shortcut, 1125
tooltip, 1126
TOOLTIP_WINDOW, 1097
top_window, 1126
top_window_offset, 1127
type, 1127
user_data, 1127
USERFLAG1, 1097
USERFLAG2, 1097
USERFLAG3, 1097
visible, 1129
VISIBLE_FOCUS, 1097
visible_focus, 1129
visible_r, 1129
w, 1130
when, 1130
window, 1131
x, 1131
y, 1131, 1132
FL_Widget.H, 1386
FL_RESERVED_TYPE, 1386
FL_Widget_Surface, 1132
draw, 1133
draw_decorated_window, 1134
FL_Widget_Surface, 1133
origin, 1134
print_window_part, 1134
printable_rect, 1135
translate, 1135
untranslate, 1135
FL_Widget_Surface.H, 1391
FL_Widget_Tracker, 1135
deleted, 1136
exists, 1136
widget, 1136
fl_width
Color & Font functions, 311
FL_Window, 1137
~FL_Window, 1142
as_double_window, 1142
as_overlay_window, 1142
as_window, 1142
border, 1143
clear_border, 1143
clear_modal_states, 1143
current, 1143
current_, 1159
cursor, 1144
decorated_h, 1144
decorated_w, 1145
default_cursor, 1145
default_icon, 1145
default_icons, 1146
default_size_range, 1147
default_xclass, 1147, 1148
draw, 1148
FL_Window, 1141
flush, 1148
force_position, 1148, 1149
free_icons, 1149
free_position, 1149
fullscreen, 1149
fullscreen_screens, 1149
handle, 1150
hide, 1150
hotspot, 1150
icon, 1151
iconize, 1151
icons, 1152
is_resizable, 1152
make_current, 1153
modal, 1153
os_id, 1153
resize, 1153
screen_num, 1154
set_menu_window, 1154
set_visible_focus, 1123
set_visible, 1123
set_visible_focus, 1123
SHORTCUT_LABEL, 1097
shortcut_label, 1123, 1124
show, 1124
size, 1124
take_focus, 1125
takeevents, 1125
test_shortcut, 1125
tooltip, 1126
TOOLTIP_WINDOW, 1097
top_window, 1126
top_window_offset, 1127
type, 1127
user_data, 1127
USERFLAG1, 1097
USERFLAG2, 1097
USERFLAG3, 1097
visible, 1129
VISIBLE_FOCUS, 1097
visible_focus, 1129
visible_r, 1129
w, 1130
when, 1130
window, 1131
x, 1131
y, 1131, 1132
FL_Widget.H, 1386
FL_RESERVED_TYPE, 1386
FL_Widget_Surface, 1132
draw, 1133
draw_decorated_window, 1134
FL_Widget_Surface, 1133
origin, 1134
print_window_part, 1134
printable_rect, 1135
translate, 1135
untranslate, 1135
FL_Widget_Surface.H, 1391
FL_Widget_Tracker, 1135
deleted, 1136
exists, 1136
widget, 1136
fl_width
Color & Font functions, 311
FL_Window, 1137
~FL_Window, 1142
as_double_window, 1142
as_overlay_window, 1142
as_window, 1142
border, 1143
clear_border, 1143
clear_modal_states, 1143
current, 1143
current_, 1159
cursor, 1144
decorated_h, 1144
decorated_w, 1145
default_cursor, 1145
default_icon, 1145
default_icons, 1146
default_size_range, 1147
default_xclass, 1147, 1148
draw, 1148
FL_Window, 1141
flush, 1148
force_position, 1148, 1149
free_icons, 1149
free_position, 1149
fullscreen, 1149
fullscreen_screens, 1149
handle, 1150
hide, 1150
hotspot, 1150
icon, 1151
iconize, 1151
icons, 1152
is_resizable, 1152
make_current, 1153
modal, 1153
os_id, 1153
resize, 1153
screen_num, 1154
set_menu_window, 1154
INDEX

set_modal, 1154
set_non_modal, 1154
set_tooltip_window, 1154
shape, 1155
show, 1155, 1156
shown, 1156
size_range, 1156
wait_for_expose, 1157
xclass, 1158

Fl_Window.H, 1391, 1392
Fl_Window_Driver.H, 1461
Fl_Wizard, 1159
draw, 1160
Fl_Wizard, 1160
next, 1160
Fl_Wizard.H, 1394
fl_wl_compositor
wayland.H, 1427

FL_WRITE
Enumerations.H, 1183
fl_write_png
fl_write_png.cxx, 1464
fl_write_png.cxx, 1463
fl_write_png, 1464
fl_x11_find
x11.H, 1430
fl_x11_gc
x11.H, 1430
fl_x11_xid
x11.H, 1430

Fl_XBM_Image, 1160
Fl_XBM_Image, 1161
Fl_XBM_Image.H, 1395
Fl_XColor, 1161
Fl_XColor.H, 1465
Fl_XPM_Image, 1160
Fl_XPM_Image, 1162
Fl_XPM_Image.H, 1395
Fl_Zenity_Native_File_Chooser_Driver, 1162

FL_ZOOM_EVENT
Enumerations.H, 1190

FL_ZOOM_GESTURE
Enumerations.H, 1190

Flags
Fl_Anim_GIF_Image, 432

fлистинг.h, 1466
flush
Fl, 415
Fl_Double_Window, 538
Fl_Gl_Window, 586
Fl_Overlay_Window, 724
Fl_Preferences, 762
Fl_Window, 1148
focus
Events handling functions, 285
Fl_Group, 601

FOLIO
Fl_Paged_Device, 730

font
Fl_Tooltip, 999
FORCE_POSITION
Fl_Widget, 1097

force_position
Fl_Window, 1148, 1149
format
Fl_Window, 1148, 1149
format_char
Fl_Browser, 457, 458
forms.H, 1396
frame
Fl_Anim_GIF_Image, 436
frame_count
Fl_Anim_GIF_Image, 436
frame_h
Fl_Anim_GIF_Image, 437
frame_uncache
Fl_Anim_GIF_Image, 437
frame_w
Fl_Anim_GIF_Image, 437
frame_x
Fl_Anim_GIF_Image, 438
frame_y
Fl_Anim_GIF_Image, 438
frames
Fl_Anim_GIF_Image, 438
free_color

Color & Font functions, 311
free_icons
Fl_Window, 1149

free_position
Fl_Window, 1149
free_timeout
Fl_Timeout, 992
freeglut_teapot_data.h, 1467

full_height
Fl_Browser, 458
Fl_Browser_, 478
full_width
Fl_Browser_, 478
FULLSCREEN
Fl_Widget, 1097

fullscreen
Fl_Window, 1149
fullscreen_screens
Fl_Window, 1149

g
Fl_Color_Chooser, 523
gap
Fl_Flex, 568
gb2312.h, 1666
georgian_academy.h, 1696
georgian_ps.h, 1697
get
Fl_Preferences, 762–766
Fl_Shared_Image, 822
Fl_Timeout, 988
get_absolute_top_line_number
   Fl_Text_Display, 945
get_color
   Color & Font functions, 311, 312
get_font
   Color & Font functions, 312
get_font_name
   Color & Font functions, 312
get_font_sizes
   Color & Font functions, 312
get_key
   Events handling functions, 285
get_mouse
   Events handling functions, 286
get_selected_items
   Fl_Tree, 1021
get_selection
   Fl_Table, 891
get_system_colors
   Fl, 415
get_userdata_path
   Fl_Preferences, 766
gl.h, 1405, 1409
gl_color, 1407
gl_draw, 1407, 1408
gl_font, 1408
gl_rect, 1409
gl_rectf, 1409
gl_texture_pile_height, 1409
gl2opengl.h, 1410
gl.color, 1407
gl.draw
   gl.h, 1407, 1408
gl_draw.H, 1411
gl_font
   gl.h, 1408
gl_rect
   gl.h, 1409
gl_rectf
   gl.h, 1409
gl_texture_pile_height, 1409
gl_visual
   Fl, 415
GLContext
   platform_types.h, 1424
global
   Fl_Menu_, 686
global_key_bindings
   Fl_Text_Editor, 975
glu.h, 1411
glut.H, 1412
grab
   Windows handling functions, 273
grammar_underline_color
   Fl_Text_Display, 945
group
   Fl_Preferences, 767
   Fl_Text_Display, 945
group_exists
   Fl_Preferences, 767
GROUP_RELATIVE
   Fl_Widget, 1097
groups
   Fl_Preferences, 768
h
   Fl_Image, 629, 630
   Fl_Widget, 1114
handle
   Events handling functions, 286
   Fl_Adjuster, 428
   Fl_Box, 449
   Fl_Browser_, 478
   Fl_Button, 492
   Fl_Check_Browser, 505
   Fl_Choice, 512
   Fl_Clock, 516
   Fl_Color_Choser, 523
   Fl_Counter, 531
   Fl_Dial, 536
   Fl_File_Input, 560
   Fl_Free, 577
   Fl_Gl_Window, 586
   Fl_Glut_Window, 593
   Fl_Group, 601
   Fl_Help_View, 615
   Fl_Input, 640
   Fl_Light_Button, 674
   Fl_Menu_Bar, 693
   Fl_Menu_Button, 696
   Fl_Positioner, 743
   Fl_Repeat_Button, 787
   Fl_Return_Button, 788
   Fl_Roller, 796
   Fl_Scheme_Choice, 802
   Fl_Scroll, 807
   Fl_Scrollbar, 812
   Fl_Secret_Input, 815
   Fl_Shortcut_Button, 826
   Fl_Slider, 841
   Fl_Spinner, 845
   Fl_Spinner::Fl_Spinner_Input, 848
   Fl_Table, 891
   Fl_Table_Row, 899
   Fl_Tabs, 905
   Fl_Text_Display, 945
   Fl_Text_Editor, 969
   Fl_Tile, 981
   Fl_Timer, 994
   Fl_Tree, 1021
   Fl_Value_Input, 1080
   Fl_Value_Output, 1084
   Fl_Value_Slider, 1087
   Fl_Widget, 1114
   Fl_Window, 1150
handle_
Events handling functions, 286
handle_backspace
  Fl_Simple_Terminal, 834
handle_key
  Fl_Input, 640
handle_mouse
  Fl_Input, 648
handle_overflow
  Fl_Tabs, 906
handle_vline
  Fl_Text_Display, 945
handletext
  Fl_Input, 648
has_scrollbar
  Fl_Browser, 479
has_timeout
  Fl, 415
  Fl_Timeout, 988
HEAVY_CURSOR
  Fl_Text_Display, 936
help
  Fl, 426
hexdump
  Fl_String, 854
hide
  Fl_Browser, 458, 459
  Fl_Double_Window, 538
  Fl_Gl_Window, 586
  Fl_Overlay_Window, 724
  Fl_Widget, 1114
  Fl_Window, 1150
hide_all_windows
  Fl, 416
hide_widgets
  Fl_Tree_Item, 1055
hidedelay
  Fl_Tooltip, 999
highlight_data
  Fl_Text_Display, 946
highlight_text
  Fl_Text_Buffer, 918
highres_image
  Fl_Image_Surface, 635
history_lines
  Fl_Simple_Terminal, 834
hit_close
  Fl_Tabs, 906
HORIZONTAL
  Fl_Browser, 476
  Fl_Flex, 564
horizontal
  Fl_Flex, 568
  Fl_Pack, 727
HORIZONTAL_ALWAYS
  Fl_Browser, 476
hotspot
  Fl_Window, 1150
hour
  Fl_Clock_Output, 519
hoverdelay
  Fl_Tooltip, 999
hposition
  Fl_Browser, 479
  Fl_Tree, 1022
hscrollbar
  Fl_Browser, 488
hsv
  Fl_Color_Selector, 524
hsv2rgb
  Fl_Color_Selector, 524
hue
  Fl_Color_Selector, 524
icon
  Fl_Browser, 459
  Fl_Window, 1151
iconize
  Fl_Window, 1151
icons
  Fl_Window, 1152
iconsizes
  Fl_File_Browser, 545
  Fl_File_Selector, 551
ID
  Fl_Preferences, 755
idle
  Fl, 426
image
  Fl_Anim_GIF_Image, 438
  Fl_Image_Surface, 635
  Fl_Menu_Item, 704
  Fl_Shared_Image, 823
  Fl_Widget, 1115
IMAGE_BOUND
  Fl_Widget, 1097
image_bound
  Fl_Widget, 1116
in_selection
  Fl_Text_Display, 947
INACTIVE
  Fl_Widget, 1097
inactive
  Fl_Image, 630
includes
  Fl_Text_Selection, 977
incr_height
  Fl_Browser, 459
  Fl_Browser, 480
increment
  Fl_Valuator, 1076
index
  Fl_Input, 648
init_sizes
  Fl_Group, 601
  Fl_Table, 892
init_value
  Fl_Scheme_Choice, 802
inp_x
  Fl_Input_Choice, 663
input
  Fl_Input_Choice, 663
input_type
  Fl_Input, 648
insert
  Fl_Browser, 460
  Fl_Chart, 502
  Fl_Group, 602
  Fl_Input, 649
  Fl_Menu, 686
  Fl_Menu_Item, 705
  Fl_Menu_Item_Array, 1067
  Fl_String, 855
  Fl_Sys_Menu_Bar, 876
  Fl_Table, 892
  Fl_Text_Buffer, 918
  Fl_Text_Display, 947
  Fl_Timeout, 989
  Fl_Tree, 1022
  Fl_Tree_Item, 1055
  Fl_Tree_Item_Array, 1067
insert_
  Fl_Text_Buffer, 919
insert_above
  Fl_Tree, 1023
  Fl_Tree_Item, 1055
insert_mode
  Fl_Text_Editor, 970
insert_position
  Fl_Input, 649, 650
  Fl_Text_Display, 947, 948
insertfile
  Fl_Text_Buffer, 919
inserting
  Fl_Browser, 480
inset
  Fl_Rect, 784, 785
inside
  Fl_Widget, 1116
INVISIBLE
  Fl_Widget, 1097
is_animated
  Fl_Anim_GIF_Image, 439
is_close
  Fl_Tree, 1023, 1024
is_current
  Fl_Copy_Surface, 527
  Fl_Image_Surface, 635
  Fl_Printer, 776
  Fl_Surface_Device, 861
is_hscroll_visible
  Fl_Tree, 1024
is_interactive_resize
  Fl_Table, 892
is_label_copied
  Fl_Widget, 1116
is_open
  Fl_Tree, 1024, 1025
is_resizable
  Fl_Window, 1152
is_scheme
  Fl, 416
is_scrollbar
  Fl_Tree, 1025
is_selected
  Fl_Table, 892
  Fl_Tree, 1025, 1026
is_vscroll_visible
  Fl_Tree, 1026
is_word_separator
  Fl_Text_Buffer, 919
iso8859_1.h, 1698
iso8859_10.h, 1698
iso8859_11.h, 1700
iso8859_13.h, 1701
iso8859_14.h, 1702
iso8859_15.h, 1703
iso8859_16.h, 1704
iso8859_2.h, 1705
iso8859_3.h, 1706
iso8859_4.h, 1708
iso8859_5.h, 1709
iso8859_6.h, 1710
iso8859_7.h, 1711
iso8859_8.h, 1712
iso8859_9.h, 1713
iso8859_9e.h, 1714
item_at
  Fl_Browser, 460
  Fl_Browser_, 480
  Fl_Check_Browser, 506
item_clicked
  Fl_Tree, 1026
item_draw
  Fl_Browser, 461
  Fl_Browser_, 481
  Fl_Check_Browser, 506
item_draw_mode
  Fl_Tree, 1027
  Fl_Tree_Prefs, 1071
item_first
  Fl_Browser, 461
  Fl_Browser_, 481
  Fl_Check_Browser, 506
item_height
  Fl_Browser, 461
  Fl_Browser_, 481
  Fl_Check_Browser, 506
item_labelbgcolor
  Fl_Tree, 1027
  Fl_Tree_Prefs, 1071, 1072
item_labelbgcolor
  Fl_Tree, 1027
item_labelbgcolor
  Fl_Tree, 1027
item_labelfont
  Fl_Tree, 1028
Index

maintain_absolute_top_line_number
  Fl_Text_Display, 950
maintaining_absolute_top_line_number
  Fl_Text_Display, 951
make_current
  Fl_Gl_Window, 586
  Fl_Timeout, 989
  Fl_Window, 1153
make_overlay_current
  Fl_Gl_Window, 587
make_visible
  Fl_Browser, 466
manage_item_destroy
  Fl_Tree_Item_Array, 1067
margin
  Fl_Flex, 568–570
  Fl_Tooltip, 999
margin_height
  Fl_Tooltip, 1000
marginbottom
  Fl_Tree_Prefs, 1072
margins
  Fl_Paged_Device, 732
  Fl_PostScript_File_Device, 748
  Fl_Printer, 776
mark
  Fl_Input, 651
math.h, 1420
max_size
  Fl_RGB_Image, 794
maximum
  Fl_Progress, 780
  Fl_Valuator, 1076
maximum_size
  Fl_Input, 651
maxsize
  Fl_Chart, 502
measure
  Fl_Label, 672
  Fl_Menu_Item, 706
measure_deleted_lines
  Fl_Text_Display, 951
measure_label
  Fl_Widget, 1120
measure_proportional_character
  Fl_Text_Display, 951
measure_vline
  Fl_Text_Display, 952
mediumarrow.h, 1469
MEMORY
  Fl_Preferences, 756
menu
  Fl_Menu, 687
  Fl_Sys_Menu_Bar, 877
menu_end
  Fl_Menu, 688
menu_linespacing
  Fl, 416, 417
MENU_WINDOW
  Fl_Widget, 1097
menu_x
  Fl_Input_Choice, 663
menubutton
  Fl_Input_Choice, 664
middleline
  Fl_Browser, 467
min_delay
  Fl_Anim_GIF_Image, 442
minimum
  Fl_Progress, 780
  Fl_Valuator, 1076, 1077
minute
  Fl_Clock_Output, 519
MODAL
  Fl_Widget, 1097
modal
  Fl_Window, 1153
Windows handling functions, 274
mode
  Fl_ColorChooser, 524, 525
  Fl_Gl_Window, 587
  Fl_Menu, 688
  Fl_Sys_Menu_Bar, 877
move
  Fl_Browser, 467
  Fl_Tree_Item, 1057
  Fl_Tree_Item_Array, 1067
move_above
  Fl_Tree_Item, 1058
move_below
  Fl_Tree_Item, 1058
move_cursor
  Fl_Table, 892
move_down
  Fl_Text_Display, 952
move_intersection
  Fl_Tile, 982
move_into
  Fl_Tree_Item, 1058
move_left
  Fl_Text_Display, 952
move_right
  Fl_Text_Display, 952
move_up
  Fl_Text_Display, 952
mTabDist
  Fl_Text_Buffer, 926
mulelao.h, 1808
Multithreading support functions, 346
  awake, 347
  lock, 347
  thread_message, 348
  unlock, 348
mvalue
  Fl_Menu, 688

Generated by Doxygen
normalize
   Fl_RGB_Image, 794
   Fl_SVG_Image, 870
now
   Fl, 417
npos
   Fl_String, 859
num_schemes
   Fl_Scheme, 800
numericsort.c, 1469
   fl_casenumericsort, 1469
   fl_numericsort, 1469
onscreen
   Fl_Image_Surface, 635
offset_line_starts
   Fl_Text_Display, 952
on_extension_data
   Fl_Anim_GIF_Image, 440
on_frame_data
   Fl_Anim_GIF_Image, 440
on_insert
   Fl_Group, 602
   Fl_Scroll, 807
   Fl_Tabs, 907
on_move
   Fl_Group, 602
   Fl_Scroll, 808
   Fl_Tabs, 907
on_remove
   Fl_Flex, 570
   Fl_Group, 603
   Fl_Tabs, 907
open
   Fl_Tree, 1033
open_toggle
   Fl_Tree, 1034
opendeicon
   Fl_Tree, 1034
openicon
   Fl_Tree, 1034
   Fl_Tree_Prefs, 1072
operator+
   Fl_String.cxx, 1455
   Fl_String.H, 1334
operator++
   Fl_String, 855, 856
operator=
   Fl_Int_Vector, 667
   Fl_String, 856
operator==
   Fl_String.cxx, 1455
   Fl_String.H, 1335
operator[]
   Fl_Int_Vector, 667, 668
   Fl_String, 857
OPTIMIZE_MEMORY
   Fl_Anim_GIF_Image, 432
Option

Generated by Doxygen
radio

Fl_Menu_Item, 707

range

Fl_Valuator, 1077

readonly

Fl_Input_, 652

readqueue

Fl, 419

ready

Fl, 419

recalc_dimensions

Fl_Table, 892

recalc_scrollbars

Fl_Scroll, 808

recalc_tree

Fl_Tree, 1035

Fl_Tree_Item, 1060

rectangle_capture

Fl_Device_Plugin, 533

redisplay_range

Fl_Text_Display, 955

redo

Fl_Input_, 652

redraw

Fl_Widget, 1121

redraw_label

Fl_Widget, 1121

redraw_line

Fl_Browser_, 484

redraw_lines

Fl_Browser_, 484

redraw_overlay

Fl_Gl_Window, 589

Fl_Overlay_Window, 724

redraw_range

Fl_Table, 893

refcount

Fl_Shared_Image, 823

release

Fl, 419

Fl_Image, 631

Fl_Shared_Image, 823

Fl_Timeout, 989

release_widget_pointer

Safe widget deletion support functions, 350

reload_scheme

Fl, 420

remove

Fl_Browser, 467

Fl_Check_Browser, 509

Fl_Group, 603, 604

Fl_Menu_, 689

Fl_Sys_Menu_Bar, 877

Fl_Text_Buffer, 922

Fl_Tree, 1035

Fl_Tree_Item_Array, 1067

remove_

Fl_Text_Buffer, 922

r

Fl_Color_Chooser, 525

Fl_Rect, 785
remove_check
 Fl, 420
remove_child
 Fl_Tree_Item, 1061
remove_handler
 Events handling functions, 287
remove_icon
 Fl_Browser, 468
remove_key_binding
 Fl_Text_Editor, 974
remove_lines
 Fl_Simple_Terminal, 835
remove_system_handler
 Events handling functions, 287
remove_timeout
 Fl, 420
 Fl_Timeout, 989
removePlugin
 Fl_Plugin_Manager, 739
reparent
 Fl_Tree_Item, 1061
 Fl_Tree_Item_Array, 1068
repeat_timeout
 Fl, 420
 Fl_Timeout, 991
replace
 Fl_Chart, 502
 Fl_Input_, 653
 Fl_Menu_, 689
 Fl_String, 857, 858
 Fl_Sys_Menu_Bar, 877
 Fl_Text_Buffer, 922
 Fl_Tree_Item, 1061
 Fl_Tree_Item_Array, 1068
replace_child
 Fl_Tree_Item, 1062
replacing
 Fl_Browser_, 484
rescale
 Fl_Image_Surface, 636
reserve
 Fl_String, 858
reset_absolute_top_line_number
 Fl_Text_Display, 955
resizable
 Fl_Group, 604
resize
 Fl_Anim_GIF_Image, 440
 Fl_Browser_, 485
 Fl_Double_Window, 538
 Fl_Flex, 570
 Fl_Gl_Window, 589
 Fl_Group, 605
 Fl_Help_View, 616
 Fl_Input_, 653
 Fl_Input_Choice, 664
 Fl_Overlay_Window, 724
 Fl_Pack, 727
 Fl_Scroll, 808
 Fl_Spinner, 845
 Fl_String, 858
 Fl_SVG_Image, 870
 Fl_Table, 893
 Fl_Tabs, 908
 Fl_Text_Display, 955
 Fl_Tile, 982
 Fl_Tree, 1035
 Fl_Value_Input, 1081
 Fl_Widget, 1122
 Fl_Window, 1153
REVERSED
 Fl_Paged_Device, 730
rewind_lines
 Fl_Text_Buffer, 922
 Fl_Text_Display, 955
rgb
 Fl_Color_Chooser, 525
rgb2hsv
 Fl_Color_Chooser, 525
RGB_scaling
 Fl_Image, 631
Root
 Fl_Preferences, 756
root
 Fl_Tree, 1036
root_label
 Fl_Tree, 1036
ROOT_MASK
 Fl_Preferences, 756
rotate
 Fl_Paged_Device, 732
 Fl_PostScript_File_Device, 749
 Fl_Printer, 777
round
 Fl_Valuator, 1077
ROW
 Fl_Flex, 564
row_col_clamp
 Fl_Table, 893
row_header
 Fl_Table, 893
row_height
 Fl_Table, 893
row_height_all
 Fl_Table, 893
row_resize
 Fl_Table, 893
row_resize_min
 Fl_Table, 894
row_selected
 Fl_Table, 894
 Fl_Table_Row, 899
rows
 Fl_Table, 894
 Fl_Table_Row, 899
run
 Fl, 421
Safe widget deletion support functions, 348
  clear_widget_pointer, 349
delete_widget, 349
do_widget_deletion, 350
release_widget_pointer, 350
watch_widget_pointer, 350
saturation
  Fl_Color_Chooser, 526
SAVEAS_CONFIRM
  Fl_Native_File_Chooser, 717
savefile
  Fl_Text_Buffer, 923
scale
  Fl_Image, 631
  Fl_Paged_Device, 732
  Fl_PostScript_File_Device, 749
  Fl_Printer, 777
scaling_algorithm
  Fl_Image, 632
scheme
  Fl, 421
  scheme_cb_
    Fl_Scheme_Choice, 802
Screen functions, 293
  keyboard_screen_scaling, 294
  screen_count, 295
  screen_dpi, 295
  screen_num, 295
  screen_scale, 296
  screen_scaling_supported, 296
  screen_work_area, 296, 297
  screen_xywh, 297, 298
screen_count
  Screen functions, 295
screen_dpi
  Screen functions, 295
screen_num
  Fl_Window, 1154
  Screen functions, 295
screen_scale
  Screen functions, 296
screen_scaling_supported
  Screen functions, 296
screen_work_area
  Screen functions, 296, 297
screen_xywh
  Screen functions, 297, 298
scroll
  Fl_Text_Display, 956
scroll_
    Fl_Text_Display, 956
scroll_timer_cb
    Fl_Text_Display, 956
scroll_to
    Fl_Scroll, 809
scrollbar
  Fl_Browser_, 488
scrollbar_align
  Fl_Text_Display, 957
scrollbar_left
  Fl_Browser_, 485
scrollbar_right
  Fl_Browser_, 485
scrollbar_size
  Fl, 422
  Fl_Browser_, 485
  Fl_Help_View, 616, 617
  Fl_Scroll, 809
  Fl_Table, 894
  Fl_Text_Display, 957
  Fl_Tree, 1036
scrollbar_width
  Fl_Browser_, 486
  Fl_Text_Display, 958
scrollvalue
  Fl_Slider, 842
search_backward
  Fl_Text_Buffer, 923
search_forward
  Fl_Text_Buffer, 923
second
  Fl_Clock_Output, 519
secondary_selection_color
  Fl_Text_Display, 958
secondary_selection_text
  Fl_Text_Display, 958
seconds_between
  Fl, 423
seconds_since
  Fl, 423
select
  Fl_Browser, 468
  Fl_Browser_, 486
  Fl_Tree, 1037
  Fl_Tree_Item, 1062
select_all
  Fl_Tree, 1038
  Fl_Tree_Item, 1062
select_all_rows
  Fl_Table_Row, 899
select_only
  Fl_Browser_, 487
  Fl_Tree, 1038
select_row
  Fl_Table_Row, 899
select_toggle
  Fl_Tree, 1039
selectbox
  Fl_Tree, 1039
selected
  Fl_Browser, 468
  Fl_Text_Selection, 977, 978
Selection & Clipboard functions, 292
Selection & Clipboard functions, 289
add_clipboard_notify, 290
clipboard_contains, 290
copy, 290
dnd, 291
paste, 291
selection, 292
selection_owner, 292
selection_to_clipboard, 292, 293
selection_color
  Fl_Widget, 1122
selection_owner
  Selection & Clipboard functions, 292
selection_text
  Fl_Text_Buffer, 924
selection_toClipboard
  Selection & Clipboard functions, 292, 293
selectmode
  Fl_Tree, 1039
  Fl_Tree_Prefs, 1072
set
  Fl_Button, 492
  Fl_FormsBitmap, 573
  Fl_FormsPixmap, 575
  Fl_Menu_Item, 707
  Fl_Preferences, 768–771
  Fl_Text_Selection, 978
set_active
  Fl_Widget, 1123
set_atclose
  Windows handling functions, 274
set_box_color
  Fl, 423
set_changed
  Fl_Widget, 1123
set_color
  Color & Font functions, 313
set_current
  Fl_Copy_Surface, 528
  Fl_Image_Surface, 636
  Fl_PostScript_File_Device, 750
  Fl_Printer, 778
  Fl_Surface_Device, 862
set_draw_cb
  Fl_Cairo_Window, 496
set_font
  Color & Font functions, 313
set_fonts
  Color & Font functions, 314
set_idle
  Fl, 423
set_item_focus
  Fl_Tree, 1040
set_menu_window
  Fl_Window, 1154
set_modal
  Fl_Window, 1154
set_non_modal
  Fl_Window, 1154
set_output
  Fl_Widget, 1123
set_selection
  Fl_Table, 895
set_tooltip_window
  Fl_Window, 1154
set_visible
  Fl_Widget, 1123
set_visible_focus
  Fl_Widget, 1123
setonly
  Fl_Menu_Item, 708
shadow
  Fl_Clock_Output, 519, 520
shape
  Fl_Window, 1155
shortcut
  Fl_Button, 492, 493
  Fl_Input, 654
  Fl_Menu_Item, 708
  Fl_Text_Display, 958
  Fl_Value_Input, 1081
SHORTCUT_LABEL
  Fl_Widget, 1097
shortcut_label
  Fl_Widget, 1123, 1124
show
  Fl_Browser, 468, 469
  Fl_Dialogs, 539
  Fl_CI_Window, 589
  Fl_Help_Dialog, 609
  Fl_Native_File_Chooser, 720
  Fl_Overlay_Window, 725
  Fl_Single_Window, 839
  Fl_Widget, 1124
  Fl_Window, 1155, 1156
show_cursor
  Fl_Text_Display, 959
show_insert_position
  Fl_Text_Display, 959
show_item
  Fl_Tree, 1040
show_item_bottom
  Fl_Tree, 1040
show_item_middle
  Fl_Tree, 1041
show_item_top
  Fl_Tree, 1041
show_self
  Fl_Tree, 1041
  Fl_Tree_Item, 1063
show_widgets
  Fl_Tree_Item, 1063
showcollapse
  Fl_Tree, 1041
  Fl_Tree_Prefs, 1072
showHiddenButton
  Fl_File_Chooser, 552
shown
  FL_File_Chooser, 552
  FL_Window, 1156
showroot
  FL_Tree, 1042
  FL_Tree_Prefs, 1073
SIMPLE_CURSOR
  FL_Text_Display, 936
size
  FL_Browser, 469
  FL_Chart, 503
  FL_Input_, 654
  FL_Int_Vector, 668
  FL_Menu_, 689
  FL_Menu_Item, 708
  FL_Preferences, 771
  FL_String, 859
  FL_Tooltip, 1000
  FL_Widget, 1124
size_range
  FL_Window, 1156
sizes
  FL_Group, 606
skip_displayed_characters
  FL_Text_Buffer, 924
skip_lines
  FL_Text_Display, 959
slider_size
  FL_Slider, 842
slowarrow.h, 1471
soft
  FL_Adjuster, 428, 429
  FL_Value_Input, 1081
  FL_Value_Output, 1084, 1085
sort
  FL_Browser_, 487
sortorder
  FL_Tree, 1042
  FL_Tree_Prefs, 1073
spacing
  FL_Flex, 571
  FL_Flex.h, 1502
speed
  FL_Anim_GIF_Image, 441
spelling_underline_color
  FL_Text_Display, 959, 960
start
  FL_Anim_GIF_Image, 441
  FL_Text_Selection, 978
start_job
  FL_Paged_Device, 733
  FL_PostScript_File_Device, 750
start_page
  FL_Paged_Device, 733
static_value
  FL_Input_, 655
stay_at_bottom
  FL_Simple_Terminal, 835, 836
step
  FL_Counter, 532
  FL_Spinner, 846
  FL_Valuator, 1077
stop
  FL_Anim_GIF_Image, 441
STRICT_RFC3629
  Unicode and UTF-8 functions, 356
string_width
  FL_Text_Display, 960
strlen
  FL_String, 859
style_buffer
  FL_Text_Display, 960
style_table
  FL_Simple_Terminal, 836
style_table_size
  FL_Simple_Terminal, 837
submenu
  FL_Menu_Item, 708
substr
  FL_String, 859
surface
  FL_Surface_Device, 862
suspended
  FL_Timer, 994
swap
  FL_Browser, 469
  swap_buffers
    FL_Gl_Window, 590
  swap_children
    FL_Tree_Item, 1063
swapping
  FL_Browser_, 487
symbol_.h, 1525
SYSTEM
  FL_Preferences, 756
SYSTEM_L
  FL_Preferences, 756
tab_align
  FL_Tabs, 908
tab_cell_nav
  FL_Table, 895
tab_distance
  FL_Text_Buffer, 924
tab_nav
  FL_Input_, 655, 656
  FL_Text_Editor, 974
tabbing_mode_automatic
  FL_Sys_Menu_Bar, 873
tabbing_mode_none
  FL_Sys_Menu_Bar, 873
tabbing_mode_preferred
  FL_Sys_Menu_Bar, 873
table_box
  FL_Table, 896
table_resized
  FL_Table, 896

Generated by Doxygen
INDEX 1869

Generated by Doxygen
INDEX

- fl_getenv, 359
- fl_make_path, 360
- fl_make_path_for_file, 360
- fl_mkdir, 360
- fl_nonspacing, 360
- fl_open, 361
- fl_open_ext, 361
- fl_putenv, 361
- fl_rename, 362
- fl_rmdir, 362
- fl_stat, 363
- fl_system, 363
- fl_ucs_to_Utf16, 363
- fl_open, 364
- fl_ut8back, 364
- fl_ut8bytes, 364
- fl_ut8decode, 364
- fl_ut8encode, 365
- fl_ut8from_mb, 365
- fl_ut8froma, 365
- fl_ut8fromwc, 366
- fl_ut8fromw, 366
- fl_ut8Blen, 366
- fl_ut8Blen1, 366
- fl_ut8locale, 367
- fl_ut8strnlen, 367
- fl_ut8test, 367
- fl_ut8to_mb, 367
- fl_ut8toa, 368
- fl_ut8toUtf16, 368
- fl_ut8towc, 368
- fl_ut8_nb_char, 369
- fl_ut8_strnncmp, 369
- fl_ut8_strnscmp, 369
- fl_ut8tolower, 370
- fl_ut8toupper, 370
- fl_wcwidth, 370
- fl_wcwidth_, 370
- STRICT RFC3629, 356
- UNKNOWN_ROOT_TYPE
- Fl_Preferences, 756
- unlock
- Multithreading support functions, 348
- untranslated
- Fl_Copy_Surface, 529
- Fl_EPS_File_Surface, 543
- Fl_Image_Surface, 637
- Fl_PostScript_File_Device, 751
- Fl_Printer, 778
- Fl_SVG_File_Surface, 865
- Fl_Widget_Surface, 1135
- up_down_position
- Fl_Input, 657
- update
- Fl_Menu_Bar, 693
- Fl_Sys_Menu_Bar, 878
- Fl_Text_Selection, 979
- update_child
- Fl_Group, 606
- update_h_scrollbar
- Fl_Text_Display, 962
- update_line_starts
- Fl_Text_Display, 962
- update_menubutton
- Fl_Input_Choice, 664
- update_prev_next
- Fl_Tree_Item, 1064
- update_v_scrollbar
- Fl_Text_Display, 962
- USE_FILTER_EXT
- Fl_Native_File_Chooser, 717
- use_high_res_GL
- Fl, 424
- USER
- Fl_Preferences, 756
- user_data
- Fl_Widget, 1127
- USER_L
- Fl_Preferences, 756
- userdeicon
- Fl_Tree_Item, 1064
- Fl_Tree_Prefs, 1073
- USERFLAG1
- Fl_Widget, 1097
- USERFLAG2
- Fl_Widget, 1097
- USERFLAG3
- Fl_Widget, 1097
- usericon
- Fl_Tree, 1042
- Fl_Tree_Item, 1064
- utf8.h, 1813
- utf8_internal.h, 1471
- valid
- Fl_Anim_GIF_Image, 441
- Fl_Gl_Window, 590
- value
- Fl_Browser, 471
- Fl_Button, 493
- Fl_Choice, 513
- Fl_Clock_Output, 520
- Fl_Color_Chooser, 526
- Fl_File_Chooser, 552
- Fl/File_Input, 560
- Fl_HELP_Dialog, 609
- Fl_HELP_View, 618
- Fl_Input, 658
- Fl_Input_Choice, 664, 665
- Fl_Menu, 690, 691
- Fl_Menu_Item, 709
- Fl_Progress, 780
- Fl_Scrollbar, 812, 813
- Fl_Shortcut_Button, 826
- Fl_Spinner, 846
- Fl_Tabs, 908
- Fl_Valuator, 1077
  Fl_Window, 1158
Ximint.h, 1816
Xlibint.h, 1816
xposition
   Fl_Scroll, 810
Xutf8.h, 1473
xy_to_position
   Fl_Text_Display, 966
xy_to_rowcol
   Fl_Text_Display, 966

y
   Fl_Widget, 1131, 1132
yposition
   Fl_Scroll, 810