the high-performance embedded GUI

User Guide

Version 5

Express Logic
858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
http://www.expresslogic.com
# Table of Contents

Table of Contents........................................................................................................... iii
About This Guide .............................................................................................................. xxii
   Organization .................................................................................................................. xxii
Guide Conventions ......................................................................................................... xxiii
GUIX Data Types ........................................................................................................... xxiv
Customer Support Center .............................................................................................. xxvi
   What We Need From You ............................................................................................ xxvi
   Where to Send Comments About This Guide .......................................................... xxviii
Chapter 1: Introduction to GUIX ....................................................................................... 1
   GUIX Feature Overview ................................................................................................. 2
      ANSI C Source Code .................................................................................................. 3
      Not A Black Box ......................................................................................................... 3
   Embedded GUI Applications ......................................................................................... 3
      Real-time GUI Software ............................................................................................ 3
   GUIX Benefits ............................................................................................................... 4
      Improved Responsiveness ......................................................................................... 4
      Software Maintenance ............................................................................................... 4
      Increased Throughput ............................................................................................... 4
      Processor Isolation .................................................................................................... 4
      Ease of Use ................................................................................................................ 4
      Improve Time to Market ........................................................................................... 4
      Protecting the Software Investment .......................................................................... 5
Chapter 2: Installation and Use of GUIX ........................................................................... 6
   Host Considerations ...................................................................................................... 7
   Target Considerations .................................................................................................. 7
   Product Distribution ..................................................................................................... 7
   GUIX Installation ........................................................................................................... 8
   Using GUIX .................................................................................................................... 8
   Troubleshooting .......................................................................................................... 8
   Configuration Options ................................................................................................. 10

GUIX User Guide

iii
Chapter 3: Functional Overview of GUIX

Execution Overview ......................................................... 14
Initialization .................................................................. 14
Application Interface Calls .............................................. 15
Internal GUIX Thread ...................................................... 15
Event Processing ............................................................. 17
Drawing ........................................................................... 17
User Input ........................................................................ 19
Modal Dialog Execution ................................................... 19
Periodic Processing ......................................................... 20
Display Driver ................................................................. 20
Display Memory Architectures .......................................... 21
String Encoding ............................................................... 24
  Static and Dynamic Strings ........................................... 24
  Passing GX_STRING arguments .................................. 25
  GUIX String Table ...................................................... 26
  Bi-directional Text Display ......................................... 27
Memory Usage ................................................................. 28
  Static Memory Usage .................................................. 28
  Dynamic Memory Usage ............................................. 29
GUIX Components ........................................................ 30
GUIX System Component ................................................ 31
  Initialization .............................................................. 31
  Thread Processing ....................................................... 32
  RTOS Binding ............................................................ 32
  Multithread Safety ....................................................... 32
  System Timers ............................................................ 32
  System Error Handling ............................................... 34
GUIX Canvas Component ................................................ 34
  Canvas Creation ........................................................ 35
  Canvas Control Block ................................................ 35
  Canvas Alpha Channel .............................................. 36
  Canvas Offset ............................................................ 36
Scrolling ........................................................................................................ 57
Event Notification .......................................................................................... 58
Event Processing ........................................................................................... 58
GUIX Image Reader Component .................................................................... 58
GUIX Animation Component ......................................................................... 59
GUIX Utility Component ................................................................................ 62
Chapter 4: Description of GUIX Services ..................................................... 64
gx_accordion_menu_create ......................................................................... 77
gx_accordion_menu_draw ............................................................................. 80
gx_accordion_menu_event_process .............................................................. 81
gx_accordion_menu_position ...................................................................... 83
gx_animation_canvas_define ........................................................................ 84
gx_animation_create ..................................................................................... 86
gx_animation_drag_disable .......................................................................... 88
gx_animation_drag_enable .......................................................................... 89
gx_animation_landing_speed_set ................................................................. 91
gx_animation_start ....................................................................................... 92
gx_animation_stop ....................................................................................... 94
gx_binres_language_table_load ................................................................... 95
gx_binres_language_table_load_ext ............................................................. 97
gx_binres_theme_load ................................................................................... 99
gx_brush_default ......................................................................................... 101
gx_brush_define .......................................................................................... 102
gx_button_background_draw ....................................................................... 104
gx_button_create ........................................................................................ 105
gx_button_deselect ...................................................................................... 107
gx_button_draw ............................................................................................ 109
gx_button_event_process ............................................................................ 110
gx_button_select ........................................................................................ 112
gx_canvas_alpha_set ................................................................................... 113
gx_canvas_arc_draw ..................................................................................... 115
gx_canvas_block_move ............................................................................... 116
gx_canvas_circle_draw ............................................................................... 118
gx_canvas_create ....................................................................................... 119
gx_canvas_delete............................................................................................................. 121
gx_canvas_drawing_complete .......................................................................................... 122
gx_canvas_drawing_initiate ......................................................................................... 124
gx_canvas_ellipse_draw ............................................................................................. 126
gx_canvas_hardware_layer_bind ................................................................................. 127
gx_canvas_hide ........................................................................................................... 129
gx_canvas_line_draw ..................................................................................................... 130
gx_canvas_memory_define .......................................................................................... 131
gx_canvas_mouse_define ............................................................................................ 132
gx_canvas_mouse_hide ............................................................................................... 133
gx_canvas_mouse_show ............................................................................................... 134
gx_canvas_offset_set .................................................................................................. 135
gx_canvas_pie_draw ................................................................................................... 137
gx_canvas_pixel_draw .................................................................................................. 138
gx_canvas_pixelmap_blend ......................................................................................... 140
gx_canvas_pixelmap_draw ......................................................................................... 142
gx_canvas_pixelmap_get ............................................................................................. 143
gx_canvas_pixelmap_rotate ....................................................................................... 144
gx_canvas_pixelmap_tile ........................................................................................... 146
gx_canvas_polygon_draw ............................................................................................ 147
gx_canvas_rectangle_draw ....................................................................................... 148
gx_canvas_rotated_text_draw .................................................................................. 149
gx_canvas_rotated_text_draw_ext ............................................................................. 151
gx_canvas_shift .......................................................................................................... 153
gx_canvas_show ......................................................................................................... 154
gx_canvas_text_draw .................................................................................................. 155
gx_canvas_text_draw_ext ............................................................................................ 157
gx_checkbox_create .................................................................................................... 159
gx_checkbox_draw ....................................................................................................... 161
gx_checkbox_event_process ........................................................................................ 162
gx_checkbox_pixelmap_set ......................................................................................... 164
gx_checkbox_select ..................................................................................................... 166
gx_circular_gauge_angle_get .................................................................................... 167
gx_circular_gauge_angle_set .................................................................................... 168
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>gx_circular_gauge_animation_set</td>
<td>169</td>
</tr>
<tr>
<td>gx_circular_gauge_background_draw</td>
<td>170</td>
</tr>
<tr>
<td>gx_circular_gauge_create</td>
<td>171</td>
</tr>
<tr>
<td>gx_circular_gauge_draw</td>
<td>173</td>
</tr>
<tr>
<td>gx_circular_gauge_event_process</td>
<td>174</td>
</tr>
<tr>
<td>gx_context_brush_default</td>
<td>176</td>
</tr>
<tr>
<td>gx_context_brush_define</td>
<td>177</td>
</tr>
<tr>
<td>gx_context_brush_get</td>
<td>179</td>
</tr>
<tr>
<td>gx_context_brush_pattern_set</td>
<td>180</td>
</tr>
<tr>
<td>gx_context_brush_set</td>
<td>182</td>
</tr>
<tr>
<td>gx_context_brush_style_set</td>
<td>184</td>
</tr>
<tr>
<td>gx_context_brush_width_set</td>
<td>185</td>
</tr>
<tr>
<td>gx_context_color_get</td>
<td>186</td>
</tr>
<tr>
<td>gx_context_fill_color_set</td>
<td>187</td>
</tr>
<tr>
<td>gx_context_font_get</td>
<td>188</td>
</tr>
<tr>
<td>gx_context_font_set</td>
<td>190</td>
</tr>
<tr>
<td>gx_context_line_color_set</td>
<td>191</td>
</tr>
<tr>
<td>gx_context_pixelmap_get</td>
<td>192</td>
</tr>
<tr>
<td>gx_context_pixelmap_set</td>
<td>194</td>
</tr>
<tr>
<td>gx_context_raw_brush_define</td>
<td>195</td>
</tr>
<tr>
<td>gx_context_raw_fill_color_set</td>
<td>197</td>
</tr>
<tr>
<td>gx_context_raw_line_color_set</td>
<td>198</td>
</tr>
<tr>
<td>gx_context_string_get</td>
<td>200</td>
</tr>
<tr>
<td>gx_context_string_get_ext</td>
<td>202</td>
</tr>
<tr>
<td>gx_display_active_language_set</td>
<td>204</td>
</tr>
<tr>
<td>gx_display_color_set</td>
<td>205</td>
</tr>
<tr>
<td>gx_display_color_table_set</td>
<td>206</td>
</tr>
<tr>
<td>gx_display_create</td>
<td>207</td>
</tr>
<tr>
<td>gx_display_delete</td>
<td>209</td>
</tr>
<tr>
<td>gx_display_font_table_set</td>
<td>210</td>
</tr>
<tr>
<td>gx_display_language_table_get</td>
<td>211</td>
</tr>
<tr>
<td>gx_display_language_table_get_ext</td>
<td>213</td>
</tr>
<tr>
<td>gx_display_language_table_set</td>
<td>214</td>
</tr>
<tr>
<td>gx_display_language_table_set_ext</td>
<td>216</td>
</tr>
</tbody>
</table>
gx_display_pixelmap_table_set.................................................................218
gx_display_string_get.............................................................................219
gx_display_string_get_ext.........................................................................220
gx_display_string_table_get.........................................................................221
gx_display_string_table_get_ext.................................................................223
gx_display_theme_install............................................................................224
gx_drop_list_close......................................................................................226
gx_drop_list_create......................................................................................227
gx_drop_list_event_process..........................................................................230
gx_drop_list_open.........................................................................................232
gx_drop_list_pixelmap_set............................................................................233
gx_drop_list_popup_get................................................................................235
gx_horizontal_list_children_position.......................................................236
gx_horizontal_list_create............................................................................237
gx_horizontal_list_event_process..............................................................239
gx_horizontal_list_page_index_set............................................................241
gx_horizontal_list_selected_index_get.......................................................242
gx_horizontal_list_selected_set..................................................................243
gx_horizontal_list_selected_widget_get.....................................................244
gx_horizontal_list_total_columns_set.........................................................246
gx_horizontalScrollbar_create..................................................................247
gx_icon_button_create................................................................................249
gx_icon_button_draw.....................................................................................251
gx_icon_button_pixelmap_set........................................................................252
gx_icon_background_draw.............................................................................253
gx_icon_create............................................................................................254
gx_icon_draw...............................................................................................256
gx_icon_event_process................................................................................257
gx_icon_pixelmap_set..................................................................................258
gx_image_reader_create..............................................................................259
gx_image_reader_palette_set.......................................................................261
gx_image_reader_start................................................................................262
gx_line_chart_axis_draw.............................................................................264
gx_line_chart_create..................................................................................265
gx_line_chart_data_draw
gx_line_chart_draw
gx_line_chart_update
gx_line_chart_y_scale_calculate
gx_menu_create
gx_menu_draw
gx_menu_insert
gx_menu_remove
gx_menu_text_draw
gx_menu_text_offset_set
gx_multi_line_text_button_create
gx_multi_line_text_button_draw
gx_multi_line_text_button_event_process
gx_multi_line_text_button_text_draw
gx_multi_line_text_button_text_id_set
gx_multi_line_text_button_text_set
gx_multi_line_text_button_text_set_ext
gx_multi_line_text_input_backspace
gx_multi_line_text_input_buffer_clear
gx_multi_line_text_input_buffer_get
gx_multi_line_text_input_char_insert
gx_multi_line_text_input_char_insert_ext
gx_multi_line_text_input_create
gx_multi_line_text_input_cursor_pos_get
gx_multi_line_text_input_delete
gx_multi_line_text_input_down_arrow
gx_multi_line_text_input_end
gx_multi_line_text_input_event_process
gx_multi_line_text_input_fill_color_set
gx_multi_line_text_input_home
gx_multi_line_text_input_left_arrow
gx_multi_line_text_input_right_arrow
gx_multi_line_text_input_style_add
gx_multi_line_text_input_style_remove
gx_pixelmap_button_create.................................................................369
gx_pixelmap_button_draw........................................................................371
gx_pixelmap_button_event_process ......................................................372
gx_pixelmap_button_pixelmap_set..........................................................374
gx_pixelmap_prompt_create.................................................................376
gx_pixelmap_prompt_draw....................................................................378
gx_pixelmap_prompt_pixelmap_set.........................................................379
gx_pixelmap_slider_create....................................................................381
gx_pixelmap_slider_draw......................................................................384
gx_pixelmap_slider_event_process.......................................................385
gx_pixelmap_slider_pixelmap_set..........................................................387
gx_progress_bar_background_draw.......................................................389
gx_progress_bar_create.....................................................................390
gx_progress_bar_draw.......................................................................392
gx_progress_bar_event_process............................................................393
gx_progress_bar_font_set...................................................................395
gx_progress_bar_info_set...................................................................396
gx_progress_bar_pixelmap_set.............................................................398
gx_progress_bar_range_set................................................................399
gx_progress_bar_text_color_set............................................................400
gx_progress_bar_text_draw..................................................................402
gx_progress_bar_value_set..................................................................403
gx_prompt_create.............................................................................404
gx_prompt_draw...............................................................................406
gx_prompt_font_set..........................................................................407
gx_prompt_text_color_set..................................................................408
gx_prompt_text_draw...........................................................................410
gx_prompt_text_get............................................................................411
gx_prompt_text_get_ext.....................................................................412
gx_prompt_text_id_set......................................................................413
gx_prompt_text_set............................................................................414
gx_prompt_text_set_ext.....................................................................416
gx_radial_progress_bar_anchor_set.....................................................418
gx_radial_progress_bar_background_draw..........................................419
gx_radial_progress_bar_create ........................................................................ 420
gx_radial_progress_bar_draw ........................................................................ 422
gx_radial_progress_bar_event_process .......................................................... 423
gx_radial_progress_bar_font_set ................................................................... 425
gx_radial_progress_bar_info_set ................................................................... 426
gx_radial_progress_bar_text_color_set .......................................................... 428
gx_radial_progress_bar_text_draw ................................................................. 430
gx_radial_progress_bar_value_set .................................................................. 432
gx_radio_button_create .................................................................................. 434
gx_radio_button_draw ..................................................................................... 436
gx_radio_button_pixelmap_set ...................................................................... 437
gx_radial_slider_anchor_angles_set ............................................................. 439
gx_radial_slider_angle_set .......................................................................... 441
gx_radial_slider_animation_set .................................................................... 442
gx_radial_slider_animation_start ................................................................. 444
gx_radial_slider_create .............................................................................. 445
gx_radial_slider_draw ................................................................................... 447
gx_radial_slider_event_process .................................................................... 448
gx_radial_slider_info_get ............................................................................ 450
gx_radial_slider_info_set ........................................................................... 451
gx_radial_slider_pixelmap_set ..................................................................... 453
gx_screen_stack_create ............................................................................... 454
gx_screen_stack_pop .................................................................................... 456
gx_screen_stack_push ................................................................................... 457
gx_screen_stack_reset .................................................................................. 459
gx_scroll_thumb_create ............................................................................... 460
gx_scroll_thumb_draw ................................................................................... 462
gx_scroll_thumb_event_process .................................................................... 463
gx_scroll_wheel_create ............................................................................... 465
gx_scroll_wheel_event_process .................................................................... 467
gx_scroll_wheel_gradient_alpha_set ............................................................ 469
gx_scroll_wheel_row_height_set .................................................................. 471
gx_scroll_wheel_selected_background_set .................................................... 473
gx_scroll_wheel_selected_get ...................................................................... 475
/ GUIX User Guide

gx_scroll_wheel_selected_set

gx_scroll_wheel_speed_set

gx_scroll_wheel_total_rows_set

gx_scrollbar_draw

gx_scrollbar_event_process

gx_scrollbar_limit_check

gx_scrollbar_reset

gx_single_line_text_input_backspace

gx_single_line_text_input_buffer_clear

gx_single_line_text_input_buffer_get

gx_single_line_text_input_character_delete

gx_single_line_text_input_character_insert

gx_single_line_text_input_create

gx_single_line_text_input_draw

gx_single_line_text_input_draw_position_get

gx_single_line_text_input_end

gx_single_line_text_input_event_process

gx_single_line_text_input_fill_color_set

gx_single_line_text_input_home

gx_single_line_text_input_left_arrow

gx_single_line_text_input_position_get

gx_single_line_text_input_right_arrow

gx_single_line_text_input_style_add

gx_single_line_text_input_style_remove

gx_single_line_text_input_style_set

gx_single_line_text_input_text_color_set

gx_single_line_text_input_text_select

gx_single_line_text_input_text_set

gx_single_line_text_input_text_set_ext

gx_slider_create

gx_slider_draw

gx_slider_event_process

gx_slider_info_set

gx_slider_needle_draw

477

478

480

482

483

485

486

488

490

492

494

496

498

500

502

504

506

508

510

512

514

516

518

520

522

524

526

528

530

532

534

535

537

539
gx_slider_needle_position_get ................................................................. 540
gx_slider_tickmarks_draw ................................................................. 542
gx_slider_travel_get ........................................................................... 543
gx_slider_value_calculate ..................................................................... 545
gx_slider_value_set ............................................................................... 547
gx_sprite_create .................................................................................. 549
gx_sprite_current_frame_set ................................................................. 551
gx_sprite_frame_list_set ........................................................................ 552
gx_sprite_start ...................................................................................... 554
gx_sprite_stop ....................................................................................... 555
gx_string_scroll_wheel_create ............................................................... 556
gx_string_scroll_wheel_string_id_list_set ........................................... 558
gx_string_scroll_wheel_string_list_set ................................................ 560
gx_studio_widget_create ........................................................................ 562
gx_studio_named_widget_create ............................................................. 564
gx_studio_display_configure ................................................................. 566
gx_system_active_language_set .............................................................. 568
gx_system_animation_get ......................................................................... 569
gx_system_animation_free ....................................................................... 571
gx_system_bidi_text_disable .................................................................... 573
gx_system_bidi_text_enable .................................................................... 574
gx_system_canvas_refresh ....................................................................... 575
gx_system_dirty_mark ............................................................................ 577
gx_system_dirty_partial_add .................................................................... 579
gx_system_draw_context_get .................................................................... 581
gx_system_event_fold ............................................................................. 582
gx_system_event_send ............................................................................. 584
gx_system_focus_claim ........................................................................... 586
gx_system_initialize .............................................................................. 587
gx_system_language_table_get .................................................................. 588
gx_system_language_table_set ............................................................... 589
gx_system_memory_allocator_set ............................................................ 590
gx_system_pen_configure ......................................................................... 592
gx_system_screen_stack_create ............................................................. 594
gx_system_screen_stack_get ................................................................. 596
gx_system_screen_stack_pop ............................................................... 598
gx_system_screen_stack_push ............................................................. 599
gx_system_screen_stack_reset ............................................................. 601
gx_system_scroll_appearance_get ......................................................... 602
gx_system_scroll_appearance_set .......................................................... 604
gx_system_start .................................................................................... 606
gx_system_string_get ............................................................................. 607
gx_system_string_table_get .................................................................... 608
gx_system_string_width_get ................................................................... 610
gx_system_string_width_get_ext ............................................................. 612
gx_system_timer_start ............................................................................ 614
gx_system_timer_stop ............................................................................. 616
gx_system_version_string_get ................................................................. 618
gx_system_version_string_get_ext ........................................................... 619
gx_system_widget_find ........................................................................... 621
gx_text_button_create ............................................................................ 623
gx_text_button_draw ............................................................................... 625
gx_text_button_font_set ........................................................................... 626
gx_text_button_text_color_set ................................................................. 627
gx_text_button_text_draw ........................................................................ 629
gx_text_button_text_get .......................................................................... 631
gx_text_button_text_get_ext .................................................................... 632
gx_text_button_text_id_set ...................................................................... 633
gx_text_button_text_set ........................................................................... 634
gx_text_button_text_set_ext ..................................................................... 636
gx_text_input_cursor_blink_interval_set .................................................. 638
gx_text_input_cursor_height_set .............................................................. 639
gx_text_input_cursor_width_set ............................................................... 640
gx_text_scroll_wheel_callback_set .......................................................... 641
gx_text_scroll_wheel_callback_set_ext ..................................................... 643
gx_text_scroll_wheel_create .................................................................... 645
gx_text_scroll_wheel_draw ....................................................................... 647
gx_text_scroll_wheel_font_set ................................................................. 649
gx_text_scroll_wheel_text_color_set ................................................................. 651
gx_tree_view_create ..................................................................................... 653
gx_tree_view_draw ......................................................................................... 655
gx_tree_view_event_process ........................................................................... 656
gx_tree_view_indentation_set ........................................................................ 658
gx_tree_view_position .................................................................................... 659
gx_tree_view_root_line_color_set .................................................................... 660
gx_tree_view_root_pixelmap_set ....................................................................... 661
gx_tree_view_selected_get ................................................................................ 662
gx_tree_view_selected_set ................................................................................ 663
gx_utility_canvas_to_bmp ............................................................................... 664
gx_utility_gradient_create .............................................................................. 666
gx_utility_gradient_delete .............................................................................. 668
gx_utility_ltoa ............................................................................................... 669
gx_utility_math_acos ...................................................................................... 671
gx_utility_math_asin ....................................................................................... 673
gx_utility_math_cos ....................................................................................... 675
gx_utility_math_sin ....................................................................................... 677
gx_utility_math_sqrt ...................................................................................... 679
gx_utility_pixelmap_resize ............................................................................. 680
gx_utility_pixelmap_rotate ............................................................................. 682
gx_utility_pixelmap_simple_rotate ................................................................. 684
gx_utility_rectangle_center ............................................................................ 686
gx_utility_rectangle_center_find ................................................................. 687
gx_utility_rectangle_combine ........................................................................... 688
gx_utility_rectangle_compare ......................................................................... 689
gx_utility_rectangle_define ............................................................................. 690
gx_utility_rectangle_overlap_detect ............................................................. 692
gx_utility_rectangle_point_detect ................................................................. 693
gx_utility_rectangle_resize ............................................................................ 695
gx_utility_rectangle_shift ............................................................................... 696
gx_utility_string_to_alphamap ...................................................................... 697
gx_utility_string_to_alphamap_ext ................................................................. 699
gx_vertical_list_children_position ............................................................... 701
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gx_widget_first_child_get</code></td>
<td>763</td>
</tr>
<tr>
<td><code>gx_widget_focus_next</code></td>
<td>764</td>
</tr>
<tr>
<td><code>gx_widget_focus_previous</code></td>
<td>765</td>
</tr>
<tr>
<td><code>gx_widget_font_get</code></td>
<td>766</td>
</tr>
<tr>
<td><code>gx_widget_free</code></td>
<td>767</td>
</tr>
<tr>
<td><code>gx_widget_front_move</code></td>
<td>769</td>
</tr>
<tr>
<td><code>gx_widget_height_get</code></td>
<td>771</td>
</tr>
<tr>
<td><code>gx_widget_hide</code></td>
<td>773</td>
</tr>
<tr>
<td><code>gx_widget_last_child_get</code></td>
<td>774</td>
</tr>
<tr>
<td><code>gx_widget_next_sibling_get</code></td>
<td>775</td>
</tr>
<tr>
<td><code>gx_widget_parent_get</code></td>
<td>777</td>
</tr>
<tr>
<td><code>gx_widget_pixelmap_get</code></td>
<td>778</td>
</tr>
<tr>
<td><code>gx_widget_previous_sibling_get</code></td>
<td>779</td>
</tr>
<tr>
<td><code>gx_widget_resize</code></td>
<td>781</td>
</tr>
<tr>
<td><code>gx_widget_shift</code></td>
<td>783</td>
</tr>
<tr>
<td><code>gx_widget_show</code></td>
<td>785</td>
</tr>
<tr>
<td><code>gx_widget_status_add</code></td>
<td>786</td>
</tr>
<tr>
<td><code>gx_widget_status_get</code></td>
<td>787</td>
</tr>
<tr>
<td><code>gx_widget_status_remove</code></td>
<td>789</td>
</tr>
<tr>
<td><code>gx_widget_status_test</code></td>
<td>790</td>
</tr>
<tr>
<td><code>gx_widget_string_get</code></td>
<td>791</td>
</tr>
<tr>
<td><code>gx_widget_string_get_ext</code></td>
<td>792</td>
</tr>
<tr>
<td><code>gx_widget_style_add</code></td>
<td>793</td>
</tr>
<tr>
<td><code>gx_widget_style_get</code></td>
<td>795</td>
</tr>
<tr>
<td><code>gx_widget_style_remove</code></td>
<td>797</td>
</tr>
<tr>
<td><code>gx_widget_style_set</code></td>
<td>799</td>
</tr>
<tr>
<td><code>gx_widget_text_blend</code></td>
<td>801</td>
</tr>
<tr>
<td><code>gx_widget_text_blend_ext</code></td>
<td>803</td>
</tr>
<tr>
<td><code>gx_widget_text_draw</code></td>
<td>805</td>
</tr>
<tr>
<td><code>gx_widget_text_draw_ext</code></td>
<td>806</td>
</tr>
<tr>
<td><code>gx_widget_text_id_draw</code></td>
<td>807</td>
</tr>
<tr>
<td><code>gx_widget_top_visible_child_find</code></td>
<td>808</td>
</tr>
<tr>
<td><code>gx_widget_type_find</code></td>
<td>809</td>
</tr>
<tr>
<td><code>gx_widget_width_get</code></td>
<td>811</td>
</tr>
</tbody>
</table>
gx_window_client_height_get ................................................................. 813
gx_window_client_scroll ................................................................. 814
gx_window_client_width_get .............................................................. 815
gx_window_close ................................................................................. 816
gx_window_create ................................................................................ 817
gx_window_draw .................................................................................... 819
gx_window_event_process ..................................................................... 820
gx_window_execute ................................................................................ 822
gx_window_root_create ................................................................. 824
gx_window_root_delete ........................................................................ 825
gx_window_root_event_process ............................................................ 826
gx_window_root_find ............................................................................ 828
gx_window_scroll_info_get ................................................................. 829
gx_window_scrollbar_find .................................................................... 831
gx_window_wallpaper_get ...................................................................... 832
gx_window_wallpaper_set ....................................................................... 833
Chapter 5: GUIX Display Drivers ...................................................... 835
GUIX Example ....................................................................................... 845
Appendix A: GUIX Color Definitions .................................................. 849
Appendix B: GUIX Color Formats ....................................................... 851
Appendix C: GUIX Widget Styles ......................................................... 852
Appendix D: GUIX Brush, Canvas and Gradient Attributes .................. 860
Appendix E: GUIX Event Description .................................................... 862
Appendix F: GUIX RTOS Binding Services .......................................... 869
Appendix G: GUIX Font Structure ......................................................... 872
Appendix H: GUIX Build-Time Configuration flags ............................... 875
Appendix I: GUIX Information Structures ........................................... 882
  GX_CIRCULAR_GAUGE_INFO ................................................................. 882
  GX_LINE_CHART_INFO ......................................................................... 884
  GX_MOUSE_CURSOR_INFO .................................................................... 886
  GX_PEN_CONFIGURATION ................................................................. 887
  GX_PIXELMAP_SLIDER_INFO ............................................................... 888
  GX_PROGRESS_BAR_INFO ................................................................. 889
  GX_RADIAL_PROGRESS_BAR_INFO ..................................................... 890

GUIX User Guide
GX_RADIAL_SLIDER_INFO ................................................................. 892
GX_RECTANGLE ........................................................................... 893
GX_SCROLL_INFO ....................................................................... 894
GX_SCROLLBAR_APPEARANCE .................................................. 895
GX_SLIDER_INFO ....................................................................... 897
GX_SPRITE_FRAME ..................................................................... 898
Index ........................................................................................... 899
About This Guide

This guide contains comprehensive information about GUIX, the high-performance GUI product from Express Logic, Inc. It is intended for embedded real-time software developers familiar with basic GUI concepts, the ThreadX RTOS, and the C programming language.

Organization

Chapter 1  Introduces GUIX
Chapter 2  Gives the basic steps to install and use GUIX with your ThreadX application
Chapter 3  Provides a functional overview of GUIX
Chapter 4  Details the application’s interface to GUIX.
Chapter 5  Describes display drivers for GUIX.
Index  Topic cross reference
Guide Conventions

*Italics*  
Typeface denotes book titles, emphasizes important words, and indicates variables.

*Boldface*  
Typeface denotes file names, key words, and further emphasizes important words and variables.

ℹ️  
Information symbols draw attention to important or additional information that could affect performance or function.
GUIX Data Types

In addition to the custom GUIX control structure data types, there are several special data types that are used in GUIX service call interfaces. These special data types map directly to data types of the underlying C compiler. This is done to ensure portability between different C compilers. The exact implementation is inherited from ThreadX and can be found in the `tx_port.h` file included in the ThreadX distribution.

The following is a list of GUIX service call data types and their associated meanings:

- **UINT**
  - Basic unsigned integer. This type is mapped to the most convenient unsigned data type.

- **INT**
  - Basic signed integer. This type is mapped to the most convenient signed data type.

- **ULONG**
  - Unsigned long type. This type must support 32-bit unsigned data.

- **VOID**
  - Almost always equivalent to the compiler’s void type.

- **GX_CHAR**
  - Most often typedefed as the compiler defined char type.

- **GX_BYTE**
  - 8-bit signed type.

- **GX_UBYTE**
  - 8-bit unsigned type.

- **GX_VALUE**
  - 16 or 32 bit signed type. Defined as needed for best performance on the target system.

- **GX_FIXED_Val**
  - Fixed point numeric data type.

- **GX_RESOURCE_ID**
  - Unsigned long type.

- **GX_COLOR**
  - Unsigned long type.

- **GX_STRING**
  - Structure containing GX_CHAR *gx_string_ptr and UINT gx_string_length.

- **GX_POINT**
  - Structure containing gx_point_x and gx_point_y.

- **GX_RECTANGLE**
  - Structure containing gx_rectangle_left, gx_rectangle_top, gx_rectangle_right, and gx_rectangle_bottom fields.

- **GX_GLYPH**
  - Structure containing glyph metrics.

- **GX_FONT**
  - Structure containing font metrics.

- **GX_BRUSH**
  - Structure containing brush metrics.
GX_PIXELMAP Structure containing pixelmap metrics.

Additional data types are used within the GUIX source. They are located in either the `tx_port.h` or `gx_port.h` files.
Customer Support Center

Support engineers 858.613.6640
Support fax 858.521.4259
Support email support@expresslogic.com
Web page http://www.expresslogic.com

Latest Product Information

Visit the Express Logic web site and select the “Support” menu option to find the latest online support information, including information about the latest GUIX product releases.

What We Need From You

To more efficiently resolve your support request, provide us with the following information in your email request:

1. A detailed description of the problem, including frequency of occurrence and whether it can be reliably reproduced.
2. A detailed description of any changes to the application and/or GUIX that preceded the problem.
3. The contents of the _tx_version_id and _gx_version_id strings found in the tx_port.h and gx_port.h files of your distribution. These strings will provide us valuable information regarding your run-time environment.
4. The contents in RAM of the following ULONG variables:
   _tx_build_options
   _gx_system_build_options
   These variables will give us information on how your ThreadX and GUIX libraries were built.
5. The contents in RAM of the following ULONG variables:
   _gx_system_last_error
   _gx_system_error_count
   These variables keep track of internal system errors in GUIX. If the _gx_system_error_count is greater than one, please set a breakpoint on the function return in the _gx_system_error_process function and provide the value of _gx_system_last_error at this point. This will yield the first internal GUIX system error.
6. A trace buffer captured immediately after the problem was detected. This is accomplished by building the ThreadX and GUIX libraries with TX_ENABLE_EVENT_TRACE and calling tx_trace_enable with the trace buffer information. Refer to the TraceX User Guide for details.
7. The GUIX Studio project you are using, if applicable, or at a minimum a small project sufficient to demonstrate the deficiency you are reporting.
Where to Send Comments About This Guide

The staff at Express Logic is always striving to provide you with better products. To help us achieve this goal, email any comments and suggestions to the Customer Support Center at support@expresslogic.com

Please enter “GUIX User Guide” in the subject line.
Chapter 1: Introduction to GUIX

GUIX is a high-performance real-time implementation of a (GUI) designed exclusively for embedded ThreadX-based applications. This chapter contains an introduction to GUIX and a description of its applications and benefits.

GUIX Feature Overview

- ANSI C Source Code
- Not A Black Box

Embedded GUI Applications

- Real-time GUI Software

GUIX Benefits

- Improved Responsiveness
- Software Maintenance
- Increased Throughput
- Processor Isolation
- Ease of Use
- Improve Time to Market
- Protecting the Software Investment
GUIX Feature Overview

Unlike many other GUI implementations, GUIX is designed to be versatile—easily scaling from small micro-controller-based applications to those that use powerful RISC and DSP processors. This is in sharp contrast to public domain or other commercial implementations originally intended for workstation environments but then squeezed into embedded designs. An overview of GUIX features follows:

- Easy to use with host-based design tool GUIX Studio
- Win32 GUIX run-time environment for complete hosted prototyping
- Supports most processors supported by ThreadX
- Written exclusively in ANSI C
- Endian neutral
- Smallest, Fasted Embedded GUI
- Run-time configurable, number of objects, screen size, etc.
- Easy to write display driver interface
- Color (up to 32-bpp color depth), monochrome, and grayscale support
- Multilingual support via UTF8 string encoding and string resources
- Default free fonts and easy to add new fonts
- Multiple drawing Canvases supported, of various sizes
- Multiple displays of different sizes and color depths supported
- Screen Transition support (fade in, fade out, swipe, etc.)
- Touch Screen, Gesture, and Virtual Keyboard Support
- Bitmap compression
- Alpha Blending Support
- Dither Support
- Anti-Aliasing Support
- Skinning and Themes
- Canvas Blending
- Complete Window Management
  - Parent/Child Relationship
  - Dynamic creation, deletion, resizing, moving
  - Separate event handling and drawing
  - Z-order
  - Clipping and views
- Extensive Set of Widgets
  - Various button types, sliders, and dials
  - Drop Down List
  - Prompt
  - Multi-Line text view
  - Single and Multi-Line text input
  - Numeric and Textual Scroll Wheels
  - Windows and Scroll Bars
  - Radial Progress Bar
  - Sprite
ANSI C Source Code

GUIX is written completely in ANSI C and is portable immediately to virtually any processor architecture that has an ANSI C compiler and ThreadX support. Although written in ANSI C, GUIX uses an object oriented model and inheritance.

Not A Black Box

Most distributions of GUIX include the complete C source code. This eliminates the “black-box” problems that occur with many commercial GUI implementations. By using GUIX, applications developers can see exactly what the GUI is doing—there are no mysteries!

Having the source code also allows for application specific modifications. Although not recommended, it is certainly beneficial to have the ability to modify the GUI if it is required. These features are especially comforting to developers accustomed to working with in-house or public domain products. They expect to have source code and the ability to modify it. GUIX is the ultimate GUI software for such developers.

Embedded GUI Applications

Embedded GUI applications are applications that have a user interface requirement and execute on microprocessors hidden inside products such as cellular phones, communication equipment, automotive engines, laser printers, medical devices, and so forth. Such applications almost always have some memory and performance constraints. Another distinction of embedded GUI is that their software and hardware have a dedicated purpose.

Real-time GUI Software

Basically, GUI software that must perform its processing within an exact period of time is called real-time GUI software, and when time constraints are imposed on GUI applications, they are classified as real-time applications. Embedded GUI applications are almost always real-time because of their inherent interaction with the external world.
GUIX Benefits

The primary benefits of using GUIX for embedded applications are high-performance, feature rich, and very small memory requirements. GUIX is also completely integrated with the high-performance, multitasking ThreadX real-time operating system.

Improved Responsiveness
The high-performance GUIX product enables applications to respond faster than ever before. This is especially important for embedded applications that either have a significant volume of visual information or strict timing requirements on displaying such information.

Software Maintenance
Using GUIX allows developers to easily partition the GUI aspects of their embedded application. This partitioning makes the entire development process easy and significantly enhances future software maintenance.

Increased Throughput
GUIX provides the highest-performance GUI available, which directly transfers to the embedded application. GUIX applications are able to process user interface information faster than non-GUIX applications!

Processor Isolation
GUIX provides a robust, processor-independent interface between the application and the underlying processor and display hardware. This allows developers to concentrate on the high-level aspects of the user interface rather than spending extra time dealing with display hardware issues.

Ease of Use
GUIX is designed with the application developer in mind. The GUIX architecture and service call interface are easy to understand. As a result, GUIX developers can quickly use its advanced features.

Improve Time to Market
The powerful features of GUIX accelerate the software development process. GUIX abstracts most processor and display hardware issues, thereby removing these concerns from a majority of application user interface implementation. This feature, coupled with the ease-of-use and advanced feature set, results in a faster time to market!
Protecting the Software Investment
GUIX is written exclusively in ANSI C and is fully integrated with the ThreadX real-time operating system. This means GUIX applications are instantly portable to all ThreadX supported processors. Better yet, a completely new processor architecture can be supported with ThreadX in a matter of weeks. As a result, using GUIX ensures the application’s migration path and protects the original development investment.
Chapter 2: Installation and Use of GUIX

This chapter contains a description of various issues related to installation, setup, and use of the high-performance user interface product GUIX, including the following:

- Host Considerations
- Target Considerations
- Product Distribution
- GUIX Installation
- Using GUIX
- Troubleshooting
- Configuration Options
- GUIX Version ID
Host Considerations

Embedded development is usually performed on Windows or Linux (Unix) host computers. After the application is compiled, linked, and the executable is generated on the host, it is downloaded to the target hardware for execution.

Usually the target download is done from within the development tool's debugger. After download, the debugger is responsible for providing target execution control (go, halt, breakpoint, etc.) as well as access to memory and processor registers.

Most development tool debuggers communicate with the target hardware via on-chip debug (OCD) connections such as JTAG (IEEE 1149.1) and Background Debug Mode (BDM). Debuggers also communicate with target hardware through In-Circuit Emulation (ICE) connections. Both OCD and ICE connections provide robust solutions with minimal intrusion on the target resident software.

As for resources used on the host, the source code for GUXI is delivered in ASCII format and requires approximately 30 Mbytes of space on the host computer's hard disk.

*Review the supplied readme_guix_generic.txt file for additional host system considerations and options.*

Target Considerations

GUIX requires between 5 KBytes and 80 Kbytes of Read-Only Memory (ROM) on the target. Another 5 to 10KBytes of the target’s Random Access Memory (RAM) are required for the GUIX thread stack and other global data structures.

In addition, GUIX requires the use of a ThreadX timer and a ThreadX mutex object. These facilities are used for periodic processing needs and thread protection inside GUIX.

Product Distribution

GUIX is normally delivered as a package including complete GUIX source code. For certain hardware targets, binary distributions of the GUIX library are provided by the silicon vendor, and in these cases you may not have access to the GUIX source code unless you purchase an upgraded source code distribution. The following is a list of the important files common to most product distributions:

- **readme_guix_generic.txt**: This file contains specific information about the GUIX release.
- **gx_api.h**: This C header file contains all system equates, data structures, and service prototypes.
- **gx_port.h**: This C header file contains all target-specific and development tool-specific data definitions and structures.
- **gx.a (or gx.lib)**: This is the binary version of the GUIX C library. This is normally built by compiling and archiving the provided GUIX library source files, however this library may be provided in pre-built form depending on your hardware target and license type.
All files are in lower-case, making it easy to convert the commands to Linux (Unix) development platforms.

GUIX Installation

Installation of GUIX is straightforward. The following instructions apply to virtually any installation. However, please examine the readme_guix.txt file for changes specific to the actual development tool environment.

Step 1: Backup the GUIX distribution disk and store it in a safe location.

Step 2: On the host hard drive, copy all the files of the GUIX distribution into the previously created and installed ThreadX directory.

Step 3: GUIX is normally distributed as a collection of C source files and corresponding .h header files which are compiled and archived into the GUIX library gx.a (or gx.lib). Your distribution will usually include a project file or makefile specific to the compiler or tools you are using for building the GUIX library.

Application software needs access to the GUIX library file, usually called gx.a (or gx.lib), and the C include files gx_api.h and gx_port.h. This is accomplished either by setting the appropriate path for the development tools or by copying these files into the application development area.

Using GUIX

Using GUIX is easy. Basically, the application code must include gx_api.h during compilation and link with the GUIX library gx.a (or gx.lib).

There are four easy steps required to build a GUIX application:

Step 1: Include the gx_api.h file in all application files that use GUIX services or data structures.

Step 2: Initialize the GUIX system by calling gx_system_initialize from the tx_application_define function or an application thread.

Step 3: Create a display instance, create a canvas for the display, and create the root window and any other windows or widgets necessary.

Step 4: Compile application source and link with the GUIX runtime library gx.a (or gx.lib). The resulting image can be downloaded to the target and executed.

Troubleshooting

Each GUIX port is delivered with a demonstration application that executes on specific display hardware. The same basic demonstration is delivered with all versions of GUIX. It is always a good idea to get the demonstration system running first.
See the `readme_guix_generic.txt` file supplied with the distribution for more specific details regarding the demonstration system.
If the demonstration system does not run properly, perform the following operations to narrow the problem:

1. Determine how much of the demonstration is running.
2. Increase the stack size of the GUIX thread by changing the compile-time constant `GX_THREAD_STACK_SIZE` and recompiling the GUIX library.
3. Recompile the GUIX library with the appropriate debug options listed in the configuration option section.
4. Examine the return status from all API calls.
5. Determine if there is an internal system error by setting a breakpoint at the function `__gx_system_error_process`. There error code and caller should give clues as to what might be going wrong.
6. Temporarily bypass any recent changes to see if the problem disappears or changes. Such information should prove useful to Express Logic support engineers.

Follow the procedures outlined in the section titled “What We Need From You” to send the information gathered from the troubleshooting steps.

**Configuration Options**

There are several configuration options when building the GUIX library and the application using GUIX. These options are used to tune the library size and feature set to best fit your application requirements. For example, if your application will have only one thread utilizing the GUIX API services, the configuration flag `GX_DISABLE_MULTITHREAD_SUPPORT` should be defined to eliminate the overhead associated with protecting critical code sections from pre-emption by multiple threads. The various configuration flags can be defined in the application source, on the command line, or within the `gx_user.h` include file.

Whenever the GUIX library configuration flags are modified, it is required to rebuild both the GUIX library and your application modules for the configuration changes to take effect.

The complete list of configuration flags is documented in Appendix H: GUIX Build-Time Configuration Flags.

**GUIX Version ID**

The current version of GUIX is available to both the user and the application software during runtime. The programmer can find the GUIX version in the `readme_guix_generic.txt` file. This file also contains a version history of the corresponding port. Application software can obtain the GUIX version by examining the global string `__gx_version_id` in `gx_port.h`.

Application software can also obtain release information from the constants shown below defined in `gx_api.h`. These constants identify the current product release by name and the product major and minor version.

```c
#define __PRODUCT_GUIX__
#define __GUIX_MAJOR_VERSION__
#define __GUIX_MINOR_VERSION__
```
Chapter 3: Functional Overview of GUIX

This chapter contains a functional overview of the high-performance GUIX user interface product.

Execution Overview
- Initialization
- Application Interface Calls
- Internal GUIX Thread
  - Event Processing
  - Drawing
- User Input
- Modal Dialog Execution
- Periodic Processing
- Display Driver
- Memory Usage
  - Static Memory Usage
  - Dynamic Memory Usage

GUIX Components

GUIX System Component
- Initialization
- Thread Processing
- RTOS Binding Layer
- Multithread Safety
- Periodic Processing
- Pen Speed Configuration
- Widget Defaults
- Screen Stack
- Clipboard Maintenance
- Dirty List Maintenance
- Animation Control Block Pool
- Scrollbar Appearance
- Skinning
- System Error Handling

GUIX Canvas Component
- Canvas Creation
- Canvas Control Block
- Canvas Alpha Channel
- Color Depth
- Transitions
- Drawing APIs
GUIX Display Component
  Display Creation
  Display Control Block
  Installing Themes
  Root Window
  Anti-Aliasing
  Clipping
  Views
  Display Driver Interface

GUIX Widget Component
  Widget Creation
  Widget Control Block
  Hierarchy
  Types
  Styles
  Background
  Event Notification
  Event Processing
  Drawing Function

GUIX Drawing Context Component
  Context Creation
  Context Brush
  Context Font
  Context Colors
  Context Pixelmaps

GUIX Window Component
  Window Creation
  Window Control Block
  Root Window
  Background
  Scrolling
  Event Notification
  Event Processing
  Drawing Function

GUIX Utility Component
  Working with Rectangles
  Defining a brush
  Converting numbers to strings
  Mathematical operations
  Manipulating Pixelmaps
  Rendering text to an alphamap
GUIX Animation Component
  Timer-driven fade and slide animations
  Pen-driven slide animations
Execution Overview

GUIX implements an event driven programming model. This means that the GUIX framework is primarily driven by the receipt of events pushed into the GUIX event queue. The processing of these events takes place in the context of the GUIX thread, which is a ThreadX thread created during GUIX system initialization.

GUIX applications define the user interface by calling GUIX API functions to create windows and child widgets, and customize the appearance of these widgets by calling additional API functions used to define colors, styles, fonts, and various other attributes of each window or widget type. If you are using GUIX Studio to create the appearance of your user-interface screens, much of this work of calling GUIX API functions to create your display is done for you by the GUIX Studio application.

GUIX applications interact with the system user and with external business logic by handling events retrieved from the GUIX event queue. These events are usually produced by GUIX widgets, but they can also be created by external threads. When a typical GUIX button is pushed, that button sends an event to the button’s parent window. Your application program will act on that button push by providing a handler for the button push event.

Additional GUIX threads are often created for things such as input drivers. A typical touch screen input driver is executed as a standalone thread external to the main GUIX thread. The touch input driver sends touch information into the GUIX thread by sending events into the GUIX event queue.

Since many user-interface operations such as animations require accurate timing information, GUIX also implements a simple and easy to use timer interface. This timer interface is built upon the ThreadX timer service, and is configured automatically at system startup.

The vast majority of the GUIX software is independent of any hardware dependencies. The framework does require hardware-specific input drivers and hardware-specific graphics drivers. The details of how these hardware specific drivers are implemented are deferred to chapter 5.

Initialization

The service `gx_system_initialize` must be called before any other GUIX service is called. GUIX system initialization can be called from the ThreadX `tx_application_define` routine (initialization context) or from application threads. The `gx_system_initialize` function creates the GUIX event queue, initializes the GUIX timer facility, creates the main GUIX system thread, and initializes various data structures maintained by GUIX during the execution of your application.
After `gx_system_initialize` returns, the application is ready to create displays, canvases, windows, widgets, and customize the properties of all GUIX objects. Much of the GUIX object creation API can be called from `tx_application_define` or from application threads.

## Application Interface Calls

Calls from the application are largely made from `tx_application_define` (initialization context) or from application threads. Please see the “Allowed From” section of each GUIX API described in Chapter 4 to determine what context it may be called from.

For the most part, processing intensive activities are deferred to the internal GUIX thread, including all event processing and widget/window drawing.

The GUIX API functions can be called from any thread at any time. However it is usually considered to be better architecture to separate your time-critical business logic from your user interface logic. Since the user interface drawing operations can sometimes take a long time depending on your display size and CPU performance, you normally would not want to have time-critical threads delayed waiting for a drawing operation to complete.

## Internal GUIX Thread

As mentioned, GUIX has an internal thread that performs the bulk of the GUI processing. This thread is created by the application software by calling `gx_system_initialize()` followed by `gx_system_start()`.

The priority of the internal GUIX thread is determined by the `#define GX_SYSTEM_THREAD_PRIORITY`. This value defaults to 16 (middle priority) but can be modified by specifying this value in the `gx_port.h` or `gx_user.h` header file, overriding the default value.

The GUIX thread time slice is similarly defined by the `#define GX_SYSTEM_THREAD_TIMESLICE`, which defaults to the value 10 ms.

The stack size of the system thread is determined by the `#define GX_THREAD_STACK_SIZE`, which is found in the `gx_port.h` header file, but can also be overridden by specifying this value in your `gx_user.h` header file.

The internal GUIX thread execution loop is composed of three actions. First, GUIX retrieves events from the GUIX event queue and dispatches those events for processing by the GUIX windows and widgets. Events are typically pushed into the GUIX event queue by periodic timers, input devices such as a touch screen or keypad, and by GUIX widgets themselves as they process user input. Next, after all events have been processed, GUIX determines if a screen refresh is needed, and if so performs the
processing necessary to update the display graphics data, mainly by calling the drawing functions of those windows and widgets which have been marked as dirty. Finally, GUIX suspends the GUIX thread until a new input event or events arrive.
Event Processing

Touch or pen input events are processed by determining the top-most window or widget beneath the touch or pen input pixel position and calling that window/widget’s event processing function. If the widget understands pen input events, it will process the event as appropriate for that widget type. If not, the top-most widget will pass the touch or pen input event to the widget’s parent for processing. This passing of the event up the chain continues until either the event is handled or the event arrives at the root window, in which case the event is discarded.

Keypad events are sent to the window/widget that has input focus. Input focus status is maintained by the GUIX gx_system component.

Timer events are always dispatched to the window or widget that owns the timer for processing.

Internally generated events, such as button click events or slider value change events, are always sent to the parent of the widget generating the event. If the parent does not process the event, it is passed up the chain similar to touch or pen input events.

Drawing

Once all the event processing is complete, the GUIX internal thread determines if any display update is needed and if so the appropriate window/widget drawing functions are called. When drawing is complete, the GUIX internal thread simply waits on its event queue for the next GUIX event to process.

GUIX implements the concept of “dirty areas” for each widget and canvas. A widget can only draw to areas that have previously been marked as dirty. When a widget drawing function is called, all drawing operations are internally clipped to the previously defined dirty rectangle. Attempts to draw outside of this area are ignored.

Widgets and windows mark themselves as dirty by calling the API function gx_system_dirty_mark. This function marks the entire widget or window as needing to be redrawn. A second function, gx_system_dirty_partial_add, can be invoked as an alternative to mark only a portion of a window or widget as dirty.

This model of marking widgets as dirty, or needing to be re-drawn, and then redrawing those widgets only when all input events have been processed is referred to as deferred drawing. The GUIX deferred drawing algorithm and dirty list maintenance is designed to improve drawing efficiency. Since drawing operations are typically expensive, GUIX works hard to prevent unnecessary drawing.

Drawing is done to a GUIX canvas. A canvas is a memory area reserved to hold graphics data. A canvas may or may not be directly linked to the hardware frame buffer,
depending on the system architecture and memory constraints. Before any drawing can occur, a canvas must first be opened for drawing by calling the gx_canvas_drawing_initiate() API function. This API prepares a canvas for drawing and established the current drawing context. When GUIX performs a system canvas refresh, the canvas is opened for drawing and the drawing context established before the widget-level drawing APIs are invoked. Therefore widgets do not need to initiate drawing on a canvas within the widget drawing function.

However, if an application desires to perform immediate drawing to a canvas, outside the flow of the standard GUIX deferred drawing algorithm, the application must directly invoke the gx_canvas_drawing_initiate() prior to calling any other drawing APIs, and must call gx_canvas_drawing_complete() once the immediate drawing has been completed.
**User Input**

GUIX supports touch screen, mouse, and keyboard devices with pre-defined event types. Additional input devices can be utilized by defining custom event types, or by mapping the custom input device to the pre-defined event types.

Actions associated with these devices are translated into events that are processed by the internal GUIX thread. Driver level software written to support a touch screen must prepare and send to the GUIX event queue events for pen-down, pen-up, and pen-drag operations. Similarly a keypad input driver must generate events for key press and key release input.

**Modal Dialog Execution**

Modal dialog execution refers to presenting a window to the user that must be closed in some way before any other GUIX windows or widgets can receive user input. Modal dialogs capture all user input while the dialog window is displayed, regardless of the x,y position of touch or mouse input events.

Modal dialogs are triggered by the application software by first creating the window in the normal way by calling `gx_window_create`, and then calling the GUIX API function `gx_window_execute`.

When the `gx_window_execute` function is called, GUIX enters a local event processing loop. The `gx_window_execute` function does not return to the caller until the dialog window is closed, either by user input or by calling `gx_window_close`. For this reason, it is very important never to call the `gx_window_execute` function from any thread other than the GUIX internal thread.
Periodic Processing

In order to provide display effects, sprite animation, and support for application periodic requests, GUIX uses one ThreadX timer. This single timer is used to drive all GUIX time-related needs. By default, the frequency for the GUIX internal timer processing is set to 20ms via the constant `GX_SYSTEM_TIMER_MS`, which is defined in `gx_api.h`, unless the constant is previously defined in `gx_port.h` or `gx_user.h` header. The default frequency may be changed by the application via a compilation option when building the GUIX library or by explicitly redefining it in `gx_user.h`.

Note that the GUIX timer frequency is expressed in RTOS timer ticks, and is defined by the constant `GX_SYSTEM_TIMER_TICKS`. The value of `GX_SYSTEM_TIMER_TICKS` is calculated using `GX_SYSTEM_TIMER_MS` and `TX_TIMER_TICKS_PER_SECOND`. The user can re-define any of these values in the `gx_port.h` or `gx_user.h` to adjust the GUIX timer frequency and resolution.

Display Driver

Display drivers are responsible for providing a set of drawing functions to the core GUIX code. The implementation of each of these drawing functions is determined by the driver, and when possible the implementation will leverage hardware acceleration support. In general the drawing function works by writing pixel data to a memory buffer, which may be the physical frame buffer or it may be a secondary buffer depending on the driver architecture. Many drivers implement double buffering using two frame buffers, and these buffers are toggled by invoking the buffer toggle function. GUIX calls these functions internally at the appropriate times. For memory constrained systems, the drawing functions may only write to a single memory frame buffer.

GUIX provides default software implementations of each low-level drawing function at every support color depth and format. These functions are invoked via function pointers maintained within the `GX_DISPLAY` structure. When hardware-specific drivers are created, they typically will overwrite some number of these function pointers with functions that are specific to the target hardware.

A typical hardware display driver is implemented by first creating the default GUIX display driver for the required color depth and format. Then the hardware driver will replace those functions that need to be optimized or customized for the particular hardware implementation.

GUIX support pixel color formats ranging from 1-bpp monochrome to 32-bpp a:r:g:b format. GUIX also supports many variations within each broad color-depth category, such as r:g:b versus b:g:r byte order, packed pixel versus word-aligned pixel formats, and alpha channels. There are currently 25 distinct color formats supported, but this list grows as hardware vendors deliver new variations.
Display Memory Architectures

Various hardware targets and displays utilize a variety of different display memory architectures, depending on the memory constraints of the target and the functionality requirements of the application. We will outline some of the common memory architectures here with a brief description of each.

Model 1) No frame buffer, graphics data held in external GRAM:

In the model above, no memory for a frame buffer exists in memory local to the CPU. All graphics data is stored in an external GRAM which is incorporated into the display itself. The interface to the external GRAM can be parallel or serial. This type of architecture is very low cost; however it can exhibit unwanted tearing effect when the graphics data is updated.

Model 2) One local frame buffer:

In this model, memory for the graphics data is allocated from a random-access memory that is directly accessible the CPU. Dedicated hardware must be present to repeatedly transmit the graphics data (along with timing signals) from the local memory to the display. This model differs from model 1 in that the graphics memory is a block of the local SRAM or DRAM available to the CPU. This may be the same memory in which stack and program variables live.

Model 3) Local frame buffer + external GRAM:

Model 3 is a combination of the first two. In this model, sufficient local memory exists to hold one frame buffer. In addition, the display device provides an external GRAM and automatically refreshes itself using the data provided in the GRAM. This architecture benefits from improved update efficiency because we can transfer the modified portion
of the local frame buffer to the external GRAM in one block transfer, often utilizing on-board DMA channels. This model also eliminates the tearing and flicker that can be present in either of the first two models, because only completed graphics contents is copied to the external GRAM.

Model 4) Ping-pong frame buffers:

In model 4, sufficient memory is present to provide two local frame buffers. In this case, GUIX treats one frame buffer as the active frame buffer, and the other as the working frame buffer. When a display update or drawing operation is in progress, it takes place in the working buffer. When the drawing operation completes, the buffers are toggled, and the working buffer becomes the active buffer and the active buffer becomes the working buffer. This model also eliminates screen flicker and tearing that can be observed in a single buffered system.

Model 5) Ping-pong buffers with canvas compositing:
In model 5, any number of canvases can be created, up to the limits of available memory. The canvases can be overlaid or blended together as defined by the application to create the canvas composite. When a new composite is created after a screen refresh operation, the active and working composite buffers are toggled in an operation identical to the standard ping-pong buffer architecture. Model 5 adds the ability to perform screen fade and blending operations by blending the canvases into the final output composite.

Model 6) Canvas compositing with external GRAM:

Model 6 is a slight variation on Model 5, in which only one composite buffer is required and the composite buffer is then transferred to external GRAM. This model also supports full screen blending and overlays.
String Encoding

GUIX by default supports UTF8 format string encoding. Support for UTF8 string encoding can be disabled by defining GX_DISABLE_UTF8_SUPPORT in the gx_user.h header file. If UTF8 encoding is disabled, GUIX will internally use only standard 8-bit ASCII plus Latin-1 code page character encoding. Disabling UTF8 string encoding results in a slightly smaller GUIX library footprint and slightly faster runtime execution of string handling and text drawing functions.

UTF8 string encoding has the following traits:

- ASCII strings take no more storage space than standard 7-bit ASCII encoding.
- Most ANSI-C string functions work with UTF8 string encoding without modification.

All active character sets in the world, including Kanji character sets, can be represented using UTF8 string encoding.

Static and Dynamic Strings

The strings assigned to your GUIX widgets which support text display can be statically defined string constants, which are normally placed in constant storage as part of the GUIX String table described below, and dynamically defined strings, which are strings generated at runtime using services such as sprintf() or gx_utility_ltoa().

Examples of dynamic strings might include a value displayed as a number within a GUIX prompt widget, or a “time / date” string which is dynamically formatted based on the user’s location and format preferences. If you create strings at runtime which will be assigned to GUIX widgets such as GX_PROMPT or GX_TEXT_BUTTON widgets, you must choose to either statically allocate the storage for these runtime generated strings (i.e global character arrays), or you can define and install a dynamic memory allocator function and use the GX_STYLE_TEXT_COPY style, which instructs those widgets to create a private copy of text strings assigned.

It is a programming error to use temporary storage, such as an automatic character array, to hold a dynamically generated string and then assign this string to a widget that does not have the GX_STYLE_TEXT_COPY style. When this style is not enabled, the widget simply copies the provided string pointer, and the string data must be statically allocated or the widget string pointer will likely end up pointing at garbage data producing unpredictable results.
Passing GX_STRING arguments

The GUIX API functions which accept a GX_STRING parameter always verify that the length of the string pointed to by the GX_STRING.gx_string_ptr field match the value of the GX_STRING.gx_string_length field. If the two fields are not consistent, a GX_INVALID_STRING_LENGTH error is returned and the API called returns without accepting the string assignment.

For safety considerations the GUIX software never internally uses the standard C string functions such as strlen or strcpy. These functions have been known to be susceptible to malicious attacks when string data is acquired dynamically which is often the case with connected applications.

GUIX library releases prior to release 5.6 defined API functions which accepted (GX_CONST GX_CHAR *text) as a parameter. These functions, while still supported for backwards compatibility, have been obsoleted and replaced by the preferred API functions which accept (GX_CONST GX_STRING *string) as an input parameter.

By default, the deprecated text handling API is enabled allowing all previously written applications to build cleanly with the latest updates to the GUIX library. To disable the the deprecated text handling API, the definition “GX_DISABLE_DEPRECATED_STRING_API” should be added to the gx_user.h header file. All new applications should define GX_DISABLE_DEPRECATED_STRING_API and should use only the replacement API functions. All output files generated by GUIX Studio for GUIX library version >= release 5.6 will utilize only the replacement API functions.

The following table lists the deprecated and newly defined replacement API function names:

<table>
<thead>
<tr>
<th>Deprecated Function Name</th>
<th>Replaced With</th>
</tr>
</thead>
<tbody>
<tr>
<td>gx_binres_language_table_load</td>
<td>gx_binres_language_table_load_ext</td>
</tr>
<tr>
<td>gx_canvas_rotated_text_draw</td>
<td>gx_canvas_rotated_text_draw_ext</td>
</tr>
<tr>
<td>gx_canvas_text_draw</td>
<td>gx_canvas_text_draw_ext</td>
</tr>
<tr>
<td>gx_context_string_get</td>
<td>gx_context_string_get_ext</td>
</tr>
<tr>
<td>gx_display_language_table_get</td>
<td>gx_display_language_table_get_ext</td>
</tr>
<tr>
<td>gx_display_language_table_set</td>
<td>gx_display_language_table_set_ext</td>
</tr>
</tbody>
</table>
### GUIX String Table

The GUIX string table and string resources are registered with a GUIX display instance. Each display in a multi-display system has its own string table, and each display can run in its own selected language. The other GUIX resource types (colors, fonts, and pixelmaps) are also maintained by the GUIX Display component, since these resource types are specific to each display color format and color depth.

While you can manually create your application string table, most often the display string table is defined by the GUIX Studio application as part of your project resource.
file. The available languages are also defined in the resource header file. The display string table is a multi-column table of pointers to application strings. Each column of the string table represents one language supported by the application. If your application supports only one language, for example English, then your string table will have only one column. Still, you can add support for additional languages at any time without modifying your application software.

The active string table is assigned by calling the gx_display_string_table_set() API function. This function is called automatically by the GUIX Studio generated startup code, but can also be called directly by the application to change the active string table.

The active language is assigned by calling the gx_display_active_language_set() API function. This function determines which column of the display string table is active. When this function is invoked, a GX_EVENT_LANGUAGE_CHANGE event is sent to all visible GUIX widgets, allowing them to update to display the newly active string data.

Widgets and application software resolve statically defined strings using string ID values and the gx_display_string_get_ext() or gx_widget_string_get_ext() API functions. These functions return the GX_STRING associated with a given string ID and the currently active language.

**Bi-directional Text Display**

GUIX provide two strategies for bi-directional text support.

One option is to do bidi text reordering within the GUIX Studio application. Using this option GUIX Studio is responsible for generating bidi text to the output file in its display order. This solution has zero impact on the runtime performance and does not require any additions to the GUIX runtime library. To allow GUIX Studio to generate display-order bidi text strings, you should select the “Generate Bidi Text in Display Order” checkbox in the GUIX Studio language configuration dialog:
With these options selected, the generated resource file will contain Bidi strings generated in display order, and no extra processing is required within the GUIX runtime library.

The second option is to do bidi text reordering at runtime. This option is supported for those applications that must handle bidi text string that are dynamically defined, and not generated by the GUIX Studio application. In this case the GUIX runtime library is responsible for reordering the bidi text before drawing each text string. This solution has a runtime performance and memory impact. Sufficient dynamic memory must be available for bidi text reordering process. This solution requires that the conditional GX_DYNAMIC_BIDI_TEXT_SUPPORT be defined when building the GUIX library. Two APIs gx_system_bidi_text_enable and gx_system_bidi_text_disable are provided to enable/disable bidi text support at runtime.

You should not use both GX_DYNAMIC_BIDI_TEXT_SUPPORT and configure GUIX Studio to generate Bidi text in display order. You must select one strategy or the other for bidi text string handling.

**Memory Usage**

GUIX resides along with the application program. As a result, the static memory (or fixed memory) usage of GUIX is determined by the development tools; e.g., the compiler, linker, and locator. Dynamic memory (or run-time memory) usage is under direct control of the application.

**Static Memory Usage**

Most of the development tools divide the application program image into five basic areas: instruction, constant, initialized data, uninitialized data, and the GUIX thread stack. Figure X on page X shows an example of these memory areas.

It is important to understand that this only an example. The actual static memory layout is specific to the processor, development tools, underlying hardware, and the application itself.

The instruction area contains all of the program’s processor instructions. This area is often located in ROM.

The constant area contains various compiled constants, which in GUIX contains default settings and all application resources (images, strings, fonts, and colors). In addition, this area contains the “initial copy” of the initialized data area. During the compiler’s initialization process, this portion of the constant area is used to set up the global initialized data in RAM. The constant area is typically the largest and usually follows the instruction area and is often located in ROM.
GUIX pixelmaps and fonts typically require large amounts of constant data storage. These large static data areas are normally kept in ROM or FLASH.

The GUIX thread stack is defined within the uninitialized data area (as a global variable) in `gx_system.h` file as follows:

```c
__gx_system_thread_stack[GX_THREAD_STACK_SIZE];
```

`GX_THREAD_STACK_SIZE` is defined in `gx_port.h`, but may be overridden by the application by defining this symbol in the `gx_user.h` header file or via project options or command line parameters. The stack size must be large enough to handle the worst-case event handling and nested drawing calls.

**Dynamic Memory Usage**

As mentioned before, dynamic memory usage is under direct control of the application. Control blocks and memory associated with canvases, etc. can be placed anywhere in the target’s memory space. This is an important feature because it facilitates easy utilization of different types of physical memory – at run-time.

For example, suppose a target hardware environment has both fast memory and slow memory. If the application needs extra performance for drawing, the canvas memory can be explicitly placed in the high-speed memory area for best performance.

Several optional GUIX services and features require a runtime dynamic memory allocation mechanism, commonly referred to as a heap. These services and features are completely optional, and many GUIX applications do not use any heap and do not define a runtime memory allocation mechanism.

If you will be using services which require runtime memory allocation, you must install functions which GUIX will call when memory must be dynamically allocated or freed. You can implement these functions as you prefer, so that even in this case the location of the dynamic memory pool is under application control. To install support for dynamic memory allocation, the application should invoke the API service `gx_system_memory_allocator_set()` during program startup to define your memory allocation and memory free services. Refer to the documentation of this API for a complete example.

GUIX services which require a runtime memory allocation and de-allocation service include:

- Loading binary resources from external storage into the GUIX runtime environment.
- The software runtime jpeg image decoder.
- The software runtime png image decoder.
Using text widgets with GX_STYLE_TEXT_COPY.
- Runtime pixmap resize and rotation utility functions.
- Runtime screen and widget control block allocation.

For smaller applications, GUIX resources are usually compiled and statically linked as part of the application image, and binary resource installation is not required. Binary resources allow an application to install resources (fonts, images, languages) at runtime loaded from some storage location, such as a flash drive or a URL.

The runtime jpeg and png decoders are optional components. Most GUIX applications allow the GUIX Studio tool to pre-decode all required image files, and store them as proprietary GUIX Pixemap data resources. These services are provided for completeness for those applications that require runtime conversion of jpeg and/or PNG images to pixmap format.

GX_STYLE_TEXT_COPY allows the user to specify that a particular widget or widgets will keep its own private copy of dynamically assigned text. Using this option requires that the memory allocation mechanism be installed prior to use. If this style flag is not provided when a text type widget is created, the application must allocate static storage areas for all dynamically created and assigned text strings. Automatic variables should not be used in this case to hold runtime generated string data. If the GX_STYLE_TEXT_COPY style is enabled, automatic variables may be used to hold string data assigned to GUIX widgets, since each widget will create its own copy of the assigned text.

Pixelmap resize and rotation utility functions return the resulting translated pixmap as a new pixmap available to the application. Sufficient dynamic memory must be available to hold these runtime generated pixmap data blocks if these services are used.

Finally, the control blocks for the GUIX screens and widgets can be statically or dynamically allocated. For smaller applications, it is common to create all application screens during program startup and use statically allocated control blocks. For large applications, it is common to create the screen and child widget controls dynamically on an as-needed bases. Dynamically allocated control blocks are specified by selecting the “Runtime Allocate” checkbox in the GUIX Studio properties view, or by passing in the style flag GX_STYLE_DYNAMICALLY_ALLOCATED when creating a widget via the standard API. Using dynamically allocated widget control blocks requires that memory allocation and deallocation services are defined as described above.

**GUIX Components**

The GUIX APIs are divided and organized into several basic groups which correspond to fundamental components of the GUIX system. The fundamental components include:
GX_SYSTEM: The GUIX system component, responsible for initialization, events, timers, string tables, and visible widget hierarchy management.

GX_CANVAS: A drawing area. A Canvas can be a thin abstraction of the hardware frame buffer, or it might also be a pure memory canvas. The canvas type is determined by parameters passed to the \texttt{gx\_canvas\_create} API function.

GX_CONTEXT: The drawing context component. The drawing context contains information about the screen, canvas, and brush, and clipping area for the current drawing operations.

GX_DISPLAY: Provides the APIs and driver-level implementations to allow your application and the GUIX widgets to perform drawing on a canvas. \texttt{GX\_DISPLAY} is implemented to correctly render graphics on each canvas using that canvas’ required color format. The \texttt{GX\_DISPLAY} component also manages the resources (colors, fonts, and pixelmaps) available to widgets drawing to canvases linked to each display.

GX_WIDGET: The basic visible widget object and associated APIs. All GUIX widget types are based on or derived from the basic \texttt{GX\_WIDGET} type.

GXUTILITY: Utility functions for working with rectangles, functions for string conversion, and non-ANSI mathematical functions are included in this group.

In addition to these basic components, GUIX includes APIs unique to each type of widget provided in the library. These APIs are described in the API manual.

**GUIX System Component**

The GUIX system component provides several services that are global to the UI application. These services include: initialization, event management, display management, resource management, timer management, and widget management. Each service is essential to the organization of your application program, and these services are described in more detail in the following sub-sections.

**Initialization**

GUIX initialization is accomplished by the application calling the service \texttt{gx\_system\_initialize}, which may be called by the application from the ThreadX \texttt{tx\_application\_define} routine (initialization context) or from application threads. The \texttt{gx\_system\_initialize} function initializes all global GUIX data structures and creates the
GUIX system mutex, event queue, timer, and thread. Once \texttt{gx\_system\_initialize} returns, the application can use GUIX.

**Thread Processing**

The internal GUIX thread – created during initialization – is responsible for most of the processing in GUIX. The processing in this thread first completes any additional initialization required by the underlying display driver. Once this is complete, the GUIX thread enters a loop which first processes all events present in the GUIX event queue and then refreshes the screen if required. The screen refresh executes the necessary GUIX drawing functions, based on what is visible and has been marked as dirty meaning it needs to be redrawn. When there are no events and nothing left to refresh on the display, the GUIX thread will suspend, waiting for the next GUIX event to arrive.

**RTOS Binding**

The GUIX system component is by default configured to utilize the ThreadX real time operating system for services such as thread services, event queue services, and timer services. GUIX can easily be ported to other operating systems by using the pre-processor directive \texttt{GX\_DISABLE\_THREADX\_BINDING} and re-building the GUIX library. This removes the ThreadX dependencies from the GUIX source code, and allows the application developer to implement the required operating system services using whatever RTOS is provided by the target system. Appendix F, \textit{GUIX RTOS Binding Services}, describes the services that need to be implemented to port GUIX to an operating system other than the ThreadX operating system.

**Multithread Safety**

The GUIX API is available from the GUIX thread context as well as other application threads. Application threads can interact with the GUIX thread by sending and receiving events, by access to shared variables, and through use of the GUIX API functions. GUIX uses an internal ThreadX mutex for multi-thread resource protection. In addition, GUIX prevents the internal structure of visible widgets from being modified once a screen refresh operation has begun. APIs which would modify the tree of visible objects are blocked while drawing operations are in progress, and released once the screen refresh is complete.

**System Timers**

GUIX provides the application with periodic timers, which are often used for periodic update of data displayed in GUIX windows. This is accomplished via a ThreadX periodic timer, which is also used to perform GUIX system-level effects like screen fade in/out, etc.
The application can create timers and utilize the same timer facility that is used internally by GUIX. Of course the application can also directly create and use ThreadX timers if required. The advantage of the GUIX timers is that they are very easy to use and are pre-configured to work within the GUIX event-driven processing system.

To create and start a GUIX timer, the application should invoke the function `gx_system_timer_start`. The parameters to this function include a pointer to the calling widget, the timer id (allowing one widget to start many timers), and the initial and reschedule timeout values. If the reschedule timeout value is 0, the timer will only run one time and will delete itself from the active timer list once it expires.

Once started, the GUIX timer will send GX_EVENT_TIMEOUT events to the timer owner, either once or periodically depending on the timer reschedule value. A GUIX timer can be stopped by calling the API function `gx_system_timer_stop`.

**Pen Speed Configuration**

The GUIX system component holds configuration information related to pen speed tracking. GUIX internally generated GX_EVENT_VERTICAL_FLICK and GX_EVENT_HORIZONTAL_FLICK events based on the speed and distance of PEN_DOWN events generated by the touch input driver, if any. The application can configure the minimum distance and speed required to trigger these internally generated events using the `gx_system_pen_configure()` API function.

**Screen Stack**

The GUIX system component provides services related to the GUIX screen stack, which is an optional functionality supporting a virtual widget stack onto which screens can be pushed, popped, and retrieved at runtime by the application. The screen stack is useful for managing complex menu systems, wherein the route by which the user may arrive at various states in the menu system is varied. Returning to the previous state in the menu system can be easily done by first pushing the previous screen state, then displaying the new screen, and allowing the new screen to pop the previous state from the screen stack when the current screen is dismissed.

**Clipboard Maintenance**

GUIX supports a clipboard for copying and pasting text during runtime execution. This support is provided by the GUIX System component.

**Dirty List Maintenance**

GUIX maintains a list of dirty widgets, meaning widgets that are visible and need to be redrawn due to status changes or being made newly visible. This dirty list improves drawing performance by allowing GUIX to do one canvas refresh operation to reflect all
current changes to the UI status, rather than doing a canvas refresh as each UI change is made. This dirty list is also maintained by the GUIX system component.

**Animation Control Block Pool**

Applications often desire to execute multiple animation sequences, often in parallel. GUIX maintains a pool of animation control blocks from which the application can allocate and use. This frees the application from statically defining these control blocks and allows them to be reused at different times, rather than defining a unique animation control block for every animation that the application might define. The animation control block pool is also maintained by the GUIX system component.

**System Error Handling**

The GUIX system error handler is intended to assist the application in finding internal system errors in GUIX that might be more difficult to determine from the API perspective. Whenever a system error occurs inside of GUIX the internal \_gx\_system\_error\_process function is called. This function saves the error code provided and increments the total number of system errors detected. The system error variables are defined as follows:

```c
UINT     _gx_system_last_error;
ULONG    _gx_system_error_count;
```

If the GUIX application is behaving strangely, it is useful to look at the error count variable in the debugger. If it is set, a good way to troubleshoot the problem is to set a breakpoint in the \_gx\_system\_error\_process function and see when/where it is being called from.

**GUIX Canvas Component**

The canvas component is responsible for all canvas related processing. A canvas is effectively a virtual frame buffer. Your application must create at least one canvas in order to produce graphical output. Typically, you would create at least one canvas for each physical display supported by your system.

All GUIX drawing takes place on a canvas. In simpler or memory constrained systems, there will likely be only one canvas which might be directly linked to the visible frame buffer, whereas systems with more memory and more advanced graphics requirements might require multiple canvases. Making multiple canvases available for one display enables features such as screen and window fade-in and fade-out effects. Canvases can be one of two main types, simple or managed.
A simple canvas is an off-screen drawing area used by the application. GUIX does nothing directly with a simple canvas, but the application can use a simple canvas to render complex drawing to an off-screen buffer, and then use this off-screen buffer to refresh the visible canvas when needed.

A managed canvas is automatically displayed within the hardware frame buffer by GUIX. A managed canvas is included in building a composite canvas for those systems with enough memory to support multiple managed canvases. Managed canvases have a Z-order maintained by GUIX, and view clipping is enforced on all managed canvases.

A canvas differs from a frame buffer in that it is more generic. In memory constrained systems, there may be only one canvas and the memory for this canvas might be the visible frame buffer memory. However, for more complex systems supporting more advanced graphical overlays and multiple canvases, the managed canvases are each allocated their own memory areas which are distinct from the hardware frame buffer memory. These managed canvases are rendered into the visible frame buffer during the frame buffer refresh or toggle operation.

For hardware supporting multiple graphics layers, i.e. multiple overlayed frame buffers, the application can bind one or more canvases to the hardware graphics layers using the `gx_canvas_hardware_layer_bind()` API. This service informs the canvas that it is linked to a particular hardware graphics layer, and once linked this canvas will attempt to utilize hardware support for canvas visibility (i.e. `gx_canvas_show`, `gx_canvas_hide`), canvas alpha blending (i.e. `gx_canvas_alpha_set`) and canvas offset within the display (`gx_canvas_offset_set`).

For architectures other than the simplest single canvas/single frame buffer organization, the size of a canvas is determined by the application and may be different than the fixed size of a frame buffer. It may also be at an offset selected by the application. Other information, such as Z-order is maintained within the canvas. When the canvas drawing is complete, the contents of the canvas are transferred to the physical display by the display driver. In some systems that don’t have enough memory for a separate canvas and frame buffer memory areas, the canvas update is actually made directly to the physical display via the display driver.

**Canvas Creation**

A canvas object can be created during initialization or anytime during the execution of application threads. There is no limit on the number of canvas objects that can be created by an application. Most applications, however, will create only one canvas object for all GUIX drawing.

**Canvas Control Block**

The characteristics of each canvas object are found in its control block `GX_CANVAS` and is defined in `gx_api.h`. The memory required for a canvas object is provided by the
application and can be located anywhere in memory. However, it is most common to make the canvas object control block and the drawing area a global structure by defining them outside the scope of any function.

**Canvas Alpha Channel**

GUIX supports alpha-blending of foreground and background colors on many levels, including bitmap alpha channel which specifies a blending ratio per pixel, brush alpha which specifies the blending ratio for a brush at 16 bpp and higher color depths, and canvas alpha which specifies the blending ratio for an overlay canvas.

The alpha value of a canvas is used when there are multiple canvases which are composited together for display within the frame buffer. If the canvas Z-order is such that a canvas is above other canvases, then the canvas alpha value can be set to blend the canvas with those that lie behind. Rapidly modifying the alpha value of a canvas is used to provide “fade in” screen transition effects, but the canvas alpha can be used in many ways.

If a canvas is bound to a hardware graphics layer using `gx_canvas_hardware_layer_bind()`, GUIX will attempt to implement canvas alpha blending utilizing hardware support, minimizing the software overhead associated with blending an overlay canvas.

Alpha values range from 0 through 255, where a value of 0 means the pixel is fully transparent and values greater than 0 are increasing less transparent canvas alpha value can only be supported for screen drivers running at 16-bpp and higher unless hardware assistance for canvas blend is provided.

**Canvas Offset**

A canvas can be offset within the visible frame buffer by invoking the `gx_canvas_offset_set()` API service. Canvas offsets are usually used to implement sprite animations. If a canvas is bound to a hardware graphics layer by invoking the `gx_canvas_hardware_layer_bind()` API, GUIX will attempt to implement canvas offset changes utilizing hardware support, minimizing the software overhead associated with shifting the canvas position.

**Canvas Drawing**

The GUIX canvas component provides a full drawing API to the application. Before the drawing APIs such as `gx_canvas_line_draw()` or `gx_canvas_pixelmap_draw()` can be invoked, the target canvas must be opened for drawing by invoking the `gx_canvas_drawing_initiate()` API function. This function prepares a canvas for drawing and creates a **drawing context**.
The drawing APIs that render to the canvas, such as gx_canvas_line_draw() or gx_canvas_text_draw(), use parameters found in the current drawing context brush to define the line style, width, and colors. These brush parameters are modified by calling the gx_context_brush_define, gx_context_brush_set, gx_context_brush_style_set, and similar API functions after a drawing context has been established by calling gx_canvas_drawing_initiate.

When GUIX invokes the window and widget drawing functions as part of a deferred canvas refresh operation, the target canvas is opened for drawing prior to calling the widget drawing function(s). Therefore the standard widget drawing functions are not required to open the target canvas, this has been done for them.

In some cases the application may want to force immediate drawing to a canvas. In this case, the application can perform the following steps:

1) Call the gx_canvas_drawing_initiate() API function, passing in the target canvas and rectangle within that canvas on which the application wants to draw.
2) Call any number of canvas drawing APIs to accomplish the desired drawing.
3) Call the gx_canvas_drawing_complete() API function to signal that drawing has been completed. This flushes the canvas to the visible frame buffer and/or triggers a buffer toggle operation, depending on the system memory architecture.

**Drawing APIs**

There are several principal drawing primitives that are required by GUIX to draw all the visual elements on the screen. These drawing APIs can also be invoked by application software, usually as part of a custom widget drawing function. These GUIX canvas drawing APIs perform parameter validation and clipping, and then pass the clipped drawing coordinates down to the display driver for hardware and color-format specific drawing implementations. These drawing APIs are defined as follows:

- gx_canvas_alpha_set()
- gx_canvas_arc_draw()
- gx_canvas_block_move()
- gx_canvas_circle_draw()
- gx_canvas_ellipse_draw()
- gx_canvas_glyphs_draw()
- gx_canvas_hardware_layer_bind()
- gx_canvas_hide()
- gx_canvas_line_draw()
- gx_canvas_offset_set()
- gx_canvas_pie_draw()
- gx_canvas_pixel_draw()
- gx_canvas_pixelmap_blend()
- gx_canvas_pixelmap_rotate()
- gx_canvas_pixelmap_tile()
- `gx_canvas_polygon_draw()`
- `gx_canvas_rectangle_draw()`
- `gx_canvas_rotated_text_draw()`
- `gx_canvas_shift()`
- `gx_canvas_show()`
- `gx_canvas_text_draw()`

The drawing API is invoked via the GUIX Canvas API, and all drawing is done using `gx_canvas_xxx()` API functions. Drawing is done using the current brush in the current drawing context. Any of the shape drawing functions above can be outlined, solid color filled, or pixelmap filled as defined by the current brush. To modify the shape outline width, color, or fill, use the `gx_context_brush_xxx` API functions to define the brush within the current drawing context.

The above application level drawing APIs don’t do actual drawing to the canvas, but instead verify the caller's parameters before invoking the display driver level drawing function. The driver level drawing function actually writes pixel data into the canvas memory.

GUIX provides stock or generic display driver drawing functions for various color depths, including 1, 2, 4, 8, 16, 24, and 32 bits per pixel (bpp). In some cases, the default software drawing implementation is replaced by hardware-accelerated implementations for those hardware targets that provide a 2D drawing accelerator.

### Color Depth

GUIX supports color depths up to 32-bpp as well as monochrome and grayscale. The type of color depth support largely determined by the capabilities of the underlying physical display and also has an impact on how much memory is required for the canvas drawing area. The following is a list of color depth support along with a brief description of the variations within that color depth.

<table>
<thead>
<tr>
<th>Color Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-bit monochrome</td>
<td>1-bit per pixel packed format.</td>
</tr>
<tr>
<td>2-bit grayscale</td>
<td>4 gray levels, packed four pixels per byte.</td>
</tr>
<tr>
<td>4-bit grayscale</td>
<td>16 gray levels, packed two pixels per byte.</td>
</tr>
<tr>
<td>4-bit color</td>
<td>A VGA format planar memory organization.</td>
</tr>
<tr>
<td>8-bit grayscale</td>
<td>256 gray levels</td>
</tr>
<tr>
<td>8-bit palette mode</td>
<td>1 byte per pixel used as palette index.</td>
</tr>
<tr>
<td>8-bit r:g:b mode</td>
<td>A less commonly used 2:3:2 r:g:b format.</td>
</tr>
<tr>
<td>16-bit</td>
<td>Each pixel requires two bytes. Can be r:g:b or b:g:r byte order. Normally 5:6:5</td>
</tr>
</tbody>
</table>
structure, but can also be 5:5:5 structure or 4:4:4:4 a:r:g:b structure.

24-bit Each pixel requires 3 (packed format) or 4 (unpacked format) bytes. Can be in r:g:b or b:g:r byte order as required by hardware.

32-bit Each pixel requires 4 bytes with an alpha channel. Can be a:r:g:b or b:g:r:a byte order and determined by hardware.

**Mouse Support**

GUIX supports drawing a mouse cursor on any desired canvas. The mouse cursor can be drawing in software or might be supported by hardware cursor overlay. In either case, the API provided to the application related to mouse cursor support is the same whether using software or hardware mouse cursor drawing.

GUIX mouse support is only enabled if the #define GX_MOUSE_SUPPORT is defined in the gx_user.h header file before building the GUIX library.

The application must define the mouse cursor and hotspot using the gx_canvas_mouse_define API. This API accepts a pointer to the canvas on which the cursor image should be drawn, and a pointer to a GX_MOUSE_CURSOR_INFO structure, which defines the mouse cursor image and hotspot of the mouse image relative the image top-left corner.

**GUIX Display Component**

The display component is fundamental in GUIX, since it manages the processing of all display objects, which in themselves contain one or more canvases, widgets, and windows. The display component also interacts with the underlying hardware screen driver associated with each display through a series of function pointers.

**Display Creation**

A display object can be created during initialization or anytime during the execution of application threads. Typically an application creates one display object to manage each physical screen. If you have used GUIX Studio to define your application and the physical displays available, you will use the gx_studio_display_configure API function to create and initialize each of your displays.

**Display Control Block**
The characteristics of each display object are found in its control block \texttt{GX\_DISPLAY} and are defined in \texttt{gx\_api.h}. The memory required for a display object is provided by the application and can be located anywhere in memory. However, it is most common to make the display control block a global structure by defining it outside the scope of any function.

\section*{Resource Management}

Resources are UI components that are needed by the application, but they are not application code. Resources are application data and are usually statically defined. Resource types include pixelmaps, fonts, colors, and strings. These resources can be changed at any time, usually without changing any application software. It is important to keep the storage of and references to resources separated from the application software to allow changing UI appearance without changing application code since changes to the application software usually require the associated re-testing and verification of that software.

The GUIX \texttt{display} module provides resource management facilities for all resources that are dependent on the color depth and format of the display. These facilities include maintaining the active pixelmap table, active font table, and active color table. The string table resource is maintained by the GUIX system module, since string resources do not normally need to be changed based on color depth and format.

The application software references resources by their resource Id, which is an index into the corresponding resource table. This allows the table to be changed, for example the color table might be changed when a product changes from “day mode" to “night mode", but the color ID values to remain the same.

Your application resources are written to a resource file (or set of resource files) by the GUIX Studio application. Default colors, pixelmaps, and fonts are provided automatically when you create a new GUIX Studio project, but these defaults are easily replaced as you define the look and feel of your application.

It is important to note that Resource IDs for colors, fonts, and pixelmaps cannot be resolved to their actual color, font, or pixelmap values until the active Display component is known. Since the GUIX architecture supports multiple active displays, Resource IDs can only be resolved to resource values when a widget and its associated Resource ID can be resolved to a specific display. This property is known as dynamic binding. The Resource ID for a property such as a text color, for example the resource ID \texttt{GX\_COLOR\_ID\_TEXT}, might resolve to a 16-bit R:G:B value for white when used on one display, but the same color ID might resolve to a monochrome black color value when used on another display.

In practice this dynamic binding of Resources IDs to resource values means that application software and GUIX internal components should most often only resolve Resource IDs to resource values within an active drawing context. An active drawing
context specifies the currently active display, which allows GUIX to resolve each Resource ID to a specific resource value. If the application software is required to find a specific resource value outside of a drawing context, this can also be done for visible widgets. Visible widgets are linked to a root window which can also be used to resolve the active canvas and display for that widget.

If a widget has been created but not yet displayed (i.e., has not been linked to any root window or other visible parent), any resource IDs associated with that widget cannot be resolved to a specific resource value other than by directly indexing into the resource table assigned to a specific display. This direct access to a specific resource table can safely be done by the application software, but is never done in the internal GUIX library software.

**Widget Defaults**

The GUIX display component also provides default definitions for various widget attributes. Unless otherwise specified by the application, widgets/windows are created with these system attributes. These system attributes are mainly composed of fonts, colors, and bitmaps maintained in the system resource tables. Additional attributes for default scrollbar appearance are also maintained by the GUIX display component.

The default color settings are defined by the color table assigned to each display and the pre-defined default color IDs. These default color ids include:

<table>
<thead>
<tr>
<th>Color ID</th>
<th>Default Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_COLOR_ID_CANVAS</td>
<td>Default canvas (i.e. display background) color</td>
</tr>
<tr>
<td>GX_COLOR_ID_WIDGET_FILL</td>
<td>Default widget fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_WINDOW_FILL</td>
<td>Default window fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_DEFAULT_BORDER</td>
<td>Default widget border color</td>
</tr>
<tr>
<td>GX_COLOR_ID_WINDOW_BORDER</td>
<td>Default window border color</td>
</tr>
<tr>
<td>GX_COLOR_ID_TEXT</td>
<td>Default text color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SELECTED_TEXT</td>
<td>Default selected text color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SELECTED_TEXT_FILL</td>
<td>Default selected text fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SCROLL_FILL</td>
<td>Scrollbar fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SCROLL_BUTTON</td>
<td>Scrollbar button fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SHADOW</td>
<td>Default shadow color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SHINE</td>
<td>Default highlight color</td>
</tr>
<tr>
<td>GX_COLOR_ID_BUTTON_BORDER</td>
<td>Button widget border color</td>
</tr>
<tr>
<td>GX_COLOR_ID_BUTTON_UPPER</td>
<td>Button widget upper fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_BUTTON_LOWER</td>
<td>Button widget lower fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_BUTTON_TEXT</td>
<td>Button widget text color</td>
</tr>
<tr>
<td>GX_COLOR_ID_TEXT_INPUT_TEXT</td>
<td>Text input widget text color</td>
</tr>
<tr>
<td>GX_COLOR_ID_TEXT_INPUT_FILL</td>
<td>Text input fill color</td>
</tr>
<tr>
<td>GX_COLOR_ID_SLIDER_GROOVE_TOP</td>
<td>Color used to draw slider groove.</td>
</tr>
</tbody>
</table>
GX_COLOR_ID_SLIDER_GROOVE_BOTOM Color used to draw slider groove
GX_COLOR_ID_SLIDER_NEEDLE_OUTLINE Color used to draw needle outline
GX_COLOR_ID_SLIDER_NEEDLE_FILL Color used to fill slider needle
GX_COLOR_ID_SLIDER_NEEDLE_LINE1 Color used to draw needle highlight
GX_COLOR_ID_SLIDER_NEEDLE_LINE2 Color used to draw needle shadow

These color ID values are mapped to a specific color value as defined by the color table assigned to each display. These defaults can be changed as a group for one display by calling the `gx_display_color_table_set()` API function. If you are using GUIX Studio, the display color table is automatically initialized when your application calls the `gx_studio_display_configure()` function.

The GUIX display component also maintains a default font table. The default font table defines the font used by each widget type unless specifically specified by the application. The pre-defined display font table IDs include:

- **GX_FONT_ID_DEFAULT** Default font used when no specific font is defined
- **GX_FONT_ID_BUTTON** Default font used for all text on buttons
- **GX_FONT_ID_PROMPT** Default font used for static text
- **GX_FONT_ID_EDIT** Default font used for text edit fields

The font ID used by any text type widget can be re-assigned by using the `gx_<widget_type>_font_set` API provided for each text-related widget type. The entire font table can be re-assigned by calling the `gx_display_font_table_set()` API function.

## Scrollbar Appearance

GUIX Display also maintains default scrollbar appearance settings for that display. These settings are defined by the GX_SCROLLBAR_APPEARANCE structure which is defined below. GUIX Display maintains one scrollbar appearance structure for vertical scrollbars and a second structure for horizontal scroll bars. The application can modify the default scrollbar appearance for any display by initializing a GX_SCROLLBAR_APPEARANCE structure and invoking the API function `gx_display_scroll_appearance_set`.

```c
typedef struct GX_SCROLLBAR_APPEARANCE_STRUCT {
    INT gx_scroll_width;
    INT gx_scroll_thumb_width;
    INT gx_scroll_thumb_travel_min;
    INT gx_scroll_thumb_travel_max;
    GX_RESOURCE_ID gx_scroll_fill_pixelmap;
    GX_RESOURCE_ID gx_scroll_thumb_pixelmap;
    GX_RESOURCE_ID gx_scroll_up_pixelmap;
    GX_RESOURCE_ID gx_scroll_down_pixelmap;
    GX_COLOR gx_scroll_fill_color;
    GX_COLOR gx_scroll_button_color;
} GX_SCROLLBAR_APPEARANCE;
```
gx_scroll_width  
Width of a vertical scrollbar or height of a horizontal scrollbar, in pixels.
gx_scroll_thumb_width  
Width of the elevator and end buttons, in pixels.
gx_scroll_thumb_travel_min  
Offset from end of scroll bar to minimum thumb button travel point.
gx_scroll_thumb_travel_max  
Offset from the end of scroll bar to maximum thumb button travel point.
gx_scroll_fill_pixelmap  
Pixelmap used to fill scroll background.
gx_scroll_thumb_pixelmap  
Pixelmap used to draw scroll thumb button.
gx_scroll_up_pixelmap  
Pixelmap used to draw scroll up button.
gx_scroll_down_pixelmap  
Pixelmap used to draw scroll down button.
gx_scroll_fill_color  
Color ID of color used to fill scrollbar background.
gx_scroll_button_color  
Color ID of color used to fill scrollbar thumb button.

In addition to these default settings for fonts, color, and styles, the application may specify any of the parameters on a case by case basis as desired using API provided by each widget type.

Skinning and Themes

Skinning allows GUIX widgets and windows to easily change their base appearance, i.e., changing the “skin” in one place will change the base appearance of all associated widgets and windows.

Re-skinning your GUIX application requires that you supply a new color, font and or pixelmap table to the GUIX Display resource tables. Since all GUIX widgets refer to their color, bitmap, or font by resource ID, providing a new resource table automatically causes all GUIX widgets to begin using your new colors and pixelmaps when they draw themselves to the desired display.

A new set of fonts, colors, and pixelmaps that are designed to work together to provide an attractive appearance is called a Theme. A theme defines a set of resource tables and the size of each resource table. Any number of themes can be defined for any display using the GUIX Studio application. You must pass the starting theme index to the GUIX Studio generated function gx_studio_display_configure(), which installs the initial theme into the created display. The active theme for any display can be changed at any time by calling the function gx_display_theme_install().

GUIX User Guide  43
**Root Window**

For each visible canvas created by an application, the application must also create one Root Window for that canvas. This special window basically acts as a container for all the top-level application windows and widgets. The root window draws the canvas background, and since the root window is derived from the GX_WINDOW class the root window can also have wallpaper. To change the background color of your display or canvas, you simply change the fill color of the root window attached to that canvas.

If you use the GUIX Studio generated function named `gx_studio_display_configure` to configure your displays, then the canvas and root window for each display are created for you as part of this initialization function.

**Anti-Aliasing**

Anti-Aliasing is an optional feature in GUIX that is used to smooth lines, curves, and fonts. Anti-aliasing is only supported when running with a display driver utilizing 16-bpp or higher color depth.

Anti-aliased line drawing is enabled by setting the GX_BRUSH_ALIAS flash in the active brush. This applies to lines drawn directly as well as to lines drawn as the border of a polygon or circle.

Anti-aliased text drawing is enabled by using an anti-aliased font which is produced by the GUIX studio application. You specify whether a font should be generated as anti-aliased or binary when you create the font. Anti-aliased fonts in GUIX utilize 16 levels of transparency for each pixel.

**Clipping**

Clipping is supported internally by the GUIX display component, and at the window and widget layers by the parent-child architecture maintained by GUIX widgets. No window or widget is ever allowed to draw outside of that widget’s area, and a widget is never allowed to draw outside of the area of that widget’s parent.

This also prevents widgets from drawing at pixel coordinates that lay outside of the canvas memory which likely lead to memory corruption or a system failure. Widgets are not allowed to draw outside of the widget’s area, the widget’s parent area, or beyond the canvas extent.

In addition, widgets can only draw to areas that have previously been marked as dirty. This prevents an entire window being drawn, for example, when only a corner of the window has been revealed. Only the portion of the window that actually needs to be refreshed is marked as dirty, and so the window drawing function only truly refreshes pixels in the dirty area.
The GUIX display component enforces these clipping algorithms before invoking the driver level drawing functions.

**Views**

GUIX always maintains a set of views for each root window and each child window of the root window. Views are a dynamic, run-time determined clipping area that changes as window position and Z-order are modified. GUIX uses views to prevent a window or widget in the background from drawing on top of a window or widget in the foreground. Views enforce Z-order discipline. In addition, views are important for efficiency in that they prevent a window in the background from drawing to any area of the canvas that cannot be seen. If a window is completely covered by another window, the covered window will not be allowed to draw to the canvas at all, even if it is attempting to do so.

**Display Driver Interface**

GUIX display drivers are responsible for all interaction with the underlying physical screen. The display drivers have three basic functions: initialization, drawing, and frame buffer display. Initialization is responsible for preparing the physical display hardware, informing GUIX of the properties of the physical display hardware, and for informing GUIX which specific drawing functions should be used. The main display driver initialization is called from the GUIX `gx_display_create()` function. In addition, the GUIX thread will also call a secondary display driver initialization from the thread context. This secondary display driver is only needed if the driver requires RTOS services during its initialization, e.g., device interrupts or `tx_thread_sleep` requests for delay between steps in the initialization process.

Once initialization is complete, the display driver is responsible for any direct drawing that can be done in the physical display hardware. Finally, the display driver is responsible for displaying the frame buffer.
GUIX Widget Component

A GUIX widget is a visible graphical element. There are GUIX components which are not visible, such as timers and math utility functions. However all visible components are derived from the basic GUIX widget component. A GUIX widget is the primary building block of the GUIX display – all other graphic elements are derived from the base widget functionality.

GUIX widgets are implemented in an object oriented manner with full support of inheritance. This is accomplished using ANSI C, which results in the smallest possible memory and processing requirements. When we speak of one particular widget, such as `GX_BUTTON`, being derived from another widget, such as the base `GX_WIDGET`, what we mean is that the `GX_BUTTON` control structure contains all of the member variables and function pointers of `GX_WIDGET`, with some additional variables that are specific to `GX_BUTTON`. GUIX builds up layers of widgets in this fashion, so that more complex widgets are always based on a simpler widget before them. This hierarchical model of derivation makes it easier to learn the APIs used to modify widget parameters. If you want to modify the color of a button, you use the same API you use to modify the color of a widget, namely `gx_widget_fill_color_set`.

The organization of visible widgets is maintained in a parent-child manner using tree structured lists linking child widgets to their parents. The children inherit characteristics from their parents such as the views into which they can draw and the canvas on which they draw. Child widgets may have their own child widgets, again inheriting various characteristics from the parent. The characteristics of any widget may be explicitly redefined via various GUIX API calls.

Widget Creation

A widget object can be created during initialization or anytime during the execution of application threads. There is no limit on the number of widget objects that can be created by an application. There is also no limit on the number of children any widget may have, within the memory limits of your target hardware.

Each widget type has its own create function, such as `gx_button_create` or `gx_prompt_create`. The first three parameters to these functions are always the same, namely a pointer to the widget control structure, a string pointer to the widget name, and a pointer to the widget’s parent. Each create function may have any number of additional parameters depending on the requirements of that particular widget type.

Widget Control Block

The characteristics of each widget object are found in its control block `GX_WIDGET` and are defined in `gx_api.h`. The memory required for a widget object is provided by the application and can be located anywhere in memory. However, it is most common to
make the widget object control block a global structure by defining it outside the scope of any function. If you are using GUIX Studio, your widget control blocks can be statically allocated within your Studio generated specifications file, or they can be dynamically allocated by your application.

**Dynamic Widget Control Block Allocation and De-allocation**

If you are using dynamic control block allocation, you will need to define two functions that GUIX will use to allocate and free the memory required for your widget control blocks. Your functions for memory management are passed to the GUIX system component via the `gx_system_memory_allocator_set()` API function. This function allows you to pass two function pointers into GUIX: the first is a pointer to a memory allocation function, and the second is a pointer to a memory free function. Most often, you will implement these functions using ThreadX byte pools, but the design of GUIX allows you to implement dynamic memory management in whatever way you prefer.

Dynamic widget allocation is most often employed within your Studio generated application specifications file, when you select the “dynamically allocated” option in the Studio widget properties field. However, you can also use dynamic control block allocation within your application. If you use dynamic control block allocation within your application, you should invoke the `gx_widget_allocate(GX_WIDGET **widget, ULONG memsize)` API function to allocate the widget control block. Next, when you create the widget, make certain you pass the `GX_WIDGET_STYLE_DynamicallyAllocated` style flag (along with any other needed style flags) to the widget create function. This flag is used to mark the widget as being dynamically allocated in the widget status field. When and if the widget is later deleted using `gx_widget_delete()`, GUIX will check this status field and automatically call your memory de-allocator function to insure there are no memory leaks.

A widget created using a dynamically allocated control block must be created with the `GX_WIDGET_STYLE_DynamicallyAllocated` style flag to prevent memory loss.

**Types**

GUIX provides a rich, fully functional set of built-in widgets. As mentioned previously, all specialized widgets are derived from the base widget. Following is a list of the built-in widgets in GUIX:

- `GX_TYPE_WIDGET`
- `GX_TYPE_BUTTON`
- `GX_TYPE_TEXT_BUTTON`
Styles

Widget styles consist of things like border properties (raised, recessed, thin, thick, or no border at all) as well as properties for specific widget types, as listed previously. The widget style flags offer the simplest method for modifying the appearance of any widget. The initial widget style is always a parameter passed to the widget type specific create function.
Colors

Widgets draw themselves using colors defined in the system color table. Color IDs are defined for canvas background, default widget fill color, button fill color, text widget fill color, window fill color, and several other default color values. In addition, GX_WINDOW objects support displaying a bitmap or wallpaper as the window client is filled.

The simplest method of changing the default color scheme is to use GUIX Studio and create or define a color scheme that meets your requirements. You can also define your color scheme manually by creating an array of GX_COLOR values and invoking the gx_system_color_table_set API function.

Event Notification

GUIX events are requests made to one or more widgets to perform a particular action and notifications to notify widgets of user input and internal system status changes. For example, when a widget gains focus, the GX_EVENT_FOCUS_GAINED is sent to the widget via the gx_system_event_send API service.

Events are passed through the GUIX event queue, and each event is an instance of the GX_EVENT data structure. The GX_EVENT data structure is defined in gx_api.h, however the most important fields of the structure are the gx_event_type, gx_event_sender, gx_event_target, and gx_event_payload.

The gx_event_type field is used to identify the particular event class. The event type indicates if this is, for example, a GX_EVENT_PEN_DOWN event or a GX_EVENT_TIMER event. The gx_event_payload is a union of various data fields, and they are not all valid for every event type. You use the event type field first, before examining the other event data fields.

The gx_event_sender field contains the ID of the widget that generated the event if the event is a child-widget notification.

The gx_event_target field can be used to route user-defined events to a particular window or widget. If you want to send an event to a particular window, you should give the window a unique Id value (so that it can be positively identified), and when building the event place the window Id value in the gx_event_target field. If you don't know the target Id or if you just want the event to be routed to the widget that has input focus, make sure to set the gx_event_target field to 0.

Finally, the gx_event_payload field is a union of various data types. For GX_EVENT_PEN_DOWN and GX_EVENT_PEN_UP events, the gx_event_pointdata field contains the x,y pixel coordinate the pen position. For timer events, the gx_event_timer_id contains the ID of the expired timer. Other payload data fields are utilized for other event types. The complete list of pre-defined event types and their
payload fields is defined in Appendix E of this manual, titled “GUIX Event Descriptions”.

The application can also add its own custom events, starting numerically after the constant `GX_FIRST_APP_EVENT`. All event numbers after this constant are reserved for the application’s use. Of course, the application’s widget event handler must have processing for such application events.

**Event Processing**

There is a default widget event processing function for each and every widget, named `gx_<widget-type>_event_process`. In most cases, the application won’t need to worry about the event handling of any given widget. However, in situations where the application requires custom or supplemental event processing, the application may override the default processing function with its own via the GUIX API `gx_widget_event_process_set`. This function overrides the default event processing function with the event function processing function specified in the API.

Application event processing functions can take advantage (i.e., not duplicate the processing) of the default processing by simply calling the default `gx_widget_event_process` processing directly.

Event processing is called exclusively from the context of the internal GUIX system thread. In this way, the stack requirements to process the event handling only applies to the GUIX thread.

**Implementing Custom Event Processing (example)**

You can provide your own event processing function for any widget or window in the GUIX system. If you are creating your own custom widget type, you will normally install your custom event handler in the widget creation function. If you are just extending or modifying the operation of an existing widget or window, you can call the `gx_widget_event_process_set` API function after the widget or window has been created. You will almost always provide your own event handling for your top-level windows (also called Screens) in order to process events generated by your child controls. Processing event generated by the child controls of a screen is the main way you add functionality to your GUIX application.

As an example, suppose you define a top-level screen named “main_menu”. This screen might be defined using GUIX Studio, or you might create this screen in your application code. If you define the screen within GUIX Studio, you simply type the name of your event handler in the Studio properties field for that screen, and the Studio generated specifications code will automatically install your event handler. In this case, we will call the custom event handler “main_menu_event_handler()” and it should be coded like this:
int main_menu_item; /* example: variable to keep track of selected item */

UINT main_menu_event_handler(GX_WINDOW *main_screen, GX_EVENT *event_ptr)
{
    UINT status = GX_SUCCESS;

    switch(event_ptr->gx_event_type)
    {
    /* this is an example for catching events from a child button */
    case GX_SIGNAL(IDB_CHILD_BUTTON, GX_EVENT_CLICKED):
    /* insert your button handler code here */
        break;

    case GX_EVENT_SHOW:
    /* add functionality to the show event handler */
    /* first, do default processing */
        status = gx_window_event_process(main_screen, event_ptr); /* note 1 */
    /* now add my own processing */
        main_menu_item = 0;
        break;

    default:
    /* pass all other events to base processing function */
    status = gx_window_event_process(main_screen, event_ptr); /* note 1 */
        break;
    }
    return status;
}

In the example above, it is important to notice that for system events like GX_EVENT_SHOW (events generated internally to notify a widget of a status change), the application must pass those events to the base widget event processing function to insure that the normal processing occurs. The application can then add additional logic as desired. All events that are not handled by the application (the default case above) should also be passed to the base event processing function. Since this example was for a top-level screen based on GX_WINDOW, the default event processing function is gx_window_event_process.

### Drawing Function

All widget drawing is performed separately from the event handling. This is more efficient because drawing is usually expensive in terms of CPU cycles. By implementing a deferred drawing algorithm, all of the outstanding events and associated display changes can be completed before any drawing is done, thus eliminating wasted drawing. Similar to event processing, there is a default widget drawing function for most widgets, named `gx_xxx_draw`, where `xxx` is the widget type. In most cases, the application won’t need to worry about the drawing function of any given widget. However, in situations where the application requires custom or supplemental drawing, the application may override the default drawing function with its own via the GUIX API `gx_widget_draw_set`. This function allows the application to provide its own customized drawing function for any widget. This further allows the application to define entire new widget types.
Application drawing functions can take advantage (i.e., not duplicate the coding) of the default drawing by simply calling it directly from the overridden drawing function.

Widget drawing is called exclusively from the context of the internal GUIX system thread. In this way, the timing and stack requirements to perform the drawing only apply to the GUIX thread.

Implementing Custom Drawing (example)

The drawing function for any widget is referenced through an indirect function pointer which is a member of the GX_WIDGET control block. If you use GUIX Studio to define your widget, you can install your own function pointer simply by typing the name of your function in the “Drawing Function” parameter of the widget properties, and Studio will install your function pointer for you when the widget is created. If you create the widget in your application code, you must use the gx_widget_draw_set() API function to install your custom drawing function after the widget has been created.

For this example, let’s assume that you want to customize the appearance of a button. The button will look very much like a GX_TEXT_BUTTON, but we will add drawing a small green "LED_ON" bitmap in the middle-right portion of the button when the button is pressed, and small “LED_OFF” bitmap when the button is not pressed. We want to create a button that looks like this:

```
button 🟢
```

custom button “on”

```
button 🔴
```

custom button “off”
In this case, we would write a button drawing function that looks something like this:

```c
UINT my_button_draw(GX_TEXT_BUTTON *button)
{
    GX_PIXELMAP *map;
    ULONG button_style;
    INT xpos;
    INT ypos;

    /* first, do the normal text button drawing */
    gx_text_button_draw(button);

    /* now add our extra pixelmap */
    gx_widget_style_get(button, &button_style);
    if (button_style & GX_STYLE_BUTTON_PUSHED)
    {
        /* use the ON pixelmap */
        gx_context_pixelmap_get(GX_PIXELMAP_ID_LED_ON, &map);
    }
    else
    {
        /* use the OFF pixelmap */
        gx_context_pixelmap_get(GX_PIXELMAP_ID_LED_OFF, &map);
    }
    if (map)
    {
        /* draw it 20 pixels in from right edge */
        xpos = button->gx_widget_size.gx_rectangle_right;
        xpos -= map->gx_pixelmap_width + 20;

        /* and draw 10 pixels from the top edge */
        ypos = button->gx_widget_size.gx_rectangle_top + 10;

        /* draw the extra pixelmap on top of the button */
        gx_canvas_pixelmap_draw(xpos, ypos, map);
    }
}
```

---

**GUIX Drawing Context Component**

The drawing context is created dynamically, at runtime, as GUIX performs each canvas refresh operation. The drawing context ties together the canvas, screen driver, and brush being used to perform the current drawing operations.

The drawing context is defined by the `GX_DRAW_CONTEXT` structure. This structure contains variables that define the clipping and view of the current drawing operation, define the current canvas, and define the current screen driver in use. The `GX_DRAW_CONTEXT` structure also holds the brush being used for drawing. The draw context brush is the member that you will work directly with in your custom drawing functions. The brush structure is defined as:

```c
typedef struct GX_BRUSH_STRUCT
{
    GX_PIXELMAP *gx_brush_pixelmap;
    GX_FONT *gx_brush_font;
    ULONG gx_brush_line_pattern;
}
```
The `gx_brush_pixelmap` defines a pixelmap to use for rectangle and polygon fills. This member is not used unless the `gx_brush_style` includes the `GX_BRUSH_PIXELMAP` style.

The `gx_brush_font` member defines the font used for text drawing. The `gx_brush_line_pattern` member defines the pattern used for dashed lines. The `gx_brush_style` member is a set of style flags that can be OR’d together to fully define the brush attributes. The available brush style flags include:

- `GX_BRUSH_OUTLINE`
- `GX_BRUSH_SOLID_FILL`
- `GX_BRUSH_PIXELMAP`
- `GX_BRUSH_ALIAS`
- `GX_BRUSH_UNDERLINE`
- `GX_BRUSH_ROUND`

The `gx_brush_width` member defines the line width for line drawing, or the outline width for outlined shape drawing.

The `gx_brush_line_color` member defines the foreground color for line drawing and for text drawing.

The `gx_brush_fill_color` member defines the solid fill color used for shape filling. The GUIX context component provides a set of APIs designed to make it very easy to modify the current brush within the active context. These APIs include `gx_context_brush_define`, `gx_context_line_color_set`, `gx_context_fill_color_set`, `gx_context_font_set`, and many others.

The draw context of a parent object is inherited by the objects children. Actually, a clone of the parent drawing context is inherited by the child objects when their drawing functions are invoked. The child can modify the context without affecting the parent drawing, but it can also inherit information from the parent such as brush color and style if desired.

**GUIX Window Component**

The window component is responsible for all window processing in GUIX. A GUIX window is fundamentally a distinct display area that may contain one or more child widgets. In GUIX, the window is actually just a special form of the fundamental widget object.
GUIX windows are implemented in an object oriented manner with full support of inheritance. This is accomplished using ANSI C, which results in the smallest possible memory and processing requirements.

GUIX windows extend the functionality of the GUIX widget primarily by adding support for horizontal and vertical scrolling. GUIX window objects can automatically create and display scroll bars and respond to scroll bar input. Movable windows also have built in event handling to allow the window to be moved or dragged based on pen input events. Finally, GUIX window responds to receiving input focus by moving the window to the front of the window Z-order.

GUIX window maintains the concept of client area, which is the inner portion of the window once the window borders and non-client objects such as scrollbars are removed from the available area. Client area child widgets are clipped to the window client area, while non-client children such as scroll bars are allowed to draw outside of the client area, but are still clipped to the window outer dimensions.

Windows are maintained in a parent-child manner, where the children inherit characteristics from their parent. Children windows may have their own child windows, again inheriting various characteristics from the parent. The characteristics of any window may be explicitly redefined via various GUIX API calls.

Window Creation

A window object can be created during initialization or anytime during the execution of application threads. There is no limit on the number of window objects that can be created by an application. There is also no limit on the number of children any window may have.

Window Control Block

The characteristics of each window object are found in its control block GX_WINDOW and are defined in gx_api.h. The memory required for a window object is provided by the application and can be located anywhere in memory. However, it is most common to make the window object control block a global structure by defining it outside the scope of any function.

Root Window

GUIX requires what is called a root window for each canvas. The root window is borderless and has the same dimensions as the canvas to which it is attached. It is used primarily as a container for all first-level widgets and windows. The root window is typically created by the application via the API gx_window_root_create, shortly after the creation of the screen and canvas. If you use the Studio generated function gx_studio_display_configure, the address of the root window can be returned in the location passed as the last parameter to this function.
A root window defaults to being un-moveable, and in the simplest case the root window is the size of the canvas. The root window in effect draws the display background, so to change the display background color or to display background wallpaper you would assign a color or wallpaper to the root window.

If a root window is moveable, it moves not by changing its position on the canvas as a child window would do, but by moving the canvas itself. This feature allows the GUIX root window to leverage hardware that supports multiple frame buffers with hardware offset registers.

**Background**

Window backgrounds are either solid colors or bitmap images. There is a default window background at the system level which provides the default for the initial set of windows. Of course, any window background can be changed via the GUIX API.

To change the solid color background of a window, use the `gx_widget_fill_color_set()` API. To assign a background wallpaper to a window, use the `gx_window_wallpaper_set()` API.

**Scrolling**

GUIX supports standard window scrolling when area required to display the window children exceeds the current size of the window – horizontally and/or vertically. To enable scrolling, the application must create the desired scroll bars and attach them to the window.

The GUIX window component provides a default scrolling implementation based on the size of the window client area and the extent of the all child widgets. Applications can also provide their own scrolling implementation and interpretation by overriding the `gx_window_scroll_info_get` function for a particular window.
Event Notification

The default window event processing function differs from the GX_WIDGET event processing primarily in the handling of scrolling and sizing events. GX_WINDOW provided default handlers for scrolling and sizing events.

The application can also add its own custom events, starting numerically after the constant `GX_FIRST_APP_EVENT`. All event numbers after this constant are reserved for the application’s use. Of course, the application’s window event handler must have processing for such application events.

Event Processing

Just like all other widget types, there is a default window event processing function for every window, named `gx_window_event_process`. You will usually override this event handling function with your own event handler in the windows that you create. This is how you will respond to events and take action based on events generated by the window child controls.

It is important to remember to invoke the base `gx_window_event_process` function for GUIX system events if you override that event handler, to allow the default event handling to occur in addition to whatever action you are adding to the event handler. For example if you provide a custom handler for the `GX_EVENT_SHOW` event, and do not pass this event to `gx_window_event_process`, your window will never become visible. To provide a custom event handler for a window, use the `gx_widget_event_process_set` function to define the address of your event handler. This function overrides the default event processing function with the event function processing function specified in the API.

Application event processing functions can take advantage (i.e., not duplicate the processing) of the default processing by simply calling the default `gx_window_event_process` directly.

Event processing is called exclusively from the context of the internal GUIX system thread. In this way, the stack requirements to process the event handling only applies to the GUIX thread.

GUIX Image Reader Component

The image reader component provides utilities and API functions to decompress raw compressed images to GUIX pixmap format. JPEG and PNG format raw image data are supported, with additional formats reserved for future releases.

Note that for the vast majority of GUIX applications, the GUIX Image Reader component is not required. Most applications rely on the GUIX Studio application to convert JPEG
and PNG format graphics assets into GUIX compatible GX_PIXELMAP resources. The GUIX image reader component is utilized when the raw graphics assets are known only at runtime, or when the system storage constraints prevent storing resources in GX_PIXELMAP format. JPEG and PNG format image data is generally more compact than GX_PIXELMAP format, however there is considerable runtime overhead associated with performing decompression and color space conversion of these image types dynamically.

If raw format JPEG or PNG images are passed to the gx_canvas_pixelmap_draw API function, GUIX dynamically decompresses and draws the JPEG or PNG data. Note that this will have a significant negative impact on runtime drawing speed, and passing RAW format image data to the gx_canvas_pixelmap_draw function is not recommended unless you are using a hardware target that supports hardware assisted JPEG or PNG decompression.

Passing raw JPEG or PNG formatted images to the gx_canvas_pixelmap_draw API incurs significant runtime overhead for most target hardware.

As an alternative, raw JPEG and PNG data may be converted to GX_PIXELMAP format at runtime using the Image Reader component. Pixelmaps produced in this way can be used and drawn just like pixelmaps produced by Studio and contained within your resource file. This allows your application to perform the image decompression, dithering, and color space conversion one time (usually during program startup) rather than performing these operations each time the image is drawn.

The Image Reader component functions include:

- `gx_image_reader_create`
- `gx_image_reader_palette_set`
- `gx_image_reader_start`

**GUIX Animation Component**

The GUIX Animation component is a set of functions and services used to automate screen and widget transition automations. The GUIX Animation component supports fading in, fading out, and movement or slide type animations of any widget type.

Fade type animations can be supported either by varying the fading widget(s) internal alpha value (if GX_BRUSH_ALPHA_SUPPORT is enabled), or by drawing any collection of widgets to a separate memory canvas when is then blended with the background. For hardware targets that support multiple hardware graphics layers, support for smooth fading effects is best accomplished using this canvas blending
approach, often with very little core CPU bandwidth required. For hardware targets that
do not support multiple graphics layers, blending using the GUIX brush alpha value is
supported when running at 16 bpp and higher color depths.

If an animation should use a separate drawing canvas, the GUIX Animation component
provides the API service gx_animation_canvas_define for this purpose. Other animation
types do not require a separate canvas, but they will utilize it if it is available. This
makes the best possible use of any underlying hardware support for multiple hardware
surfaces.

The variables controlling an animation are defined by two control blocks. First, the
GX_Animation control block which defines the animation controller. The animation
controller is the driving engine that executes the animation sequence you define. A
single animation controller can be re-used many times to run many different animation
sequences. If you need to run multiple animation sequences simultaneously, you can
create multiple GX_Animation animation controllers.

The GUIX system component can provide a re-usable block of GX_Animation control
structures, which can be requested by the application when and animation is needed
and are automatically returned to the system pool when the animation sequence is
completed. This frees the application from statically defining a GX_Animation
structure for every animation that might be implemented. The size of this pool of
GX_Animation structures is defined by the constant GX_Animation_POOL_SIZE,
which defaults to 6, meaning that by default 6 simultaneous animations can be allocated
from the system pool. This constant can of course be re-defined in the gx_user.h
header file is more simultaneous animations are required. If
GX_Animation_POOL_SIZE is set to zero, then the GUIX system component does
not provide an animation pool or related services.

The second control block or structure used to define an animation is the
GX_Animation_INFO structure. This structure is used to define one particular
animation sequence. You pass this information structure to your animation controller to
initiate an animation sequence using the gx_animation_start API service. The
GX_Animation_INFO structure contains the following fields:

typedef struct GX_Animation_INFO_STRUCT
{
    GX_WIDGET      *gx_animation_target;
    GX_WIDGET      *gx_animation_parent;
    GX_WIDGET      *gx_animation_screen_list;
    USHORT         gx_animation_style;
    USHORT         gx_animation_id;
    USHORT         gx_animation_start_delay;
    USHORT         gx_animation_frame_interval;
    GX_POINT       gx_animation_start_position;
    GX_POINT       gx_animation_end_position;
    GX_UBYTE       gx_animation_start_alpha;
    GX_UBYTE       gx_animation_end_alpha;
    GX_UBYTE       gx_animation_steps;
}
The `gx_animation_target` member defines the target widget that the animation controller will act upon.

The `gx_animation_parent` member defines the parent widget, if any, to which the target widget will be attached when the animation sequence is complete. The `gx_animation_parent` is also the recipient of the GX_ANIMATION_COMPLETE event that is generated when an animation is completed.

The `gx_animation_screen_list` member defines a widget list for pen-input-driven screen slide animations. The widget list should be terminated with GX_NULL pointer, and each widget in the list should have the same x,y dimensions as the `gx_animation_parent`.

The `gx_animation_style` is a bitmask defining the type of animation to be performed and associated options. The animation style flags include:

- `GX_ANIMATION_TRANSLATE` - Request a slide or fade type animation
- `GX_ANIMATION_SCREEN_DRAG` - Request a pen-input driven screen drag animation

The following flags can be used in combination with SCREEN_DRAG type animations:

- `GX_ANIMATION_WRAP` - The screen list should wrap from end back to start
- `GX_ANIMATION_HORIZONTAL` - Screen drag allowed in horizontal direction
- `GX_ANIMATION_VERTICAL` - Screen drag allowed in vertical direction

The following flag can be used in combination with translate animations:

- `GX_ANIMATION_DETACH` - Detach the animation target from the animation parent when the animation is completed. If the target was dynamically allocated and created by the GUIX Studio generated automated event handling, the target will be deleted after it is detached.

`GX_ANIMATION_TRANSLATE` animation types are timer driven animations. The application defines the starting and ending position and starting and ending alpha value for the target widget, and the animation manager creates a timer to serve as the driving force to execute the animation.

`GX_ANIMATION_SCREEN_DRAG` differs from the TRANSLATE animations in that this animation type is driven by pen input events. This animation type tracks along with the touch screen input to swipe the target widget as the user drags a pen or stylus across the input touch screen. To utilize this type of animation, the application should call the `gx_animation_drag_enable()` API to enable this animation.

The `gx_animation_id` value is passed back to the animation parent in the event.gx_event_sender field of the GX_ANIMATION_COMPLETE event. This value is used by the animation parent to determine which of possibly several child animations is

```c
} GX_ANIMATION_INFO;
```
reporting completion. This value can be 0, and an animation with ID value 0 will not
generate an ANIMATION_COMPLETE event at all.

The `gx_animation_start_delay` value is a GUIX tick count indicating the number of
timer ticks to delay after `gx_animation_start()` is called before actually executing the
animation. The value can be 0 to start the animation immediately upon calling
`gx_animation_start()`.

The `gx_animation_frame_interval` field defines the number of GUIX timer ticks (a
multiple of the underlying OS tick rate) to delay between each frame of the animation
sequence. The minimum value is 1.

The `gx_animation_start_position` defines the top-left starting point for the target
widget for translation animations.

The `gx_animation_end_position` defines the top-left ending position for the target
widget for translation type animations.

The `gx_animation_start_alpha` field defines the starting canvas alpha value for
translation type animations.

The `gx_animation_end_alpha` field defines the ending canvas alpha value for
translation type animations.

The `gx_animation_steps` field defines how many steps or frames the controller should
execute for translation animations. A larger number of steps produces a smoother slide
and/or fade appearance, but also requires greater system bandwidth.

To implement animation effects in your application, you must first call
gx_animation_create() to initialize your animation controller. If your animation will be
using a secondary canvas, initialize this canvas by calling `gx_animation_canvas_define`
Next, you should initialize the GX_ANIMATION_INFO structure to define the specific
type of animation to be performed and the other animation parameters. Finally, call
gx_animation_start to trigger the animation sequence.

When the animation controller completes an animation sequence, it sends an
GX_ANIMATION_COMPLETE event to the parent widget, allowing the any desired
cleanup of the animation canvas to be done at that time.

**GUIX Utility Component**

The utility component is responsible for all common utility functions in GUIX. These are
common functions that are useful utilities and can be invoked from anywhere in the
application or the internal GUIX code. The utility component functions include:

```c
gx_utility_alphamap_create
```

GUIX User Guide
gx_utility_gradient_create
gx_utility_gradient_delete
gx_pixelmap_transparent_detect
gx_utility_ltoa
gx_utility_math_acos
gx_utility_math_asin
gx_utility_math_cos
gx_utility_math_sin
gx_utility_math_sqrt
gx_utility_pixelmap_resize
gx_utility_pixelmap_rotate
gx_utility_pixelmap_simple_rotate
gx_utility_rectangle_center
gx_utility_rectangle_center_find
gx_utility_rectangle_combine
gx_utility_rectangle_compare
gx_utility_rectangle_define
gx_utility_rectangle_grow
gx_utility_rectangle_inside_detect
gx_utility_rectangle_overlap_detect
gx_utility_rectangle_point_detect
gx_utility_rectangle_resize
gx_utility_rectangle_shift
gx_utility_string_to_alphamap
 gx_utility_unicode_to_utf8
gx_utility_utf8_string_character_count_get
Chapter 4: Description of GUIX Services

This chapter contains a description of all GUIX services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the `GX_DISABLE_ERROR_CHECKING` define that is used to disable API error checking, while non-bold values are completely disabled.

<table>
<thead>
<tr>
<th>GUIX Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gxAccordionMenuCreate</td>
<td>Create accordion menu</td>
</tr>
<tr>
<td>gxAccordionMenuDraw</td>
<td>Draw accordion menu</td>
</tr>
<tr>
<td>gxAccordionMenuEventProcess</td>
<td>Process accordion menu event</td>
</tr>
<tr>
<td>gxAccordionMenuPosition</td>
<td>Position menu items</td>
</tr>
<tr>
<td>gxAnimationCanvasDefine</td>
<td>Provide memory to an animation controller for a canvas to be used for subsequent animations.</td>
</tr>
<tr>
<td>gxAnimationCreate</td>
<td>Create an animation controller</td>
</tr>
<tr>
<td>gxAnimationDelete</td>
<td>Delete an animation controller</td>
</tr>
<tr>
<td>gxAnimationDragDisable</td>
<td>Disable screen drag animation hook</td>
</tr>
<tr>
<td>gxAnimationDragEnable</td>
<td>Enable screen drag animation hook</td>
</tr>
<tr>
<td>gxAnimationLandingSpeedSet</td>
<td>Set landing speed for screen drag animation</td>
</tr>
<tr>
<td>gxAnimationStart</td>
<td>Initiate an animation sequence</td>
</tr>
<tr>
<td>gxAnimationStop</td>
<td>Suspend an animation sequence</td>
</tr>
<tr>
<td>gxBinresLanguageTableLoad</td>
<td>(deprecated) Load a language table from binary resource data buffer</td>
</tr>
<tr>
<td>gxBinresLanguageTableLoadExt</td>
<td>Load a language table from binary resource data buffer</td>
</tr>
<tr>
<td>gxBinresThemeLoad</td>
<td>Load a theme from binary resource data buffer</td>
</tr>
<tr>
<td>gxBrushDefault</td>
<td>Initialize current brush to defaults</td>
</tr>
<tr>
<td>gxBrushDefine</td>
<td>Define brush</td>
</tr>
<tr>
<td>gxButtonBackgroundDraw</td>
<td>Draw button background</td>
</tr>
<tr>
<td>gxButtonCreate</td>
<td>Create button</td>
</tr>
<tr>
<td>gxButtonDeselect</td>
<td>Deselect button</td>
</tr>
<tr>
<td>gxButtonDraw</td>
<td>Draw button</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>gx_button_event_process</td>
<td>Process button event</td>
</tr>
<tr>
<td>gx_button_select</td>
<td>Select button</td>
</tr>
<tr>
<td>gx_canvas_alpha_set</td>
<td>Set alpha-blend value for canvas</td>
</tr>
<tr>
<td>gx_canvas_arc_draw</td>
<td>Draw circle arc</td>
</tr>
<tr>
<td>gx_canvas_block_move</td>
<td>Move block</td>
</tr>
<tr>
<td>gx_canvas_circle_draw</td>
<td>Draw circle</td>
</tr>
<tr>
<td>gx_canvas_create</td>
<td>Create a canvas</td>
</tr>
<tr>
<td>gx_canvas_delete</td>
<td>Delete a canvas</td>
</tr>
<tr>
<td>gx_canvas_drawing_complete</td>
<td>Complete canvas drawing</td>
</tr>
<tr>
<td>gx_canvas_drawing_initiate</td>
<td>Initiate drawing on canvas</td>
</tr>
<tr>
<td>gx_canvas_ellipse_draw</td>
<td>Draw an ellipse</td>
</tr>
<tr>
<td>gx_canvas Hardware_layer_bind</td>
<td>Bind canvas to graphics layer</td>
</tr>
<tr>
<td>gx_canvas_hide</td>
<td>Make a canvas invisible</td>
</tr>
<tr>
<td>gx_canvas_line_draw</td>
<td>Draw line</td>
</tr>
<tr>
<td>gx_canvas_memory_define</td>
<td>Assign canvas memory address</td>
</tr>
<tr>
<td>gx_canvas_offset_set</td>
<td>Assign canvas x,y display offset</td>
</tr>
<tr>
<td>gx_canvas_pie_draw</td>
<td>Draw a pie (wedge) shape</td>
</tr>
<tr>
<td>gx_canvas_pixel_draw</td>
<td>Draw a single pixel</td>
</tr>
<tr>
<td>gx_canvas_pixelmap_blend</td>
<td>Blend a pixelmap with background</td>
</tr>
<tr>
<td>gx_canvas_pixelmap_get</td>
<td>Get a pixelmap pointing to canvas data</td>
</tr>
<tr>
<td>gx_canvas_pixelmap_draw</td>
<td>Draw pixelmap</td>
</tr>
<tr>
<td>gx_canvas_pixelmap_tile</td>
<td>Tile pixelmap</td>
</tr>
<tr>
<td>gx_canvas_polygon_draw</td>
<td>Draw polygon</td>
</tr>
<tr>
<td>gx_canvas_rectangle_draw</td>
<td>Draw rectangle</td>
</tr>
<tr>
<td>gx_canvas_rotated_text_draw</td>
<td>(deprecated) Draw text rotated about center point</td>
</tr>
<tr>
<td>gx_canvas_rotated_text_draw_ext</td>
<td>Draw text rotated about center point</td>
</tr>
<tr>
<td>gx_canvas_shift</td>
<td>Shift canvas by x,y</td>
</tr>
<tr>
<td>gx_canvas_show</td>
<td>Make a canvas visible</td>
</tr>
<tr>
<td>gx_canvas_text_draw</td>
<td>(deprecated) Draw text</td>
</tr>
<tr>
<td>gx_canvas_text_draw_ext</td>
<td>Draw text</td>
</tr>
<tr>
<td>gx_checkbox_create</td>
<td>Create a checkbox</td>
</tr>
<tr>
<td>gx_checkbox_draw</td>
<td>Draw a checkbox</td>
</tr>
<tr>
<td>gx_checkbox_event_process</td>
<td>Checkbox event process function</td>
</tr>
<tr>
<td>gx_checkbox_pixelmap_set</td>
<td>Assign checkbox pixelmap</td>
</tr>
<tr>
<td>gx_checkbox_select</td>
<td>Select checkbox</td>
</tr>
<tr>
<td>gx_circular_gauge_angle_get</td>
<td>Retrieve gauge widget needle angle</td>
</tr>
<tr>
<td>gx_circular_gauge_angle_set</td>
<td>Assign gauge widget needle angle</td>
</tr>
<tr>
<td>gx_circular_gauge_animation_set</td>
<td>Define circular gauge animation</td>
</tr>
<tr>
<td>gx_circular_gauge_background_draw</td>
<td>Draw circular gauge background</td>
</tr>
<tr>
<td>gx_circular_gauge_create</td>
<td>Create a circular gauge widget</td>
</tr>
<tr>
<td>gx_circular_gauge_draw</td>
<td>Draw a circular gauge widget</td>
</tr>
<tr>
<td>gx_circular_gauge_event_process</td>
<td>Process circular gauge event</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>gx_context_brush_default</code></td>
<td>Set the brush of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_define</code></td>
<td>Define brush of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_get</code></td>
<td>Get brush of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_pattern_set</code></td>
<td>Set pattern of the brush of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_set</code></td>
<td>Set brush of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_style_set</code></td>
<td>Set brush style of current context</td>
</tr>
<tr>
<td><code>gx_context_brush_width_set</code></td>
<td>Set brush width of current context</td>
</tr>
<tr>
<td><code>gx_context_color_get</code></td>
<td>Resolve a color ID to color value</td>
</tr>
<tr>
<td><code>gx_context_fill_color_set</code></td>
<td>Set fill color of current context</td>
</tr>
<tr>
<td><code>gx_context_font_get</code></td>
<td>Resolve a font ID to font pointer value</td>
</tr>
<tr>
<td><code>gx_context_font_set</code></td>
<td>Set font of current context</td>
</tr>
<tr>
<td><code>gx_context_line_color_set</code></td>
<td>Set line color of current context</td>
</tr>
<tr>
<td><code>gx_context_pixelmap_get</code></td>
<td>Resolve a pixelmap ID to pixelmap pointer value</td>
</tr>
<tr>
<td><code>gx_context_pixelmap_set</code></td>
<td>Assign brush pixelmap, used for area fills</td>
</tr>
<tr>
<td><code>gx_context_raw_brush_define</code></td>
<td>Define raw brush of current context</td>
</tr>
<tr>
<td><code>gx_context_raw_fill_color_set</code></td>
<td>Set raw fill color of current context</td>
</tr>
<tr>
<td><code>gx_context_raw_line_color_set</code></td>
<td>Set raw line color of current context</td>
</tr>
<tr>
<td><code>gx_context_string_get</code></td>
<td>Retrieve string associated with current drawing context (deprecated).</td>
</tr>
<tr>
<td><code>gx_context_string_get_ext</code></td>
<td>Retrieve string associated with current drawing context (deprecated).</td>
</tr>
<tr>
<td><code>gx_display_color_set</code></td>
<td>Replace one color value in display color table.</td>
</tr>
<tr>
<td><code>gx_display_color_table_set</code></td>
<td>Assign the color table used by a display</td>
</tr>
<tr>
<td><code>gx_display_create</code></td>
<td>Create display</td>
</tr>
<tr>
<td><code>gx_display_delete</code></td>
<td>Delete display</td>
</tr>
<tr>
<td><code>gx_display_font_table_set</code></td>
<td>Assign the font table used by a display</td>
</tr>
<tr>
<td><code>gx_display_language_table_get</code></td>
<td>Retrieve the language table associated with a display (deprecated)</td>
</tr>
<tr>
<td><code>gx_display_language_table_get_ext</code></td>
<td>Retrieve the language table associated with a display</td>
</tr>
<tr>
<td><code>gx_display_language_table_set</code></td>
<td>Assign the language table to indicated display (deprecated)</td>
</tr>
<tr>
<td><code>gx_display_language_table_set_ext</code></td>
<td>Assign the language table to indicated display.</td>
</tr>
<tr>
<td><code>gx_display_pixelmap_table_set</code></td>
<td>Assign the pixelmap table used by a display</td>
</tr>
<tr>
<td><code>gx_display_string_get</code></td>
<td>Retrieve string associated with string ID (deprecated)</td>
</tr>
<tr>
<td><code>gx_display_string_get_ext</code></td>
<td>Retrieve string associated with</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gx_display_string_table_get</td>
<td>Retrieve string table associated with indicated display (deprecated).</td>
</tr>
<tr>
<td>gx_display_string_table_get_ext</td>
<td>Retrieve string table associated with indicated display</td>
</tr>
<tr>
<td>gx_display_theme_install</td>
<td>Install themes to the specified display</td>
</tr>
<tr>
<td>gx_drop_list_close</td>
<td>Close drop list</td>
</tr>
<tr>
<td>gx_drop_list_create</td>
<td>Create drop list</td>
</tr>
<tr>
<td>gx_drop_list_open</td>
<td>Open drop list</td>
</tr>
<tr>
<td>gx_drop_list_popup_set</td>
<td>Set popup to drop list</td>
</tr>
<tr>
<td>gx_horizontal_list_children_position</td>
<td>Position children widgets in horizontal list</td>
</tr>
<tr>
<td>gx_horizontal_list_create</td>
<td>Create horizontal list</td>
</tr>
<tr>
<td>gx_horizontal_list_event_process</td>
<td>Process event in horizontal list</td>
</tr>
<tr>
<td>gx_horizontal_list_selected_index_get</td>
<td>Get the selected item index</td>
</tr>
<tr>
<td>gx_horizontal_list_selected_widget_get</td>
<td>Get the selected item widget</td>
</tr>
<tr>
<td>gx_horizontal_list_selected_set</td>
<td>Set the selected item</td>
</tr>
<tr>
<td>gx_horizontal_list_total_columns_set</td>
<td>Change number of list columns after creation</td>
</tr>
<tr>
<td>gx_horizontal_scrollbar_create</td>
<td>Create horizontal scrollbar</td>
</tr>
<tr>
<td>gx_icon_button_create</td>
<td>Create icon button</td>
</tr>
<tr>
<td>gx_icon_button_draw</td>
<td>Draw an icon button</td>
</tr>
<tr>
<td>gx_icon_button_pixelmap_set</td>
<td>Set pixelmap in icon button</td>
</tr>
<tr>
<td>gx_icon_background_draw</td>
<td>Draw icon background</td>
</tr>
<tr>
<td>gx_icon_create</td>
<td>Create icon</td>
</tr>
<tr>
<td>gx_icon_draw</td>
<td>Draw icon</td>
</tr>
<tr>
<td>gx_icon_event_process</td>
<td>Icon event processing function</td>
</tr>
<tr>
<td>gx_icon_pixelmap_set</td>
<td>Set pixelmap for icon</td>
</tr>
<tr>
<td>gx_image_reader_create</td>
<td>Create image reader module instance</td>
</tr>
<tr>
<td>gx_image_reader_palette_set</td>
<td>Define image reader palette</td>
</tr>
<tr>
<td>gx_image_reader_start</td>
<td>Start the decompress and conversion process</td>
</tr>
<tr>
<td>gx_line_chart_axis_draw</td>
<td>Draw line chart x,y axis</td>
</tr>
<tr>
<td>gx_line_chart_create</td>
<td>Create GX_LINE_CHART instance</td>
</tr>
<tr>
<td>gx_line_chart_data_draw</td>
<td>Draw line chart data line</td>
</tr>
<tr>
<td>gx_line_chart_draw</td>
<td>Default line chart drawing</td>
</tr>
<tr>
<td>gx_line_chart_update</td>
<td>Force update of line chart data</td>
</tr>
<tr>
<td>gx_line_chart_y_scale_calculate</td>
<td>Calculate scale of y axis data values to pixel coordinates.</td>
</tr>
<tr>
<td>gx_menu_create</td>
<td>Create menu</td>
</tr>
<tr>
<td>gx_menu_draw</td>
<td>Draw menu</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td><code>gx_menu_insert</code></td>
<td>Insert a new item</td>
</tr>
<tr>
<td><code>gx_menu_remove</code></td>
<td>Remove an item</td>
</tr>
<tr>
<td><code>gx_menu_text_draw</code></td>
<td>Draw menu text</td>
</tr>
<tr>
<td><code>gx_menu_text_offset_set</code></td>
<td>Set menu text draw offset</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_create</code></td>
<td>Create multi-line text button</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_draw</code></td>
<td>Draw multi-line text button</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_event_process</code></td>
<td>Set font for multi-line text button</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_text_draw</code></td>
<td>Text drawing portion of drawing</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_text_id_set</code></td>
<td>Set system string to text button</td>
</tr>
<tr>
<td><code>gx_multi_line_text_button_text_set_ext</code></td>
<td>Assign user-defined string to text button (deprecated)</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_backspace</code></td>
<td>Delete the character before multi-line text input cursor position</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_buffer_get</code></td>
<td>Retrieves buffer information of text input widget</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_buffer_clear</code></td>
<td>Deletes all characters from the text input buffer</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_char_insert</code></td>
<td>Insert UTF8-format string at multi-line text input cursor position (deprecated)</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_char_insert_ext</code></td>
<td>Insert UTF8-format string at multi-line text input cursor position</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_create</code></td>
<td>Create multi-line text input</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_cursor_pos_get</code></td>
<td>Retrieve multi-line text input cursor position</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_delete</code></td>
<td>Delete the character after multi-line text input cursor position</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_down_arrow</code></td>
<td>Move multi-line text input cursor to the next line</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_end</code></td>
<td>Move multi-line text input cursor to the end of the current line</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_event_process</code></td>
<td>Process multi-line text input text</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_fill_color_set</code></td>
<td>Set fill colors for multi line text input</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_home</code></td>
<td>Move multi-line text input cursor to the start of the current line</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_left_arrow</code></td>
<td>Move multi-line text input cursor left by one character</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_right_arrow</code></td>
<td>Move multi-line text input cursor right by one character</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_style_add</code></td>
<td>Add multi-line text style flags</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_style_remove</code></td>
<td>Remove multi-line text style flags</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_style_set</code></td>
<td>Assign multi-line text style flags</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_text_color_set</code></td>
<td>Assign text colors for multi line text input</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_text_set</code></td>
<td>Assign text to multi line text</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_text_set_ext</code></td>
<td>Assign user-defined string to text button (deprecated)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_text_set_ext</code></td>
<td>Assign text to multi line text input</td>
</tr>
<tr>
<td><code>gx_multi_line_text_input_up_arrow</code></td>
<td>Move multi line text input cursor to the previous line</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_create</code></td>
<td>Create multi-line text view</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_event_process</code></td>
<td>Process multi-line text view event</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_font_set</code></td>
<td>Set font used in multi line text view</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_line_space_set</code></td>
<td>Set multi-line text view line space</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_scroll_info_get</code></td>
<td>Get multi-line text view scroll info</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_text_color_set</code></td>
<td>Set text color in multi line text view</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_text_id_set</code></td>
<td>Set system text string in multi line text view</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_text_set</code></td>
<td>Set user-defined string to multi line text view</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_text_set_ext</code></td>
<td>Set user-defined string to multi line text view (deprecated)</td>
</tr>
<tr>
<td><code>gx_multi_line_text_view_whitespace_set</code></td>
<td>Set multi-line text view whitespace</td>
</tr>
<tr>
<td><code>gx_numeric_pixelmap_prompt_create</code></td>
<td>Create numeric pixelmap prompt</td>
</tr>
<tr>
<td><code>gx_numeric_pixelmap_prompt_format_function_set</code></td>
<td>Override format function of numeric pixelmap prompt</td>
</tr>
<tr>
<td><code>gx_numeric_pixelmap_prompt_value_set</code></td>
<td>Set numeric prompt value</td>
</tr>
<tr>
<td><code>gx_numeric_prompt_create</code></td>
<td>Create numeric prompt</td>
</tr>
<tr>
<td><code>gx_numeric_prompt_format_function_set</code></td>
<td>Override format function of numeric prompt</td>
</tr>
<tr>
<td><code>gx_numeric_prompt_value_set</code></td>
<td>Set numeric prompt value</td>
</tr>
<tr>
<td><code>gx_numeric_scroll_wheel_create</code></td>
<td>Create numeric scroll wheel widget</td>
</tr>
<tr>
<td><code>gx_numeric_scroll_wheel_range_set</code></td>
<td>Assign scroll wheel value range</td>
</tr>
<tr>
<td><code>gx_pixelmap_button_create</code></td>
<td>Create pixelmap button</td>
</tr>
<tr>
<td><code>gx_pixelmap_button_draw</code></td>
<td>Draw pixelmap button</td>
</tr>
<tr>
<td><code>gx_pixelmap_button_event_process</code></td>
<td>Pixelmap button event processing</td>
</tr>
<tr>
<td><code>gx_pixelmap_button_pixelmap_set</code></td>
<td>Set pixelmap in pixelmap button</td>
</tr>
<tr>
<td><code>gx_pixelmap_prompt_create</code></td>
<td>Create pixelmap prompt</td>
</tr>
<tr>
<td><code>gx_pixelmap_prompt_draw</code></td>
<td>Draw pixelmap prompt</td>
</tr>
<tr>
<td><code>gx_pixelmap_prompt_pixelmap_set</code></td>
<td>Set pixelmap in pixelmap prompt</td>
</tr>
<tr>
<td><code>gx_pixelmap_slider_create</code></td>
<td>Create pixelmap slider</td>
</tr>
<tr>
<td><code>gx_pixelmap_slider_draw</code></td>
<td>Draw pixelmap slider</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td><code>gx_pixelmap_slider_event_process</code></td>
<td>Pixelmap slider event processing</td>
</tr>
<tr>
<td><code>gx_pixelmap_slider_pixelmap_set</code></td>
<td>Set pixelmap in pixelmap slider</td>
</tr>
<tr>
<td><code>gx_progress_bar_background_draw</code></td>
<td>Draw progress bar background</td>
</tr>
<tr>
<td><code>gx_progress_bar_create</code></td>
<td>Create a progress bar</td>
</tr>
<tr>
<td><code>gx_progress_bar_draw</code></td>
<td>Draw a progress bar</td>
</tr>
<tr>
<td><code>gx_progress_bar_event_process</code></td>
<td>Process a progress bar event</td>
</tr>
<tr>
<td><code>gx_progress_bar_font_set</code></td>
<td>Set font of progress bar text</td>
</tr>
<tr>
<td><code>gx_progress_bar_info_set</code></td>
<td>Set progress bar information structure</td>
</tr>
<tr>
<td><code>gx_progress_bar_pixelmap_set</code></td>
<td>Set pixelmap used to draw progress bar</td>
</tr>
<tr>
<td><code>gx_progress_bar_range_set</code></td>
<td>Set value range of progress bar</td>
</tr>
<tr>
<td><code>gx_progress_bar_text_color_set</code></td>
<td>Set progress bar text color</td>
</tr>
<tr>
<td><code>gx_progress_bar_text_draw</code></td>
<td>Draw progress bar text</td>
</tr>
<tr>
<td><code>gx_progress_bar_value_set</code></td>
<td>Set progress bar value</td>
</tr>
<tr>
<td><code>gx_prompt_create</code></td>
<td>Create prompt</td>
</tr>
<tr>
<td><code>gx_prompt_draw</code></td>
<td>Draw prompt</td>
</tr>
<tr>
<td><code>gx_prompt_font_set</code></td>
<td>Set prompt font</td>
</tr>
<tr>
<td><code>gx_prompt_text_color_set</code></td>
<td>Set prompt text color</td>
</tr>
<tr>
<td><code>gx_prompt_text_draw</code></td>
<td>Text drawing portion of prompt draw</td>
</tr>
<tr>
<td><code>gx_prompt_text_get</code></td>
<td>Get prompt text (deprecated)</td>
</tr>
<tr>
<td><code>gx_prompt_text_get_ext</code></td>
<td>Get prompt text</td>
</tr>
<tr>
<td><code>gx_prompt_text_id_set</code></td>
<td>Set prompt with system text string</td>
</tr>
<tr>
<td><code>gx_prompt_text_set</code></td>
<td>Set prompt text (deprecated)</td>
</tr>
<tr>
<td><code>gx_prompt_text_set_ext</code></td>
<td>Set prompt text</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_anchor_set</code></td>
<td>Set starting angle</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_background_draw</code></td>
<td>Draw radial progress bar background</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_create</code></td>
<td>Create a radial progress bar</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_draw</code></td>
<td>Draw a radial progress bar</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_event_process</code></td>
<td>Process radial progress bar event</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_font_set</code></td>
<td>Set radial progress bar font</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_info_set</code></td>
<td>Set radial progress bar information</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_text_color_set</code></td>
<td>Set radial progress bar text color</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_text_draw</code></td>
<td>Draw radial progress bar text</td>
</tr>
<tr>
<td><code>gx_radial_progress_bar_value_set</code></td>
<td>Set radial progress bar value</td>
</tr>
<tr>
<td><code>gx_radio_button_create</code></td>
<td>Create radio button</td>
</tr>
<tr>
<td><code>gx_radio_button_draw</code></td>
<td>Draw radio button</td>
</tr>
<tr>
<td><code>gx_radio_button_pixelmap_set</code></td>
<td>Set pixelmap in radio button</td>
</tr>
<tr>
<td><code>gx_radial_slider_anchor_angles_set</code></td>
<td>Set radial slider anchor angle list</td>
</tr>
<tr>
<td><code>gx_radial_slider_angle_set</code></td>
<td>Set radial slider angle</td>
</tr>
<tr>
<td><code>gx_radial_slider_animation_set</code></td>
<td>Set radial slider animation info</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>gx_radial_slider_animation_start</code></td>
<td>Set radial slider angle with animation</td>
</tr>
<tr>
<td><code>gx_radial_slider_create</code></td>
<td>Create a radial slider</td>
</tr>
<tr>
<td><code>gx_radial_slider_draw</code></td>
<td>Draw a radial slider</td>
</tr>
<tr>
<td><code>gx_radial_slider_event_process</code></td>
<td>Process a radial slider event</td>
</tr>
<tr>
<td><code>gx_radial_slider_info_get</code></td>
<td>Retrieve radial slider information pointer</td>
</tr>
<tr>
<td><code>gx_radial_slider_info_set</code></td>
<td>Set radial slider information</td>
</tr>
<tr>
<td><code>gx_radial_slider_pixelmap_set</code></td>
<td>Set radial slider pixelmaps</td>
</tr>
<tr>
<td><code>gx_screen_stack_create</code></td>
<td>Create the GUIX screen stack control block and memory area.</td>
</tr>
<tr>
<td><code>gx_screen_stack_pop</code></td>
<td>Pop the top screen from the screen stack.</td>
</tr>
<tr>
<td><code>gx_screen_stack_push</code></td>
<td>Push the current screen to the screen stack.</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_create</code></td>
<td>Create base scroll wheel widget</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_event_process</code></td>
<td>Scroll wheel event processing</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_gradient_alpha_set</code></td>
<td>Modify scroll wheel overlay gradient</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_row_height_set</code></td>
<td>Assign scroll wheel row height</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_selected_background_set</code></td>
<td>Assign background image for selected row</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_selected_get</code></td>
<td>Retrieve selected row index</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_selected_set</code></td>
<td>Assign selected row index</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_speed_set</code></td>
<td>Assign scrolling speed</td>
</tr>
<tr>
<td><code>gx_scroll_wheel_total_rows_set</code></td>
<td>Assign total number of available rows</td>
</tr>
<tr>
<td><code>gx_scrollbar_draw</code></td>
<td>Draw scrollbar</td>
</tr>
<tr>
<td><code>gx_scrollbar_event_process</code></td>
<td>Process scrollbar event</td>
</tr>
<tr>
<td><code>gx_scrollbar_limit_check</code></td>
<td>Check scrollbar limit</td>
</tr>
<tr>
<td><code>gx_scrollbar_reset</code></td>
<td>Reset scrollbar</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_backspace</code></td>
<td>Handle backspace character</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_buffer_clear</code></td>
<td>Clear the character buffer</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_character_delete</code></td>
<td>Delete character</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_character_insert</code></td>
<td>Insert character</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_create</code></td>
<td>Create single-line text input</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_draw</code></td>
<td>Draw single-line text input widget</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_draw_position_get</code></td>
<td>Retrieve text draw start position</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_end</code></td>
<td>Move cursor to end</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_event_process</code></td>
<td>Text input event processing</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_fill_color_set</code></td>
<td>Set fill colors for single line text input</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_home</code></td>
<td>Move cursor to home</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_left_arrow</code></td>
<td>Handle left arrow key</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_position_get</code></td>
<td>Get cursor position</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_right_arrow</code></td>
<td>Handle right arrow key</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_style_add</code></td>
<td>Add style flags</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_style_remove</code></td>
<td>Remove style flags</td>
</tr>
<tr>
<td><code>gx_single_line_text_input_style_set</code></td>
<td>Assign style flags</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>gx_single_line_text_input_text_color_set</td>
<td>Set text colors for single line text input</td>
</tr>
<tr>
<td>gx_single_line_text_input_text_select</td>
<td>Select single line text input text</td>
</tr>
<tr>
<td>gx_single_line_text_input_text_set</td>
<td>Set single line text input text (deprecated)</td>
</tr>
<tr>
<td>gx_single_line_text_input_text_set_ext</td>
<td>Set single line text input text</td>
</tr>
<tr>
<td>gx_slider_create</td>
<td>Create slider</td>
</tr>
<tr>
<td>gx_slider_draw</td>
<td>Draw slider</td>
</tr>
<tr>
<td>gx_slider_event_process</td>
<td>Process slider event</td>
</tr>
<tr>
<td>gx_slider_info_set</td>
<td>Set slider information block</td>
</tr>
<tr>
<td>gx_slider_needle_draw</td>
<td>Draw slider needle</td>
</tr>
<tr>
<td>gx_slider_needle_position_get</td>
<td>Get slider needle position</td>
</tr>
<tr>
<td>gx_slider_tickmarks_draw</td>
<td>Draw slider tickmarks</td>
</tr>
<tr>
<td>gx_slider_travel_get</td>
<td>Get slider travel</td>
</tr>
<tr>
<td>gx_slider_value_calculate</td>
<td>Calculate slider value</td>
</tr>
<tr>
<td>gx_slider_value_set</td>
<td>Set slider value</td>
</tr>
<tr>
<td>gx_sprite_create</td>
<td>Create GX_SPRITE widget</td>
</tr>
<tr>
<td>gx_sprite_current_frame_set</td>
<td>Assign current display frame for sprite widget</td>
</tr>
<tr>
<td>gx_sprite_frame_list_set</td>
<td>Assign or modify a sprite frame list</td>
</tr>
<tr>
<td>gx_sprite_start</td>
<td>Start a sprite sequence</td>
</tr>
<tr>
<td>gx_sprite_stop</td>
<td>Stop a sprite sequence</td>
</tr>
<tr>
<td>gx_string_scroll_wheel_create</td>
<td>Create GX_STRING_SCROLL_WHEEL widget</td>
</tr>
<tr>
<td>gx_string_scroll_wheel_string_id_list_set</td>
<td>Assign array of String IDs</td>
</tr>
<tr>
<td>gx_string_scroll_wheel_string_list_set</td>
<td>Modify displayed string array</td>
</tr>
<tr>
<td>gx_string_scroll_wheel_text_get</td>
<td>Retrieve text for scroll wheel row</td>
</tr>
<tr>
<td>gx_studio_widget_create</td>
<td>Create widget defined within Studio</td>
</tr>
<tr>
<td>gx_studio_named_widget_create</td>
<td>Create screen defined within Studio</td>
</tr>
<tr>
<td>gx_studio_display_configure</td>
<td>Create and initialize GX_DISPLAY, GX_CANVAS, and GX_WINDOW_ROOT</td>
</tr>
<tr>
<td>gx_system_active_language_set</td>
<td>Assign active language ID</td>
</tr>
<tr>
<td>gx_system_canvas_refresh</td>
<td>Force refresh (drawing) of dirty canvases</td>
</tr>
<tr>
<td>gx_system_dirty_mark</td>
<td>Mark area dirty</td>
</tr>
<tr>
<td>gx_system_dirty_partial_add</td>
<td>Mark partial area dirty</td>
</tr>
<tr>
<td>gx_system_draw_context_get</td>
<td>Get drawing context</td>
</tr>
<tr>
<td>gx_system_event_fold</td>
<td>Fold event</td>
</tr>
<tr>
<td>gx_system_event_send</td>
<td>Send event</td>
</tr>
<tr>
<td>gx_system_focus_claim</td>
<td>Claim focus</td>
</tr>
<tr>
<td>gx_system_initialize</td>
<td>Initialize GUIX</td>
</tr>
<tr>
<td>gx_system_language_table_get</td>
<td>Retrieve language table</td>
</tr>
<tr>
<td>gx_system_language_table_set</td>
<td>Assign language table</td>
</tr>
<tr>
<td>gx_system_memory_allocator_set</td>
<td>Assign memory allocator/de-</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>gx_system_pen_configure</code></td>
<td>Set pen configuration</td>
</tr>
<tr>
<td><code>gx_system_screen_stack_create</code></td>
<td>Create screen stack control</td>
</tr>
<tr>
<td><code>gx_system_screen_stack_get</code></td>
<td>Retrieve screen stack pointers</td>
</tr>
<tr>
<td><code>gx_system_screen_stack_pop</code></td>
<td>Pop top screen from screen stack</td>
</tr>
<tr>
<td><code>gx_system_screen_stack_push</code></td>
<td>Push indicated screen to the screen stack</td>
</tr>
<tr>
<td><code>gx_system_screen_stack_reset</code></td>
<td>Reset the screen stack</td>
</tr>
<tr>
<td><code>gx_system_scroll_appearance_get</code></td>
<td>Get scroll appearance</td>
</tr>
<tr>
<td><code>gx_system_scroll_appearance_set</code></td>
<td>Set scroll appearance</td>
</tr>
<tr>
<td><code>gx_system_start</code></td>
<td>Start GUIX</td>
</tr>
<tr>
<td><code>gx_system_string_get</code></td>
<td>Get string</td>
</tr>
<tr>
<td><code>gx_system_string_table_get</code></td>
<td>Get string table</td>
</tr>
<tr>
<td><code>gx_system_string_table_set</code></td>
<td>Set string table</td>
</tr>
<tr>
<td><code>gx_system_string_width_get</code></td>
<td>Get string width (deprecated)</td>
</tr>
<tr>
<td><code>gx_system_string_width_get_ext</code></td>
<td>Get string width</td>
</tr>
<tr>
<td><code>gx_system_theme_install</code></td>
<td>Install font/color/pixelmap tables</td>
</tr>
<tr>
<td><code>gx_system_timer_start</code></td>
<td>Start timer</td>
</tr>
<tr>
<td><code>gx_system_timer_stop</code></td>
<td>Stop timer</td>
</tr>
<tr>
<td><code>gx_system_version_string_get</code></td>
<td>Retrieve GUIX library version string (deprecated)</td>
</tr>
<tr>
<td><code>gx_system_version_string_get_ext</code></td>
<td>Retrieve GUIX library version string</td>
</tr>
<tr>
<td><code>gx_system_widget_find</code></td>
<td>Find widget</td>
</tr>
<tr>
<td><code>gx_text_button_create</code></td>
<td>Create text button</td>
</tr>
<tr>
<td><code>gx_text_button_draw</code></td>
<td>Draw text button</td>
</tr>
<tr>
<td><code>gx_text_button_font_set</code></td>
<td>Set font for text button</td>
</tr>
<tr>
<td><code>gx_text_button_text_color_set</code></td>
<td>Set text button color</td>
</tr>
<tr>
<td><code>gx_text_button_text_draw</code></td>
<td>Text drawing portion of button drawing</td>
</tr>
<tr>
<td><code>gx_text_button_text_get</code></td>
<td>Get text used in text button (deprecated)</td>
</tr>
<tr>
<td><code>gx_text_button_text_get_ext</code></td>
<td>Get text used in text button</td>
</tr>
<tr>
<td><code>gx_text_button_text_id_set</code></td>
<td>Assign system string to text button</td>
</tr>
<tr>
<td><code>gx_text_button_text_set</code></td>
<td>Assign user-defined string to text button (deprecated)</td>
</tr>
<tr>
<td><code>gx_text_button_text_set_ext</code></td>
<td>Assign user-defined string to text button</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_callback_set</code></td>
<td>Assign string retrieval callback (deprecated)</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_callback_set_ext</code></td>
<td>Assign string retrieval callback</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_create</code></td>
<td>Create base text scroll wheel</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_draw</code></td>
<td>Textual scroll wheel drawing function</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_font_set</code></td>
<td>Assign text scroll wheel fonts</td>
</tr>
<tr>
<td><code>gx_text_scroll_wheel_text_color_set</code></td>
<td>Assign text scroll wheel text colors</td>
</tr>
<tr>
<td><code>gx_transition_window_create</code></td>
<td>Create a transition window</td>
</tr>
<tr>
<td>Function Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>gx_tree_view_create</code></td>
<td>Create a tree view</td>
</tr>
<tr>
<td><code>gx_tree_view_draw</code></td>
<td>Draw tree view</td>
</tr>
<tr>
<td><code>gx_tree_view_event_process</code></td>
<td>Process tree view event</td>
</tr>
<tr>
<td><code>gx_tree_view_indentation_set</code></td>
<td>Set tree view indentation</td>
</tr>
<tr>
<td><code>gx_tree_view_position</code></td>
<td>Position tree view items</td>
</tr>
<tr>
<td><code>gx_tree_view_root_line_color_set</code></td>
<td>Set tree view root line color</td>
</tr>
<tr>
<td><code>gx_tree_view_root_pixelmap_set</code></td>
<td>Set tree view root pixelmaps</td>
</tr>
<tr>
<td><code>gx_tree_view_selected_get</code></td>
<td>Retrieve selected item</td>
</tr>
<tr>
<td><code>gx_tree_view_selected_set</code></td>
<td>Set selected item</td>
</tr>
<tr>
<td><code>gx_utility_ltoa</code></td>
<td>Convert long integer to ASCII</td>
</tr>
<tr>
<td><code>gx_utility_math_acos</code></td>
<td>Compute arc cosine</td>
</tr>
<tr>
<td><code>gx_utility_math_asin</code></td>
<td>Compute arc sine</td>
</tr>
<tr>
<td><code>gx_utility_math_cos</code></td>
<td>Compute cosine</td>
</tr>
<tr>
<td><code>gx_utility_math_sin</code></td>
<td>Compute sine</td>
</tr>
<tr>
<td><code>gx_utility_math_sqrt</code></td>
<td>Compute square root</td>
</tr>
<tr>
<td><code>gx_utility_pixelmap_resize</code></td>
<td>Resize pixelmap</td>
</tr>
<tr>
<td><code>gx_utility_pixelmap_rotate</code></td>
<td>Rotate pixelmap</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_center</code></td>
<td>Center rectangle inside another rectangle</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_center_find</code></td>
<td>Find center of rectangle</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_combine</code></td>
<td>Combine two rectangles into first</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_compare</code></td>
<td>Compare two rectangles</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_define</code></td>
<td>Define rectangle</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_resize</code></td>
<td>Resize rectangle</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_overlap_detect</code></td>
<td>Detect overlap of rectangles</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_point Detect_point</code></td>
<td>Detect if point resides in rectangle</td>
</tr>
<tr>
<td><code>gx_utility_rectangle_shift</code></td>
<td>Shift rectangle</td>
</tr>
<tr>
<td><code>gx_utility_string_to_alphamap</code></td>
<td>Render text string to alphamap (deprecated)</td>
</tr>
<tr>
<td><code>gx_utility_string_to_alphamap_ext</code></td>
<td>Render text string to alphamap</td>
</tr>
<tr>
<td><code>gx_vertical_list_children_position</code></td>
<td>Position children in vertical list</td>
</tr>
<tr>
<td><code>gx_vertical_list_create</code></td>
<td>Create vertical list</td>
</tr>
<tr>
<td><code>gx_vertical_list_event_process</code></td>
<td>Process vertical list event</td>
</tr>
<tr>
<td><code>gx_vertical_list_selected_index_get</code></td>
<td>Get selected item index</td>
</tr>
<tr>
<td><code>gx_vertical_list_selected_widget_get</code></td>
<td>Get selected widget</td>
</tr>
<tr>
<td><code>gx_vertical_list_selected_set</code></td>
<td>Set entry in vertical list</td>
</tr>
<tr>
<td><code>gx_vertical_list_total_rows_set</code></td>
<td>Change number of list rows after creation</td>
</tr>
<tr>
<td><code>gx_vertical_scrollbar_create</code></td>
<td>Create vertical scrollbar</td>
</tr>
<tr>
<td><code>gx_widget_allocate</code></td>
<td>Dynamically allocate a widget</td>
</tr>
<tr>
<td><code>gx_widget_attach</code></td>
<td>Attach widget to parent</td>
</tr>
<tr>
<td><code>gx_widget_background_draw</code></td>
<td>Draw widget background</td>
</tr>
<tr>
<td><code>gx_widget_back_attach</code></td>
<td>Attach widget in back</td>
</tr>
<tr>
<td><code>gx_widget_back_move</code></td>
<td>Move widget to back</td>
</tr>
<tr>
<td><code>gx_widget_block_move</code></td>
<td>Move block of pixels</td>
</tr>
<tr>
<td><code>gx_widget_border_draw</code></td>
<td>Draw widget border</td>
</tr>
<tr>
<td><code>gx_widget_border_style_set</code></td>
<td>Set widget border style</td>
</tr>
<tr>
<td><code>gx_widget_border_width_get</code></td>
<td>Set widget border width</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gx_widget_canvas_get</td>
<td>Get widget canvas</td>
</tr>
<tr>
<td>gx_widget_child_detect</td>
<td>Detect widget child</td>
</tr>
<tr>
<td>gx_widget_children_draw</td>
<td>Draw widget children</td>
</tr>
<tr>
<td>gx_widget_client_get</td>
<td>Get widget client area</td>
</tr>
<tr>
<td>gx_widget_color_get</td>
<td>Resolve color ID to color value for a visible widget</td>
</tr>
<tr>
<td>gx_widget_create</td>
<td>Create widget</td>
</tr>
<tr>
<td>gx_widget_created_test</td>
<td>Test if widget created</td>
</tr>
<tr>
<td>gx_widget_delete</td>
<td>Delete widget</td>
</tr>
<tr>
<td>gx_widget_detach</td>
<td>Detach widget from parent</td>
</tr>
<tr>
<td>gx_widget_draw</td>
<td>Draw widget</td>
</tr>
<tr>
<td>gx_widget_draw_set</td>
<td>Set draw function of widget</td>
</tr>
<tr>
<td>gx_widget_event_generate</td>
<td>Generate widget event</td>
</tr>
<tr>
<td>gx_widget_event_process</td>
<td>Process widget event</td>
</tr>
<tr>
<td>gx_widget_event_process_set</td>
<td>Set event processing function of widget</td>
</tr>
<tr>
<td>gx_widget_event_to_parent</td>
<td>Send event to widget's parent</td>
</tr>
<tr>
<td>gx_widget_fill_color_set</td>
<td>Assign widget fill color</td>
</tr>
<tr>
<td>gx_widget_find</td>
<td>Find widget</td>
</tr>
<tr>
<td>gx_widget_first_child_get</td>
<td>Return pointer to first child</td>
</tr>
<tr>
<td>gx_widget_focus_next</td>
<td>Move input focus to next widget</td>
</tr>
<tr>
<td>gx_widget_focus_previous</td>
<td>Move input focus to previous widget</td>
</tr>
<tr>
<td>gx_widget_font_get</td>
<td>Resolve font ID to a font pointer for a visible widget</td>
</tr>
<tr>
<td>gx_widget_free</td>
<td>Free widget control block memory</td>
</tr>
<tr>
<td>gx_widget_front_move</td>
<td>Move widget to front</td>
</tr>
<tr>
<td>gx_widget_height_get</td>
<td>Get widget height</td>
</tr>
<tr>
<td>gx_widget_hide</td>
<td>Hide widget</td>
</tr>
<tr>
<td>gx_widget_last_child_get</td>
<td>Return pointer to last child</td>
</tr>
<tr>
<td>gx_widget_next_sibling_get</td>
<td>Return pointer to next sibling</td>
</tr>
<tr>
<td>gx_widget_parent_get</td>
<td>Return pointer to parent widget</td>
</tr>
<tr>
<td>gx_widget_pixelmap_get</td>
<td>Resolve pixelmap ID to a pixelmap pointer for a visible widget</td>
</tr>
<tr>
<td>gx_widget_previous_sibling_get</td>
<td>Return pointer to previous sibling</td>
</tr>
<tr>
<td>gx_widget_resize</td>
<td>Resize widget</td>
</tr>
<tr>
<td>gx_widget_shift</td>
<td>Shift widget</td>
</tr>
<tr>
<td>gx_widget_show</td>
<td>Show widget</td>
</tr>
<tr>
<td>gx_widget_status_add</td>
<td>Add widget status</td>
</tr>
<tr>
<td>gx_widget_status_get</td>
<td>Retrieve widget status flags</td>
</tr>
<tr>
<td>gx_widget_status_remove</td>
<td>Remove widget status</td>
</tr>
<tr>
<td>gx_widget_string_get</td>
<td>Retrieve string associated with string ID for visible widget (deprecated)</td>
</tr>
<tr>
<td>gx_widget_string_get_ext</td>
<td>Retrieve string associated with string ID for visible widget.</td>
</tr>
<tr>
<td>gx_widget_status_test</td>
<td>Test widget status</td>
</tr>
<tr>
<td>gx_widget_style_add</td>
<td>Add widget style</td>
</tr>
<tr>
<td>gx_widget_style_get</td>
<td>Retrieve widget style flags</td>
</tr>
<tr>
<td>gx_widget_style_remove</td>
<td>Remove widget style</td>
</tr>
<tr>
<td>gx_widget_style_set</td>
<td>Set widget style</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>gx_widget_text_blend</code></td>
<td>Render blended text over widget (deprecated)</td>
</tr>
<tr>
<td><code>gx_widget_text_blend_ext</code></td>
<td>Render blended text over widget</td>
</tr>
<tr>
<td><code>gx_widget_text_draw</code></td>
<td>Render text over widget (deprecated)</td>
</tr>
<tr>
<td><code>gx_widget_text_draw_ext</code></td>
<td>Render text over widget</td>
</tr>
<tr>
<td><code>gx_widget_text_id_draw</code></td>
<td>Render text identified by string ID over widget</td>
</tr>
<tr>
<td><code>gx_widget_top_visible_child_find</code></td>
<td>Return pointer to visible child that is drawn at the top of the Z order</td>
</tr>
<tr>
<td><code>gx_widget_type_find</code></td>
<td>Find widget type</td>
</tr>
<tr>
<td><code>gx_widget_width_get</code></td>
<td>Get widget width</td>
</tr>
<tr>
<td><code>gx_window_canvas_set</code></td>
<td>Set window canvas</td>
</tr>
<tr>
<td><code>gx_window_client_height_get</code></td>
<td>Get window client height</td>
</tr>
<tr>
<td><code>gx_window_client_scroll</code></td>
<td>Scroll window clients</td>
</tr>
<tr>
<td><code>gx_window_client_width_get</code></td>
<td>Get window client width</td>
</tr>
<tr>
<td><code>gx_window_close</code></td>
<td>Terminate modal window execution</td>
</tr>
<tr>
<td><code>gx_window_create</code></td>
<td>Create window</td>
</tr>
<tr>
<td><code>gx_window_draw</code></td>
<td>Draw window</td>
</tr>
<tr>
<td><code>gx_window_event_process</code></td>
<td>Process window event</td>
</tr>
<tr>
<td><code>gx_window_execute</code></td>
<td>Modal window execution</td>
</tr>
<tr>
<td><code>gx_window_root_create</code></td>
<td>Create root window</td>
</tr>
<tr>
<td><code>gx_window_root_delete</code></td>
<td>Destroy root window</td>
</tr>
<tr>
<td><code>gx_window_root_event_process</code></td>
<td>Process event for root window</td>
</tr>
<tr>
<td><code>gx_window_root_find</code></td>
<td>Find root window</td>
</tr>
<tr>
<td><code>gx_window_scroll_info_get</code></td>
<td>Get window scroll info</td>
</tr>
<tr>
<td><code>gx_window_scrollbar_find</code></td>
<td>Find window scrollbar</td>
</tr>
<tr>
<td><code>gx_window_wallpaper_get</code></td>
<td>Get window wallpaper</td>
</tr>
<tr>
<td><code>gx_window_wallpaper_set</code></td>
<td>Set window wallpaper</td>
</tr>
</tbody>
</table>
gx_accordion_menu_create

Create an accordion menu

Prototype

UINT gx_accordion_menu_create(GX_ACCORDION_MENU *accordion,
GX_CONST GX_CHAR *name, GX_WIDGET *parent, ULONG style,
USHORT accordion_menu_id, GX_CONST GX_RECTANGLE *size);

Description

This service creates an accordion menu as specified and attaches
the accordion menu to the supplied parent widget. An accordion
menu is an expanding/collapsing menu display widget. It accepts all
types of widget as its child menu items. Accordion menus can be
nested, meaning several levels of menu depth can be created.

To insert a child item into a menu item widget, it’s recommended to
use GX_MENU type widget as a parent menu item.

Tips for creating a single level accordion menu:

1. Create an accordion menu.
2. Attach GX_MENU type widgets to the accordion menu.
3. Attach child widgets to the GX_MENU type parent. The
   child item type can be any GUIX widget type.

Tips for creating multi level accordion menu:

1. Create an accordion menu.
2. Attach GX_MENU type widgets to the accordion menu.
3. Attach GX_ACCORDION_MENU type widget to the
   GX_MENU type parent.
4. Attach menu items to the GX_ACCORDION_MENU type
   parent as described in the single level accordion menu
   creation.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accordion</strong></td>
<td>Pointer to accordion menu control block</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Name of the accordion menu</td>
</tr>
<tr>
<td><strong>parent</strong></td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td><strong>style</strong></td>
<td>Style of the widget. <strong>Appendix D</strong> contains pre-defined general styles for all widgets as well as widget specific styles.</td>
</tr>
<tr>
<td><strong>accordion_menu_id</strong></td>
<td>Application-defined ID of the accordion menu</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>Size of the accordion menu</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful accordion menu creation</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>Invalid widget control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_ACCORDION_MENU my_accordion;
GX_MENU menu_1;
GX_MENU item_1;
GX_RECTANGLE size;

gx_utility_rectangle_define(&size, 100, 100, 300, 150);

status = gx accordion_menu_create(&my_accordion, “my_accordion”,
parent, GX_STYLE_ENABLED,
MY_ACCORDION_ID, &size);

gx_menu_create(&menu_1, “menu_1”, &my_accordion,
GX_STRING_ID_MENU_1, GX_ID_NONE,
GX_STYLE_ENABLED | GX_TYLE_BORDER_THIN, 0, &size);

gx_menu_create(&item_1, “item_1”, &my_accordion,
GX_STRING_ID_ITEM_1, GX_ID_NONE,
GX_STYLE_ENABLED | GX_STYLE_BORDER_THIN, 0, &size);

gx_text_offset_set(&item_1, 30, 0);

gx_menu_insert(&menu_1, &item_1);
/* If status is GX_SUCCESS the accordion menu was successfully
created. */

The demo application demo_guix_widget_types, provided as part of
the GUIX Studio installation, provides a complete example of using
the accordion menu widget.

See Also

gx accordion_menu_draw, gx accordion_menu_event_process,
gx accordion_menu_position, gx_menu_create, gx_menu_draw, gx_menu_insert,
gx_menu_remove, gx_menu_text_draw, gx_menu_text_offset_set
**gx_accordion_menu_draw**

**Draw accordion menu**

**Prototype**

```c
VOID gx_accordion_menu_draw(GX_ACCORDION_MENU *accordion);
```

**Description**

This service draws the specified accordion menu. This service is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom accordion menu widgets.

**Parameters**

`accordion`  
Pointer to accordion menu control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Define a custom accordion menu draw function */

VOID my_accordion_menu_draw(GX_ACCORDION_MENU *accordion)
{
    /* Call default accordion menu draw. */
    gx_accordion_menu_draw(accordion);
    /* Add custom drawing here. */
}
```

**See Also**

`gx_accordion_menu_create`, `gx_accordion_menu_event_process`,  
`gx_accordion_menu_position`, `gx_menu_create`, `gx_menu_draw`, `gx_menu_insert`,  
`gx_menu_remove`, `gx_menu_text_draw`, `gx_menu_text_offset_set`
**gx_accordion_menu_event_process**

Process accordion menu event

**Prototype**

```c
UINT gx_accordion_menu_event_process(GX_ACCORDION_MENU *accordion,
                                       GX_EVENT *event_ptr);
```

**Description**

This service processes an event for the specified accordion menu. This service should be called as the default event handler by any custom accordion menu event processing functions.

This service handles GX_EVENT_PEN_DOWN and GX_EVENT_PEN_UP events to expand/collaps a menu item.

**Parameters**

- `accordion` Pointer to accordion menu control block
- `event_ptr` Pointer to the event to process

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful accordion menu event process</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>
Allowed From

Threads

Example

/* Call generic accordion menu event processing as part of custom event processing function. */

UINT custom_accordion_event_process(GX_ACCORDION_MENU *accordion, 
                                       GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
        case xyz:
            /* Insert custom event handling here */
            break;
        default:
            /* Pass all other events to the default accordion menu event processing */
            status = gxAccordionMenuEventProcess(accordion, event);
            break;
    }
    return status;
}

See Also

gxAccordionMenuCreate, gxAccordionMenuDraw,
gxAccordionMenuPosition, gx_menu_create, gx_menu_draw, gx_menu_insert,
gx_menu_remove, gx_menu_text_draw, gx_menu_text_offset_set
gx_accordion_menu_position

Position menu items

Prototype

UINT gx_accordion_menu_position(GX_ACCORDION_MENU *accordion);

Description

This service positions the menu items for the accordion menu. This function is normally called internally when the accordion menu is becoming visible. If you want to insert/remove items to/from an accordion menu, or change the expand styles of the child item, this function should be called to reposition the child items.

Parameters

accordion Pointer to accordion menu control block

Return Values

GX_SUCCESS (0x00) Successful accordion menu position
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

/* Position menu items in the accordion menu "my_accordion" */
status = gx_accordion_menu_position(&my_accordion);

/* If status is GX_SUCCESS the children in the accordion menu
"my_accordion" are positioned. */

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_menu_create, gx_menu_draw,
gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set
**gx_animation_canvas_define**

Provide canvas memory to an animation controller

**Prototype**

```c
UINT gx_animation_canvas_define(GX_ANIMATION *animation,
                                GX_CANVAS *canvas)
```

**Description**

This service provides a memory canvas to an animation controller used to implement the animation sequence. This provided canvas should be large enough to hold the animation target widget.

When an animation canvas is defined, the target widget is drawn once to this animation canvas, and the screen slide or fade effect is accomplished by modifying the canvas offset and/or canvas alpha value. When hardware support for multiple graphics layers is provided, defining an animation canvas that is bound to a hardware graphics overlay layer can greatly improve the performance of slide and fade animations.

The animation manager does require an animation canvas to execute fade-in and fade-out animation types if running at color depth less than 16 bpp.

**Parameters**

- **animation**: Pointer to animation control block
- **canvas**: Memory canvas used to implement the translation animation.

**Return Values**

- **GX_SUCCESS** (0x00): Successfully defined animation canvas
- **GX_INVALID_STATUS** (0x10): Invalid animation status
- **GX_INVALID_MEMORY_SIZE** (0x29): The provided memory block is not large enough to create the canvas
- **GX_PTR_ERROR** (0x07): Invalid pointer
Allowed From
Initialization and threads

Example

```c
GX_ANIMATION        animation;
GX_CANVAS           animation_canvas;
GX_ROOT_WINDOW      animation_root;

/* Create animation canvas. */
status = gx_canvas_create(
    &animation_canvas, /* Canvas control block. */
    "animation_canvas", /* Name of canvas. */
    display, /* Display control block. */
    GX_ANIMATION_MANAGED, /* Type of canvas. */
    width, /* Width of canvas. */
    height, /* Height of canvas. */
    memory_area, /* Memory area of canvas. */
    memory_size /* Size of canvas memory. */
);

if (status == GX_SUCCESS)
{
    /* Create the root window for the canvas. */
    status = gx_window_root_create(
        &animation_root, /* Root window control block. */
        "animation_root", /* Name of root window. */
        &animation_canvas, /* Canvas of the root window. */
        GX_STYLE_NONE, /* Style of the window. */
        GX_ID_NONE, /* Root window ID. */
        &root_size /* Window size. */
    );
}

if (status == GX_SUCCESS)
{
    /* Define canvas for the animation. */
    status = gx_animation_canvas_define(&animation,
                                         &animation_canvas);
}

/* If status is GX_SUCCESS the new canvas was successfully set to
the animation manager. */

See Also

gx_animation_create, gx_animation_delete, gx_animation_drag_disable,
gx_animation_drag_enable, gx_animation_landing_speed_set,
gx_animation_start, gx_animation_stop
```
**gx_animation_create**

Create an animation controller

Prototype

```c
UINT  gx_animation_create(GX_ANIMATION *animation);
```

Description

This service creates an animation controller. The controller is initialized to the idle state. One cannot start an animation unless it is in the IDLE state. The GX_ANIMATION control block pointer may be obtained using `gx_system_animation_get()`, or it may be a statically defined control block.

Parameters

- **animation**  
  Pointer to animation control block

Return Values

- **GX_SUCCESS**  
  (0x00) Successfully created animation controller
- **GX_ALREADY_CREATED**  
  (0x13) Control block already initialized
- **GX_PTR_ERROR**  
  (0x07) Invalid pointer

Allowed From

- Initialization and threads
Example

GX_ANIMATION *animation;

/* Allocate an animation control from system pool */
gx_system_animation_get(&animation);

/* Initialize the control block */
if (animation)
{
    status = gx_animation_create(&animation);
}

/* If status is GX_SUCCESS the new animation controller was
   successfully created and initialized. */

See Also

gx_animation_canvas_define, gx_animation_delete, gx_animation_drag_disable,
gx_animation_drag_enable, gx_animation_start,
gx_animation_landing_speed_set, gx_animation_stop, gx_system_animation_get,
gx_system_animation_free
**gx_animation_drag_disable**

Disable screen drag animation hook

**Prototype**

```c
UINT gx_animation_drag_disable(GX_ANIMATION *animation, 
                               GX_WIDGET *widget);
```

**Description**

This service removes the screen drag animation hook procedure from the widget’s default event process function and stops the animation sequence. The screen drag animation hook procedure handles events for a screen drag animation.

**Parameters**

- **animation**: Pointer to animation control block
- **widget**: Pointer to widget control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

**Allowed From**

Initialization and threads

**Example**

```c
GX_ANIMATION  animation;
GX_WIDGET      *animation_parent;

/* Disable screen drag animation of "animation_parent". */
status = gx_animation_drag_disable(&animation, animation_parent);

/* If status is GX_SUCCESS the screen drag hook has been removed from the event process of "animation_parent". */
```

**See Also**

gx_animation_canvas_define, gx_animation_create, gx_animation_drag_enable, 
gx_animation_landing_speed_set, gx_animation_start, gx_animation_stop, 
gx_system_animation_get, gx_system_animation_free
**gx_animation_drag_enable**

Enable screen drag animation hook

**Prototype**

```c
UINT gx_animation_drag_enable(GX_ANIMATION *animation, GX_WIDGET *widget, GX_ANIMATION_INFO *info);
```

**Description**

This service sets the internally defined screen drag animation event process function as a hook procedure of a widget’s default event process function. The screen drag animation event process function handles events for a screen drag animation.

The screen drag hook procedure becomes the default handler for pen input events sent to the target widget. The original widget event processing function is called in a daisy-chain fashion after checking for screen drag input event types.

**Parameters**

- **animation**: Pointer to animation control block
- **widget**: Pointer to widget control block
- **info**: Animation information

**Return Values**

- **GX_SUCCESS** (0x00) Successful
- **GX_INVALID_STATUS** (0x26) Invalid animation status
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid value
- **GX_INVALID_WIDGET** (0x12) Slide screen list is not provided

**Allowed From**

Initialization and threads
Example

```c
GX_ANIMATION animation;
GX_ANIMATION_INFO info;
GX_WIDGET *animation_parent;
GX_WIDGET *screen_list[] = {
    screen_1,
    screen_2,
    screen_3,
    GX_NULL
};

memset(&info, 0, sizeof(GX_ANIMATION_INFO));
info.gx_animation_parent = animation_parent;

/* If GX_STYLE_ANIMATION_WRAP is set, the screen list will wrap itself. */
info.gx_animation_style = GX_ANIMATION_SCREEN_DRAG |
                         GX_ANIMATION_HORIZONTAL |
                         GX_STYLE_ANIMATION_WRAP;
info.gx_animation_frame_interval = 1;
info.gx_animation_slide_screen_list = screen_list;

status = gx_animation_drag_enable(&animation, animation_parent, info);

/* If status is GX_SUCCESS the screen slide animation event
process function has been set as a hook procedure of
"animation_parent". */
```

See Also

- `gx_animation_canvas_define`
- `gx_animation_create`
- `gx_animation_drag_disable`
- `gx_animation_landing_speed_set`
- `gx_animation_start`
- `gx_animation_stop`
- `gx_system_animation_get`
- `gx_system_animation_free`
**gx_animation_landing_speed_set**

Set landing speed for screen drag animation

**Prototype**

```c
UINT  gx_animation_landing_speed_set(GX_ANIMATION *animation,
                                            USHORT shift_per_step);
```

**Description**

This service sets the landing speed for screen drag animation.

**Parameters**

- **animation**: Pointer to animation control block
- **shift_per_step**: Shift distance for each step

**Return Values**

- **GX_SUCCESS** (0x00): Successful
- **GX_INVALID_VALUE** (0x22): Invalid parameter

**Allowed From**

Initialization and threads

**Example**

```c
/* Set landing speed of "my_animation" to 20. */
status = gx_animation_landing_speed_set(&my_animation, 20);

/* If status is GX_SUCCESS the landing speed is successfully set to 20. */
```

**See Also**

- `gx_animation_canvas_define`, `gx_animation_create`, `gx_animation_slide_disable`,
- `gx_animation_slide_enable`, `gx_animation_start`, `gx_animation_stop`,
- `gx_system_animation_get`, `gx_system_animation_free`
**gx_animation_start**

Start a timer-driven animation

**Prototype**

```c
UINT  gx_animation_start(GX_ANIMATION *animation,
                           GX_ANIMATION_INFO *params);
```

**Description**

This service initiates an animation sequence using a previously created animation instance and a new set of animation parameters. This function makes a local copy of the parameters, meaning the parameter structure does not need to be statically defined.

The GX_ANIMATION control structure can be statically defined by the application, or it can be obtained using the gx_system_animation_get() API.

The GX_ANIMATION_INFO structure defines the parameters of the animation to be executed. For a complete description of this structure and the meaning of each field, refer to the GUIX Animation Component section in Chapter 3 of this manual.

**Parameters**

- **animation** Pointer to animation control block
- **params** Pointer to parameter structure

**Return Values**

- **GX_SUCCESS** (0x00) Successful
- **GX_INVALID_VALUE** (0x22) Invalid parameter
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Invalid animation target
- **GX_INVALID_STATUS** (0x26) Invalid animation status
- **GX_INVALID_CANVAS** (0x20) Invalid animation canvas

**Allowed From**

- Initialization and threads
Example

GX_ANIMATION_INFO params;
GX_ANIMATION *animation;

/* obtain an animation control block pointer */
gx_system_animation_get(&animation);
if (animation)
{
    /* define a slide down and to the right */
    params.gx_animation_start_position.gx_point_x = 0;
    params.gx_animation_start_position.gx_point_y = 0;
    params.gx_animation_end_position.gx_point_x = 100;
    params.gx_animation_end_position.gx_point_y = 200;
    params.gx_animation_style = GX_ANIMATION_TRANSLATE;
    params.gx_animation_target = &my_window;
    params.gx_animation_parent = root_window;
    params.gx_animation_start_alpha = 255;
    params.gx_animation_end_alpha = 255;
    params.gx_animation_delay_before = 0;
    params.gx_animation_steps = 10;
    params.gx_animation_tick_rate = 2;

    status = gx_animation_start(&animation, &params);
}

/* If status is GX_SUCCESS the animation is initiated. */

See Also

gx_animation_canvas_define, gx_animation_create, gx_animation_slide_disable,
gx_animation_slide_enable, gx_animation_landing_speed_set,
gx_animation_stop, gx_system_animation_get, gx_system_animation_free
**gx_animation_stop**

Stop an active timer-driven animation

**Prototype**

```c
UINT gx_animation_stop(GX_ANIMATION *animation);
```

**Description**

Stop a previously started animation. If the animation control block pointer was allocated using `gx_system_animation_get`, the application can re-use the control block or return it to the system pool using `gx_system_animation_free()`

**Parameters**

- `animation` Pointer to animation control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_STATUS** (0x26) Invalid controller status

**Allowed From**

Initialization and threads

**Example**

```c
GX_ANIMATION animation;

status = gx_animation_stop(&animation);

/* If status is GX_SUCCESS the animation is stopped */
```

**See Also**

- `gx_animation_canvas_define`, `gx_animation_create`, `gx_animation_drag_disable`,
- `gx_animation_drag_enable`, `gx_animation_start`, `gx_system_animation_get`,
- `gx_system_animation_free`
**gx_binres_language_table_load**

Load language table resource (deprecated)

**Prototype**

```c
UINT gx_binres_language_table_load(GX_UBYTE *root_address,
                                    GX_UBYTE ****returned_language_table);
```

**Description**

This deprecated API allows applications to load string table data from older (prior to release 5.6) binary resource data files.

New applications should use `gx_binres_language_table_load_ext()`.

This service builds up a language table structure containing pointers to table resources, the generated data structures point to the resource data “in place”, it does not copy the resource data. The resource data must be placed in a general access memory location, and the base address of this memory location is passed to this API.

This service requires a runtime allocated memory block sufficient in size to hold the language table structure, and therefore the `gx_system_memory_allocator_set` API must be invoked once before this service is requested.

The returned language table defines one or more string table(s), each string table containing pointers to string resources in resource data memory.

**Parameters**

- `root_address`: Address of binary resource data in memory
- `returned_language_table`: Pointer to loaded language table

**Return Values**

- `GX_SUCCESS` (0x00): Successful
- `GX_INVALID_FORMAT` (0x24): Invalid binary resource
- `GX_PTR_ERROR` (0x07): Invalid pointer
- `GX_SYSTEM_MEMPRY_ERROR` (0x30): Memory allocator or free function is not defined

**Allowed From**
Initialization and threads

Example

GX_UBYTE ***language_table = GX_NULL;

/* Specify address that binary resource data is located. */
GX_UBYTE   *root_address = 0x60000000;

status = gx_binres_language_table_load(root_address,
           &language_table);

/* If status is GX_SUCCESS, the language table was
 successfully loaded. */

See Also

gx_binres_language_table_load_ext
**gx_binres_language_table_load_ext**

Load language table resource

**Prototype**

```c
UINT gx_binres_language_table_load_ext(GX_UBYTE *root_address, GX_STRING ***returned_language_table);
```

**Description**

This service builds up a language table structure containing pointers to table resources, the generated data structures point to the resource data “in place”, it does not copy the resource data. The resource data must be placed in a general access memory location, and the base address of this memory location is passed to this API.

This service requires a runtime allocated memory block sufficient in size to hold the language table structure, and therefore the `gx_system_memory_allocator_set` API must be invoked once before this service is requested.

The returned language table defines one or more string table(s), each string table containing pointers to string resources in resource data memory.

**Parameters**

- **root_address**: Address of binary resource data in memory
- **returned_language_table**: Pointer to loaded language table

**Return Values**

- **GX_SUCCESS** (0x00): Successful
- **GX_INVALID_FORMAT** (0x24): Invalid binary resource
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_SYSTEM_MEMPRY_ERROR** (0x30): Memory allocator or free function is not defined

**Allowed From**

Initialization and threads
Example

GX_STRING **language_table = GX_NULL;

/* Specify address that binary resource data is located. */
GX_UBYTE   *root_address = 0x60000000;

status = gx_binres_language_table_load_ext(root_address,
                                           &language_table);

/* If status is GX_SUCCESS, the language table was successfully loaded. */

See Also

gx_binres_theme_load
### gx_binres_theme_load

**Load theme resource**

**Prototype**

```c
UINT gx_binres_theme_load(GX_UBYTE *root_address, INT theme_id,
GX_THEME **returned_theme);
```

**Description**

This service builds up a GX_THEME structure containing pointers to the resource tables for the requested theme. The generated data structures point to the resource data “in place”, it does not copy the resource data. So the resource data must be placed in a general access memory location, and the base address of this memory location is passed to this API.

This service requires a runtime allocated memory block sufficient in size to hold the theme table structure, and therefore the gx_system_memory_allocator_set API must be invoked once before this service is requested.

**Parameters**

- `root_address`: Address of binary resource data in memory
- `theme_id`: The identifier of the theme
- `returned_theme`: Pointer to loaded theme

**Return Values**

- **GX_SUCCESS** (0x00) Successful
- **GX_INVALID_FORMAT** (0x24) Invalid binary resource
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid theme ID
- **GX_SYSTEM_MEMORY_ERROR** (0x30) Memory allocator or free function is not defined

**Allowed From**

Initialization and threads
Example

GX_CHAR   *theme = GX_NULL;
GX_UBYTE  *root_address = 0x60000000;
INT       theme_id = 0;

status = gx_binres_theme_load(root_address, theme_id, &theme);

/* If status is GX_SUCCESS, the theme was successfully loaded. */

See Also

gx_binres_language_table_read
gx_brush_default

Set the default brush

Prototype

UINT gx_brush_default(GX_BRUSH *brush);

Description

This service sets the brush for the current context to the system default value.

Parameters

brush Pointer to brush control block.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful brush definition</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid brush pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/*Reset the brush to its default value. */
status = gx_brush_default(&my_brush);

/* If status is GX_SUCCESS the brush was successfully reset to its default value. */

See Also

gx_brush_define
**gx_brush_define**

Define brush

Prototype

```c
UINT gx_brush_define(GX_BRUSH *brush, GX_COLOR line_color,
                      GX_COLOR fill_color, UINT style);
```

Description

This service defines a brush with the specified line color, fill color and style.

Parameters

- **brush**: Pointer to brush control block
- **line_color**: Color of brush line. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.
- **fill_color**: Color of brush fill. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.
- **style**: Brush style. Appendix D describes the supported brush styles. Brush styles can be combined into one variable using bitwise OR operation.

Return Values

- **GX_SUCCESS**: (0x00) Successful brush definition
- **GX_PTR_ERROR**: (0x07) Invalid brush pointer

Allowed From

Initialization and threads
Example

/* Define a brush. */
status = gx_brush_define(&my_brush, GX_COLOR_BLACK, GX_COLOR_BLACK,
                        GX_STYLE_BORDER_NONE);

/* If status is GX_SUCCESS the brush was successfully created. */

See Also

gx_brush_default
gx_button_background_draw

Draw button background

Prototype

VOID gx_button_background_draw(GX_BUTTON *button);

Description

This service draws the button background. This function is normally called internally by the gx_button_draw function, but is exposed to the application to assist in writing custom drawing functions.

Parameters

button Pointer to button control block

Return Values

None

Allowed From

Threads

Example

VOID custom_button_draw(GX_BUTTON *button)
{
    /* Draw button background. */
    gx_button_background_draw(button);
    /* Add custom drawing here. */
    /* Draw child widgets. */
    gx_widget_children_draw((GX_WIDGET *)button);
}

See Also

gx_button_create, gx_button_deselect, gx_button_draw,
gx_button_event_process, gx_button_select, gx_icon_button_create,
gx_pixelmap_button_create, gx_pixelmap_button_draw, gx_radio_button_create,
gx_radio_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
gx_button_create

Create button

Prototype

UINT gx_button_create(GX_BUTTON *button, GX_CONST GX_CHAR *name, GX_WIDGET *parent, ULONG style, USHORT button_id, GX_CONST GX_RECTANGLE *size);

Description

This service creates a button as specified and associates the button with the supplied parent widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>Pointer to button control block</td>
</tr>
<tr>
<td>name</td>
<td>Logical name of button</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget of the button</td>
</tr>
<tr>
<td>style</td>
<td>Button style. Appendix D contains pre-defined general styles for all widgets as well as widget specific styles.</td>
</tr>
<tr>
<td>button_id</td>
<td>Application-defined ID of the button</td>
</tr>
<tr>
<td>size</td>
<td>Size of the button</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful button creation</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid widget control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_BUTTON my_top_button;

/* Create a stop button. */
status = gx_button_create(&my_stop_button, "my stop button",
            &my_parent_window,
            GX_STYLE_BUTTON_TOGGLE,
            MY_STOP_BUTTON_ID, &size);

/* If status is GX_SUCCESS the stop button was successfully
created. */

See Also

gx_button_background_draw, gx_button_deselect, gx_button_draw,
gx_button_event_process, gx_button_select, gx_radio_button_create,
gx_radio_button_draw, gx_icon_button_create, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
gx_button_deselect

Deselect button

Prototype

UINT gx_button_deselect(GX_BUTTON *button, GX_BOOL gen_event);

Description

This service deselects the specified button and generate a signal event depending on button styles.

<table>
<thead>
<tr>
<th>Button Style</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>GX_EVENT_CLICKED</td>
</tr>
<tr>
<td>GX_STYLE_BUTTON_RADIO</td>
<td>GX_EVENT_RADIO_DESELECT</td>
</tr>
<tr>
<td>GX_STYLE_BUTTON_TOGGLE</td>
<td>GX_EVENT_TOGGLE_OFFSET</td>
</tr>
</tbody>
</table>

Parameters

- **button**: Pointer to button control block
- **gen_event**: If GX_TRUE, the button will generate a GX_EVENT_CLICKED, GX_EVENT_DESELECT, or GX_EVENT_TOGGLE_OFFSET event depending on the button style. If GX_FALSE, the button will not generate any higher level event even if it would normally do so.

Return Values

- **GX_SUCCESS** (0x00): Successful button deselect
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

- **Threads**

Example

/* Deselect button. */
status = gx_button_deselect(&my_stop_button, GX_TRUE);
/* If status is GX_SUCCESS the stop button was successfully deselected. */

See Also

gx_button_background_draw, gx_button_create, gx_button_draw,
gx_button_event_process, gx_button_select, gx_radio_button_create,
gx_radio_button_draw, gx_icon_button_create, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
gx_button_draw

Prototype

VOID gx_button_draw(GX_BUTTON *button);

Description

This service draws the specified button. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom button widgets.

Parameters

button Pointer to button control block

Return Values

None

Allowed From

Threads

Example

/* Write a custom button draw function. */
VOID custom_button_draw(GX_BUTTON *button)
{
    /* Call default button draw. */
    gx_button_draw(button);
    /* Add custom drawing here. */
}

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_event_process, gx_button_select, gx_radio_button_create,
gx_radio_button_draw, gx_icon_button_create, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
**gx_button_event_process**

Process button event

**Prototype**

```c
UINT gx_button_event_process(GX_BUTTON *button, GX_EVENT *event);
```

**Description**

This service processes an event for the specified button.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>Pointer to button control block</td>
</tr>
<tr>
<td>event_ptr</td>
<td>Pointer to event to process</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful button event process</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

**Allowed From**

Threads

**Example**

```c
/* Call generic button event processing as part of custom event processing function. */

UINT custom_button_event_process(GX_BUTTON *button, GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
        case xyz:
            /* Insert custom event handling here */
            break;
        default:
            /* Pass all other events to the default button event processing */
            status = gx_button_event_process(button, event);
            break;
    }
    return status;
}
```
See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_select, gx_radio_button_create,
gx_radio_button_draw, gx_icon_button_create, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
gx_button_select

Select button

Prototype

(UINT) gx_button_select(GX_BUTTON *button);

Description

This service selects the specified button and generate a signal event depending on button styles.

Deselects the siblings for a radio button group.

<table>
<thead>
<tr>
<th>Button Style</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_STYLE_BUTTON_RADIO</td>
<td>GX_EVENT_RADIO_SELECT</td>
</tr>
<tr>
<td>GX_STYLE_BUTTON_EVENT_ON_PUSH</td>
<td>GX_EVENT.CLICKED</td>
</tr>
<tr>
<td>GX_STYLE_BUTTON_TOGGLE</td>
<td>GX_EVENT_TOGGLE_ON</td>
</tr>
</tbody>
</table>

Parameters

button Pointer to button control block

Return Values

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful button select</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Threads

Example

/* Select button. */
status = gx_button_select(&my_stop_button);

/* If status is GX_SUCCESS the stop button was successfully selected. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_radio_button_create,
gx_radio_button_draw, gx_icon_button_create, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_text_button_create, gx_text_button_color_set,
gx_text_button_draw
**gx_canvas_alpha_set**

Set alpha-blend value for canvas

**Prototype**

```c
UINT  gx_canvas_alpha_set(GX_CANVAS *canvas, GX_UBYTE alpha);
```

**Description**

This service sets the alpha-blend value for the specified canvas. Canvas alpha values can range from 0 (transparent) to 255 (fully opaque).

Blending overlay canvases requires either hardware graphics layer support, or software support via the creation of a composite canvas.

Hardware support for canvas blending is enabled by invoking the `gx_canvas_hardware_layer_bind()` API prior to setting the canvas alpha value. When a canvas is bound to a hardware graphics layer, calling the `gx_canvas_alpha_set()` API directly invokes the hardware graphics layer blending services.

To utilize software support for canvas blending, the application must create a canvas with `GX_CANVAS_COMPOSITE` style, into which all other managed canvases are composited prior to final display. Software support for canvas blending is only provided when running with a display driver of 16-bpp or higher color depth.

**Parameters**

- **canvas**  Pointer to canvas control block
- **alpha**  Alpha-blend value, range from 0 (transparent) to 255 (opaque).

**Return Values**

- **GX_SUCCESS**  (0x00)  Successful alpha-blend value set
- **GX_CALLER_ERROR**  (0x11)  Invalid caller of this function
- **GX_PTR_ERROR**  (0x07)  Invalid pointer
- **GX_INVALID_ERROR**  (0x20)  Invalid canvas

**Allowed From**

Initialization and threads
Example

/* Set the alpha-blend value of "my_canvas". */
status = gx_canvas_alpha_set(&my_canvas, GX_ALPHA_VALUE_OPAQUE);

/* If status is GX_SUCCESS the alpha-blend value was successfully set. */

See Also

gx_canvas_create, gx_canvas_drawing_complete, gx_canvas_drawing_initiate,
gx_canvas_offset_set, gx_canvas_shift, gx_canvas_hardware_layer_bind,
gx_canvas_show, gx_canvas_hide
**gx_canvas_arc_draw**

**Prototype**

```c
UINT gx_canvas_arc_draw(INT xcenter, INT ycenter, UINT r,
                         INT start_angle, INT end_angle);
```

**Description**

This service draws a circle arc on the canvas using the current brush. The circle arc is clipped to the canvas invalid region. This service requires GX_ARC_DRAWING_SUPPORT to be defined.

**Parameters**

- **xcenter**: x-position of center of the circle arc
- **ycenter**: y-position of center of the circle arc
- **r**: Radius of the circle arc
- **start_angle**: Starting angle of the circle arc
- **end_angle**: Ending angle of the circle arc

**Return Values**

- **GX_SUCCESS**: (0x00) Successful arc draw
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_INVALID_VALUE**: (0x22) Invalid value
- **GX_INVALID_CONTEXT**: (0x06) No open drawing context

**Allowed From**

Threads

**Example**

```c
/* Draw a circle arc from 0 degree to 90 degree in clockwise. */
status = gx_canvas_arc_draw(100, 100, 50, 0, 90);

/* If status is GX_SUCCESS the arc has been drawn on “my_canvas”. */
```

**See Also**

- gx_canvas_block_move, gx_canvas_circle_draw, gx_display_create,
- gx_canvas_ellipse_draw, gx_canvas_line_draw, gx_canvas_pie_draw,
- gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_polygon_draw,
- gx_canvas_rectangle_draw, gx_canvas_text_draw
**gx_canvas_block_move**

Move block of canvas pixels

Prototype

```c
UINT gx_canvas_block_move(GX_RECTANGLE *block,
                           GX_VALUE x_shift, GX_VALUE y_shift,
                           GX_RECTANGLE *dirty);
```

Description

This service moves a block of canvas pixel data in the direction specified. This service is used internally by GUIX to accomplish fast scrolling, but may also be used by the application software.

Parameters

- **block**: Coordinates of area to move
- **x_shift**: Number of pixels to shift on the x-axis
- **y_shift**: Number of pixels to shift on the y-axis
- **dirty**: If the block move is successful, this function returns the portion of the source rectangle that is still dirty to the caller in this parameter.

Return Values

- **GX_SUCCESS** (0x00): Successful block move
- **GX_FAILURE** (0x10): Failed block move
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

Allowed From

Threads
Example

GX_RECTANGLE invalid;
GX_RECTANGLE move;

/* define 100 x 100 pixel rectangle */
gx_utility_rectangle_define(&move, 0, 0, 99, 99);

/* Move this rectangle 10 pixels to the right. */
status = gx_canvas_block_move(&move, 10, 0, &invalid);

/* If status is GX_SUCCESS, then ‘invalid’ marks the area that needs to be redrawn after the block move. */

See Also

gx_canvas_arc_draw, gx_canvas_circle_draw, gx_display_create,
gx_canvas_ellipse_draw, gx_canvas_line_draw, gx_canvas_pie_draw,
gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_polygon_draw,
gx_canvas_rectangle_draw, gx_canvas_text_draw
**gx_canvas_circle_draw**

**Draw circle**

### Prototype

```c
UINT gx_canvas_circle_draw(INT xcenter, INT ycenter, UINT r)
```

### Description

This service draws a circle on the canvas using the current brush. The circle is clipped to the canvas invalid region. This service requires GX_ARC_DRAWING_SUPPORT to be defined.

### Parameters

- **xcenter**: x-coord of center of the circle
- **ycenter**: y-coord of center of the circle
- **r**: Radius of the circle

### Return Values

- **GX_SUCCESS**: 0x00 (Successful circle draw)
- **GX_CALLER_ERROR**: 0x11 (Invalid caller of this function)
- **GX_INVALID_VALUE**: 0x22 (Invalid circle radius)
- **GX INVALID_CONTEXT**: 0x06 (No open drawing context)

### Allowed From

**Threads**

### Example

```c
/* Draw a circle of radius 10 centered at (100, 100). */
status = gx_canvas_circle_draw(100, 100, 50);

/* If status is GX_SUCCESS the circle has been drawn on "my_canvas". */
```

### See Also

- `gx_canvas_arc_draw`, `gx_canvas_block_move`, `gx_display_create`
- `gx_canvas_ellipse_draw`, `gx_canvas_line_darw`, `gx_canvas_pie_draw`
- `gx_canvas_pixelmap_draw`, `gx_canvas_pixelmap_tile`, `gx_canvas_polygon_draw`
- `gx_canvas_rectangle_draw`, `gx_canvas_text_draw`
gx_canvas_create

Create canvas

Prototype

UINT gx_canvas_create(GX_CANVAS *canvas, GX_CONST GX_CHAR *name, GX_DISPLAY *display, UINT type, UINT width, UINT height, GX_COLOR *memory_area, ULONG memory_size);

Description

This service creates the canvas with the specified properties and associated memory.

Parameters

canvas Pointer to canvas control block
name Logical name for the canvas
display Pointer to previously created display
type Type of canvas

The canvas types include:

GX_CANVAS_SIMPLE: A memory canvas which is used to off-screen drawing.

GX_CANVAS_MANAGED: A canvas which automatically flushed to the active display, either as part of the composite building process or as part of the buffer toggle operation for single-canvas architectures.

GX_CANVAS_VISIBLE: This flag can be used to turn on and off a canvas, without losing the canvas drawing contents.

GX_CANVAS_MODIFIED: Reserved for future use.

GX_CANVAS_COMPOSITE: This flag is used by the application when configuring a multiple-canvas system which will composite multiple managed canvases into the composite canvas, and the composite is the driven to the hardware frame buffer.
width
height
memory_area
memory_size

Width in pixels
Height in pixels
Memory area for canvas. This value can
GX_NULL at the time of canvas creation
and later initialized using
gx_canvas_memory_define
Size of memory area in bytes, or 0 if the
canvas memory will be defined after the
canvas is created.

Return Values

GX_SUCCESS (0x00) Successful canvas create
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_ALREADY_CREATED (0x13) Widget already created
GX_INVALID_CANVAS_SIZE (0x1C) Invalid canvas control block
size
GX_INVALID_TYPE (0x1B) Invalid canvas type

Allowed From

Initialization and threads

Example

/* Define global canvas memory area. */
GX_COLOR my_canvas_memory[272*480];

... /* Create "my_canvas". */
status = gx_canvas_create(&my_canvas, "my canvas", &my_display,
(GX_CANVAS_MANAGED | GX_CANVAS_VISIBLE),
272, 480,
my_canvas_memory,
sizeof(default_canvas_memory));

/* If status is GX_SUCCESS the 272 x 480 canvas was successfully
created. */

See Also

gx_canvas_delete, gx_canvas_hardware_layer_bind, gx_canvas_memory_define
**gx_canvas_delete**

Delete canvas

**Prototype**

```c
UINT gx_canvas_delete(GX_CANVAS *canvas);
```

**Description**

This service deletes the canvas. The canvas is removed from the internal linked list of canvas maintained by GUIX.

**Parameters**

- `canvas` Pointer to canvas control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful canvas create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_CANVAS** (0x20) Invalid canvas

**Allowed From**

Initialization and threads

**Example**

```c
/* Delete "my_canvas". */
status = gx_canvas_delete(&my_canvas);

/* If status is GX_SUCCESS my_canvas was deleted. */
```

**See Also**

- `gx_canvas_alpha_set`, `gx_canvas_drawing_complete`, `gx_canvas_create`, `gx_canvas_drawing_initiate`, `gx_canvas_offset_set`, `gx_canvas_shift`, etc.
gx_canvas_drawing_complete

Complete canvas drawing

Prototype

UINT gx_canvas_drawing_complete(GX_CANVAS *canvas,
                               GX_BOOL flush);

Description

This service lets GUIX know the application’s drawing on the specified canvas is complete.

The application can use this service to force immediate drawing to a canvas. This flushes the canvas to the visible frame buffer and/or triggers a bugger toggle operation, depending on the system memory architecture.

This service should only be called by the application to close a drawing sequence begun with gx_canvas_drawing_initiate().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>Pointer to canvas control block</td>
</tr>
<tr>
<td>flush</td>
<td>If GX_TRUE, canvas changes are flushed to the display</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successful drawing completion</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Complete drawing on "my_canvas" and flush to display. */
status = gx_canvas_drawing_complete(&my_canvas, GX_TRUE);

/* If status is GX_SUCCESS the canvas drawing was successfully completed. */

See Also

gx_canvas_alpha_set, gx_canvas_create, gx_canvas_drawing_initiate,
gx_canvas_offset_set, gx_canvas_shift
gx_canvas_drawing_initiate

Initiate canvas drawing

Prototype

UINT gx_canvas_drawing_initiate(GX_CANVAS *canvas,
GX_WIDGET *who,
GX_RECTANGLE *dirty_area);

Description

This service initiates drawing on the specified canvas. This service is called internally as part of the deferred drawing operation performed automatically by GUIX when a canvas needs to be update. However, the application is allowed bypass the GUIX deferred drawing algorithm and perform immediate and direct drawing on a canvas by first calling gx_canvas_drawing_initiate, then calling the desired drawing functions, then calling gx_canvas_drawing_complete().

Parameters

canvas Pointer to canvas control block
who Pointer to widget control block of the caller. This parameter is used to initialize the drawing clipping and view parameters for subsequent drawing operations.
dirty_area Area to draw within. This parameter is passed by the caller to indicate the area to which the caller wants all drawing operations clipped. This is usually the area previously marked as dirty, but the caller is free to expand or contract the clipping area.
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful drawing initiation</td>
</tr>
<tr>
<td>GX_DRAW_NESTING_EXCEEDED (0x05)</td>
<td>Exceed maximum nesting count</td>
</tr>
<tr>
<td>GX_NO_VIEW (0x03)</td>
<td>No viewports for the caller</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_CANVAS (0x20)</td>
<td>Invalid canvas</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
/* Initiate drawing on “my_canvas”, my_widget.gx_widget_size specify the area the application wants GUIX to redraw. */
status = gx_canvas_drawing_initiate(&my_canvas, &my_widget, &my_widget.gx_widget_size);

/* If status is GX_SUCCESS the canvas drawing was successfully initiated. */
```

See Also

gx_canvas_alpha_set, gx_canvas_create, gx_canvas_drawing_complete,
gx_canvas_offset_set, gx_canvas_shift
gx_canvas_ellipse_draw

Draw ellipse

Prototype

UINT gx_canvas_ellipse_draw(INT xcenter, INT ycenter, INT a,
INT b)

Description

This service draws an ellipse on the canvas using the current brush. The ellipse is clipped to the canvas invalid region. This service requires GX_ARC_DRAWING_SUPPORT to be defined.

Parameters

xcenter x-coord of center of the ellipse
ycenter y-coord of center of the ellipse
a Length of the semi-major axis
b Length of the semi-minor axis

Return Values

GX_SUCCESS (0x00) Successful circle draw
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_INVALID_VALUE (0x22) Invalid value
GX_INVALID_CONTEXT (0x06) No open drawing context

Allowed From

Threads

Example

/* Draw an ellipse of semi-major radius 100, semi-minor radius 50 and centered at (200, 200). */
status = gx_canvas_ellipse_draw(200, 200, 100, 50);

/* If status is GX_SUCCESS the ellipse has been drawn on "my_canvas". */

See Also

gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
gx_display_create, gx_canvas_line_draw, gx_canvas_pie_draw,
gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_polygon_draw,
gx_canvas_rectangle_draw, gx_canvas_text_draw
**gx_canvas.hardware_layer_bind**

**Bind canvas to hardware graphics layer**

**Prototype**

```c
UINT gx_canvas.hardware_layer_bind(GX_CANVAS *canvas, INT layer)
```

**Description**

This service binds a GUIX drawing canvas to a hardware graphics layer. This service is only required for hardware devices supporting multiple hardware graphics layers.

Binding a canvas to a hardware graphics layer results in the `gx_canvas_show()`, `gx_canvas_hide()`, `gx_canvas_alpha_set()`, and `gx_canvas_offset_set()` APIs being implemented directly by hardware display driver services.

If the hardware display driver does not support multiple graphics layers, this service will fail returning `GX_INVALID_DISPLAY`.

**Parameters**

- **canvas**
  - canvas to be implement in hardware layer

- **layer**
  - hardware graphics layer

**Return Values**

- **GX_SUCCESS** (0x00) Successful binding
- **GX_INVALID_DISPLAY** (0x1D) Display layer service is not defined
- **GX_PTR_ERROR** (0x17) Invalid pointers
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_CANVAS** (0x20) Invalid canvas
- **GX_NOT_SUPPORTED** (0x28) Not supported

**Allowed From**

- Initialization and threads
Example

/* Binds the canvas to the hardware graphics layer 1. */
status = gx_canvas_hardware_layer_bind(&my_canvas, 1);

/* If status is GX_SUCCESS, the drawing canvas is bound to the
hardware graphics. */

See Also

gx_canvas_create, gx_canvas_memory_define
**gx_canvas_hide**

Hide a canvas, making it invisible

### Prototype

```c
UINT  gx_canvas_hide(GX_CANVAS *canvas)
```

### Description

This service hides a GUIX canvas. If the canvas has been bound to a hardware graphics layer using `gx_canvas_hardware_layer_bind()`, this service is implemented using hardware support.

### Parameters

- **canvas**: canvas to be hidden

### Return Values

- **GX_SUCCESS** (0x00): Successful hide
- **GX_INVALID_CANVAS** (0x20): Invalid canvas
- **GX_PTR_ERROR** (0x17): Invalid pointers
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function

### Allowed From

Initialization and threads

### Example

```c
/* Make my_canvas invisible. */
status = gx_canvas_hide(&my_canvas);

/* If status is GX_SUCCESS, the canvas has been hidden. */
```

### See Also

- `gx_canvas_create`, `gx_canvas_drawing_complete`, `gx_canvas_drawing_initiate`, `gx_canvas_offset_set`, `gx_canvas_shift`, `gx_canvas_hardware_layer_bind`, `gx_canvas_show`, `gx_canvas_hide`
gx_canvas_line_draw

Draw line

Prototype

UINT gx_canvas_line_draw(GX_VALUE x_start, GX_VALUE y_start,
                        GX_VALUE x_end, GX_VALUE y_end);

Description

This service draws a line on the canvas using the current brush. The line is clipped to the canvas invalid region.

Parameters

- x_start: Starting x-position of the line
- y_end: Starting y-position of the line
- x_start: Ending x-position of the line
- y_end: Ending y-position of the line

Return Values

- GX_SUCCESS (0x00): Successful line draw
- GX_CALLER_ERROR (0x11): Invalid caller of this function
- GX_PTR_ERROR (0x07): Invalid pointer
- GX_INVALID_CONTEXT (0x06): No open drawing context
- GX_INVALID_WIDTH (0x1E): Invalid brush width

Allowed From

Threads

Example

/* Draw line on canvas. */
status = gx_canvas_line_draw(0, 1, 320, 480);

/* If status is GX_SUCCESS, the line has been drawn to canvas. */

See Also

gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
gx_display_create, gx_canvas_ellipse_draw, gx_canvas_pie_draw,
gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_polygon_draw,
gx_canvas_rectangle_draw, gx_canvas_text_draw
gx_canvas_memory_define

Define canvas memory

Prototype

UINT gx_canvas_memory_define(GX_CANVAS *canvas, GX_COLOR *memory, ULONG memsize);

Description

This service can be used to assign the canvas memory address after the canvas has been created.

Parameters

canvas Pointer to previously created canvas
memory Canvas memory address
memsize Size of the canvas memory block in bytes

Return Values

GX_SUCCESS (0x00) Successful assignment
GX_INVALID_CANVAS (0x20) Invalid control block
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Threads

Example

/* Assign canvas memory block. */
status = gx_canvas_memory_define(canvas, (GX_COLOR *) DRAM_MEMORY, (640 * 480 * 2));

/* If status is GX_SUCCESS, the canvas memory pointer has be reassigned. */

See Also

gx_canvas_create, gx_canvas_hardware_layer_bind
gx_canvas_mouse_define

Define the mouse cursor image

Prototype

UINT gx_canvas_mouse_define(GX_CANVAS *canvas,
                            GX_MOUSE_CURSOR_INFO *info);

Description

This service defines mouse information for the specified canvas. This service requires GX_MOUSE_SUPPORT to be defined.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>Pointer to canvas control block</td>
</tr>
<tr>
<td>info</td>
<td>Pointer to mouse cursor information.</td>
</tr>
</tbody>
</table>

Appendix I contains definition to GX_MOUSE_CURSOR_INFO structure.

Return Values

- **GX_SUCCESS** (0x00) Successful mouse info set
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

Allowed From

Initialization and threads

Example

```c
/* Set mouse cursor info. */
GX_MOUSE_CURSOR_INFO mouse_cursor;
mouse_cursor.gx_mouse_cursor_image_id = GX_PIXELMAP_ID_MOUSE;
mouse_cursor.gx_mouse_cursor_hotspot_x = 0;
mouse_cursor.gx_mouse_cursor_hotspot_y = 0;

status = gx_canvas_mouse_define(&my_canvas, &mouse_cursor);

/* If status is GX_SUCCESS the mouse info of “my_canvas” has been set successfully. */
```

See Also

- gx_canvas_mouse_show
- gx_canvas_mouse_hide
**gx_canvas_mouse_hide**

**Turn off the mouse cursor**

**Prototype**

```c
UINT gx_canvas_mouse_hide(GX_CANVAS *canvas);
```

**Description**

This service makes the mouse cursor hidden from the specified canvas. This service requires GX_MOUSE_SUPPORT to be defined.

**Parameters**

- **canvas**
  Pointer to canvas control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful mouse cursor hide
- **GX_FAILURE** (0x10) Failed mouse cursor hide
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

**Allowed From**

Initialization and threads

**Example**

```c
/* Hide the mouse cursor. */
status = gx_canvas_mouse_hide(&my_canvas);

/* If status is GX_SUCCESS the mouse cursor of “my_canvas” has been hidden successfully. */
```

**See Also**

- `gx_canvas_mouse_show`, `gx_canvas_mouse_define`
**gx_canvas_mouse_show**

Turn on the mouse cursor

**Prototype**

```c
UINT  gx_canvas_mouse_show(GX_CANVAS *canvas);
```

**Description**

This service makes the mouse cursor visible for the specified canvas. This service requires GX_MOUSE_SUPPORT to be defined. The `gx_canvas_mouse_define` API should be invoked to define the mouse cursor image before this service is requested.

**Parameters**

- **canvas**
  Pointer to canvas control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful mouse info set
- **GX_FAILURE** (0x10) Failed mouse cursor show
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

**Allowed From**

Initialization ansd threads

**Example**

```c
/* Make mouse cursor hidden */
status = gx_canvas_mouse_show(&my_canvas);

/* If status is GX_SUCCESS the mouse of “my_canvas” has been hidden successfully. */
```

**See Also**

`gx_canvas_mouse_show`, `gx_canvas_mouse_define`


gx_canvas_offset_set

Assign canvas x,y display offset

Prototype

UINT gx_canvas_offset_set(GX_CANVAS *canvas, GX_VALUE x, GX_VALUE y);

Description

This service assigns an x,y display offset for the specified canvas. This controls the position at which the canvas is composited into the visible frame buffer, and is often used when the canvas is smaller than the physical display.

If the canvas has been bound to a hardware graphics layer using the gx_canvas_hardware_layer_bind() API, the gx_canvas_offset_set service is implemented directly using hardware support.

Parameters

- **canvas**: Pointer to canvas control block
- **x**: X coordinate of offset
- **y**: Y coordinate of offset

Return Values

- **GX_SUCCESS**: (0x00) Successful assignment of offset
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_CANVAS**: (0x20) Invalid canvas

Allowed From

- Initialization and threads

GUIX User Guide 135
Example

/* Set display offset for "my_canvas". */
status = gx_canvas_offset_set(&my_canvas, 20, 30);

/* If status is GX_SUCCESS the canvas drawing is now offset from position 20,30. */

See Also

gx_canvas_alpha_set, gx_canvas_create, gx_canvas_drawing_complete,
gx_canvas_initiate, gx_canvas_shift, gx_canvas_show, gx_canvas_hide,
gx_canvas_hardware_layer_bind
**gx_canvas_pie_draw**

**Draw pie**

**Prototype**

```c
UINT gx_canvas_pie_draw(INT xcenter, INT ycenter, UINT r,
                        INT start_angle, INT end_angle);
```

**Description**

This service draws a pie into the canvas using the current drawing context brush. The pie is clipped to the canvas invalid region. This service requires the configuration option GX_ARC_DRAWING_SUPPORT to be defined.

**Parameters**

- **xcenter**: x-position of center of the pie
- **ycenter**: y-position of center of the pie
- **r**: Radius of the pie
- **start_angle**: Starting angle of the pie
- **end_angle**: Ending angle of the pie

**Return Values**

- **GX_SUCCESS**: (0x00) Successful arc draw
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_INVALID_VALUE**: (0x22) Invalid value
- **GX_INVALID_CONTEXT**: (0x06) No open drawing context

**Allowed From**

Initialization and threads

**Example**

```c
/* Draw a pie from 0 degree to 90 degree in clockwise. */
status = gx_canvas_pie_draw(100, 100, 50, 0, 90);

/* If status is GX_SUCCESS the pie has been drawn to canvas. */
```

**See Also**

- gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
- gx_display_create, gx_canvas_ellipse_draw, gx_canvas_line_draw,
- gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_polygon_draw,
- gx_canvas_rectangle_draw, gx_canvas_text_draw
**gx_canvas_pixel_draw**

**Prototype**

```c
UINT  gx_canvas_pixel_draw(GX_POINT position);
```

**Description**

This service draws a pixel on the canvas using the line color of the current drawing context brush. If configuration option GX_BRUSH_ALPHA_SUPPORT is defined, blend the pixel with the background color using the current drawing context brush alpha, otherwise, draw the pixel as fully opaque.

**Parameters**

- **point**
  
  x,y position of pixel to draw

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0x00</strong></td>
<td>GX_SUCCESS Successful pixelmap draw</td>
</tr>
<tr>
<td><strong>0x11</strong></td>
<td>GX_CALLER_ERROR Invalid caller of this function</td>
</tr>
<tr>
<td><strong>0x06</strong></td>
<td>GX_INVALID_CONTEXT No open drawing context</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads
Example

GX_POINT point;    /* the x,y position you want to draw to */
GX_RECTANGLE drawto; /* the rectangle bounding your drawing */
GX_CANVAS *mycanvas; /* the canvas you want to draw to */

/* calculate 1x1 pixel drawing area: */
gx_utility_rectangle_define(&drawto,
    point.gx_point_x, point.gx_point_y,
    point.gx_point_x, point.gx_point_y);

/* get my canvas: */
gx_widget_canvas_get(win, &mycanvas);

/* open my canvas for drawing: */
gx_canvas_drawing_initiate(mycanvas, win, &drawto);

/* setup my brush colors. Use any color ID in your resources: */
gx_context_line_color_set(GX_COLOR_ID_WINDOW_BORDER);

/* draw a pixel: */
status = gx_canvas_pixel_draw(point);

/* close the canvas: */
gx_canvas_drawing_complete(mycanvas, GX_TRUE);

/* If status is GX_SUCCESS, the pixel was successfully drawn to mycanvas. */

See Also

gx_canvas_block_move, gx_canvas_pixelmap_tile, gx_canvas_pixelmap_blend,
**gx_canvas_pixelmap_blend**

**Blend pixelmap**

**Prototype**

```c
UINT gx_canvas_pixelmap_blend(GX_VALUE x_position,
                               GX_VALUE y_position,
                               GX_PIXELMAP *pixelmap,
                               GX_UBYTE alpha);
```

**Description**

This service blends a pixelmap with the canvas background. The blending ratio is specified by the caller. The alpha value can range from 0 (fully transparent) to 255 (fully opaque). The pixelmap may also include an internal alpha channel which is combined with the incoming blending value. This service is only supported by display drivers running at 16-bpp color depth and higher.

**Parameters**

- **x_start**
  - Starting x-position of the pixelmap
- **y_end**
  - Starting y-position of the pixelmap
- **pixelmap**
  - Pointer to pixelmap

**Return Values**

- **GX_SUCCESS** (0x00) Successful pixelmap draw
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_NOT_SUPPORTED** (0x28) Not supported
- **GX_INVALID_CONTEXT** (0x06) No open drawing context

**Allowed From**

- Initialization and threads
Example

/* Draw pixmap on active canvas */

GX_PIXEMAP *map;
gx_system_pixmap_get(ID_MY_PIXEMAP, &map);

status = gx_canvas_pixmap_blend(10, 20, map, 128);

/* If status is GX_SUCCESS the pixmap has been blended onto the current canvas. */

See Also

gx_canvas_block_move, gx_canvas_pixmap_get, gx_canvas_pixmap_tile,
gx_canvas_pixmap_draw
**gx_canvas_pixelmap_draw**

**Draw pixelmap**

**Prototype**

```c
UINT gx_canvas_pixelmap_draw(GX_VALUE x_position,
                              GX_VALUE y_position,
                              GX_PIXELMAP *pixelmap);
```

**Description**

This service draws a pixelmap on the canvas.

**Parameters**

- **x_start**: Starting x-position of the pixelmap
- **y_end**: Starting y-position of the pixelmap
- **pixelmap**: Pointer to pixelmap

**Return Values**

- **GX_SUCCESS**: (0x00) Successful pixelmap draw
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_CONTEXT**: (0x06) No open drawing context

**Allowed From**

Initialization and threads

**Example**

```c
/* Draw pixelmap on canvas. */
status = gx_canvas_pixelmap_draw(10, 20, &my_pixelmap);
/* If status is GX_SUCCESS the pixelmap "my_pixelmap" has been
drawn. */
```

**See Also**

- `gx_canvas_block_move`
- `gx_canvas_pixelmap_get`
- `gx_canvas_pixelmap_tile`
- `gx_canvas_pixelmap_blend`
gx_canvas_pixelmap_get

Get canvas pixelmap

Prototype

UINT  gx_canvas_pixelmap_get(GX_PIXELMAP *pixelmap);

Description

This service returns a GX_PIXELMAP structure pointing to the canvas data. The pixelmap format is set to the current display color format.

Parameters

pixelmap  Returned pixelmap

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful pixelmap get</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Draw pixelmap on active canvas */

GX_PIXELMAP *map;

status = gx_canvas_pixelmap_get(map);

/* If status is GX_SUCCESS the pixelmap has been retrieved. */

See Also

gx_canvas_pixelmap_blend, gx_canvas_pixelmap_tile,
gx_canvas_pixelmap_draw
**gx_canvas_pixelmap_rotate**  
Draw rotated pixelmap

**Prototype**

```c
UINT gx_canvas_pixelmap_rotate(GX_VALUE x_position,
                                GX_VALUE y_position,
                                GX_PIXELMAP *pixelmap,
                                INT angle,
                                INT rot_cx,
                                INT rot_cy);
```

**Description**

This service rotates a pixelmap at the specified angle and renders the pixelmap to the canvas directly as the rotation is performed. This service differs from `gx_utility_pixelmap_rotate` in that the output of the rotation is directly rendered to the canvas memory, and the rotated pixelmap is not returned to the caller.

The advantage of this service over `gx_utility_pixelmap_rotate` is that no additional memory is required to hold the rotated pixelmap. The disadvantage is that the rotation code must be executed each time the pixelmap is drawn.

Clipping and viewport validation are enforced during rendering of the rotated pixelmap.

**Parameters**

- **x_position**  
  Starting x-position of the pixelmap
- **y_position**  
  Starting y-position of the pixelmap
- **pixelmap**  
  Pointer to pixelmap
- **angle**  
  Angle to rotate
- **rot_cx**  
  X-coord of center of rotation. If this value is set to -1, the center of the image is used as the rotation center.
- **rot_cy**  
  Y-coord of center of rotation. If this value is set to -1, the center of the image is used as the center of rotation.

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successful pixelmap draw
- **GX_CALLER_ERROR**  
  (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  
  Invalid pointer
- **GX_INVALID_CONTEXT**  
  (0x06)  
  No open drawing context
Allowed From

Initialization and threads

Example

/* rotate "src_pixelmap" by 30 degree in clockwise direction and draw in on canvas. */
status = gx_canvas_pixelmap_rotate(10, 20, &my_pixelmap, 30, -1, -1);

/* If status is GX_SUCCESS the rotated pixelmap "my_pixelmap" has been drawn. */

See Also

gx_canvas_block_move, gx_canvas_pixelmap_get, gx_canvas_pixelmap_tile,
gx_canvas_pixelmap_blend,
gx_canvas_pixelmap_tile

Prototype

UINT gx_canvas_pixelmap_tile(GX_RECTANGLE *fill,
                           GX_PIXELMAP *pixelmap);

Description

This service fills a rectangle within a canvas with the requested pixelmap.

Parameters

- fill: Area to tile with pixelmap
- pixelmap: Pointer to pixelmap

Return Values

- GX_SUCCESS (0x00): Successful pixelmap tile
- GX_CALLER_ERROR (0x11): Invalid caller of this function
- GX_PTR_ERROR (0x07): Invalid pointer
- GX_INVALID_CONTEXT (0x06): No open drawing context
- GX_INVALID_VALUE (0x22): Invalid fill size

Allowed From

Initialization and threads

Example

/* Tile pixelmap on canvas */
status = gx_canvas_pixelmap_tile(&tile_area, &my_pixelmap);

/* If status is GX_SUCCESS the pixelmap "my_pixelmap" has been tiled on canvas */

See Also

gx_canvas_block_move, gx_canvas_pixelmap_get, gx_canvas_pixelmap_blend,
gx_canvas_pixelmap_draw,
**gx_canvas_polygon_draw**

**Draw polygon**

**Prototype**

```c
UINT gx_canvas_polygon_draw(GX_POINT *point_array,
                             INT number_of_points);
```

**Description**

This service draws a polygon on the canvas using the current drawing context brush.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point_array</td>
<td>Array of points of the polygon</td>
</tr>
<tr>
<td>number_of_points</td>
<td>Number of points of polygon</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful polygon draw</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_CONTEXT</td>
<td>(0x06) No open drawing context</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_POINT my_polygon[4] = { { 208, 63 }, { 274, 63 }, { 274, 163 }, { 208, 163 } };

/* Draw polygon "my_polygon" on canvas. */
status = gx_canvas_polygon_draw(&my_polygon, 4);

/* If status is GX_SUCCESS the polygon "my_polygon" has been drawn. */
```

**See Also**

gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw, gx_display_create, gx_canvas_ellipse_draw, gx_canvas_line_draw, gx_canvas_pie_draw, gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile, gx_canvas_rectangle_draw, gx_canvas_text_draw
gx_canvas_rectangle_draw

Draw rectangle

Prototype

UINT gx_canvas_rectangle_draw(GX_RECTANGLE *rectangle);

Description

This service draws a rectangle on the canvas.

Parameters

rectangle Rectangle to draw

Return Values

GX_SUCCESS (0x00) Successful rectangle draw
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_CONTEXT (0x06) No open drawing context

Allowed From

Initialization and threads

Example

/* Draw rectangle “my_rectangle” on canvas. */
status = gx_canvas_rectangle_draw(&my_rectangle);

/* If status is GX_SUCCESS the rectangle “my_rectangle” has been
drawn. */

See Also

gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
gx_display_create, gx_canvas_ellipse_draw, gx_canvas_line_draw,
gx_canvas_pie_draw, gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile,
gx_canvas_polygon_draw, gx_canvas_text_draw


**gx_canvas_rotated_text_draw**

Draw text rotated about a center point (deprecated)

**Prototype**

```c
UINT gx_canvas_rotated_text_draw(const GX_CHAR *text,
                                 GX_VALUE xCenter, GX_VALUE yCenter, INT angle);
```

**Description**

This API has been deprecated in favor of `gx_canvas_rotated_text_draw_ext()`. While still supported, new applications should not use this API and should instead use `gx_canvas_rotated_text_draw_ext()`.

This service draws text to the canvas. The text is drawn rotated about the requested center point. The current drawing context font and drawing context line color is used to render the text.

This service uses the function `gx_utility_string_to_alphamap` to render the text string to a temporary 8bpp pixelmap containing only alpha value. The service then rotates the alphamap using the function `gx_utility_pixelmap_rotate`. After the final alphamap is rendered to the canvas, this service frees the temporary alphamap and associated memory.

Since a temporary alphamap is required to render rotated text, the application must configure the `gx_system_memory_allocator` by the calling `gx_system_memory_allocator_set()` API before attempting to draw rotated text.

This service should only be used to render rotated text “one time”. If the same text string will be drawn multiple times at different locations or different rotation angles, it is more efficient to use the utility function `gx_utility_string_to_alphamap()` to create the text alphamap once, then use `gx_utility_pixelmap_rotate` multiple times to rotate the resulting alphamap repeatedly.

**Parameters**

- **text**: Text string to be drawn
- **xCenter**: Center position around which text will be rotated.
- **yCenter**: Center position around which text will be rotated.
angle

The desired text rotation angle, in degrees.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful text rendering</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_SYSTEM_MEMORY_ERROR</td>
<td>Insufficient memory available or gx_system_memory_allocator has not been assigned</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
void my_window_draw(GX_WINDOW *window)
{
    GX_VALUE xpos = 100;
    GX_VALUE ypos = 100;
    INT dynamic_count = 1234567;
    GX_CHAR dynamic_text[10];

    /* Call default window draw routine. */
    gx_window_draw(window);

    /* Set font. */
    gx_context_font_set(GX_FONT_ID_SMALL_BOLD);

    /* Convert int value to string. */
    gx_utility_ltoa(dynamic_count, dynamic_text, 20);

    /* Draw rotate text. */
    gx_canvas_rotated_text_draw(dynamic_text, xpos, ypos, 45);
}
```

See Also

gx_canvas_alpha_set, gx_canvas_drawing_complete, gx_canvas_create,
gx_canvas_drawing_initiate, gx_canvas_offset_set, gx_canvas_shift,
**gx_canvas_rotated_text_draw_ext**

Draw text rotated about a center point

**Prototype**

```c
UINT gx_canvas_rotated_text_draw_ext(GX_CONST GX_STRING *text,
                                       GX_VALUE xCenter, GX_VALUE yCenter, INT angle);
```

**Description**

This service draws text to the canvas. The text is drawn rotated about the requested center point. The current drawing context font and drawing context line color is used to render the text.

This service uses the function `gx_utility_string_to_alphamap` to render the text string to a temporary 8bpp pixelmap containing only alpha value. The service then rotates the alphamap using the function `gx_utility_pixelmap_rotate`. After the final alphamap is rendered to the canvas, this service frees the temporary alphamap and associated memory.

Since a temporary alphamap is required to render rotated text, the application must configure the `gx_system_memory_allocator` by the calling `gx_system_memory_allocator_set()` API before attempting to draw rotated text.

This service should only be used to render rotated text “one time”. If the same text string will be drawn multiple times at different locations or different rotation angles, it is more efficient to use the utility function `gx_utility_string_to_alphamap()` to create the text alphamap once, then use `gx_utility_pixelmap_rotate` multiple times to rotate the resulting alphamap repeatedly.

**Parameters**

- **text**
  - Text string to be drawn

- **xCenter**
  - Center position around which text will be rotated.

- **yCenter**
  - Center position around which text will be rotated.

- **angle**
  - The desired text rotation angle, in degrees.
Return Values

**GX_SUCCESS** (0x00) Successful text rendering

**GX_PTR_ERROR** (0x07) Invalid pointer

**GX_CALLER_ERROR** (0x11) Invalid caller of this function

**GX_INVALID_CONTEXT** (0x06) Invalid draw context

**GX_SYSTEM_MEMORY_ERROR** (0x30) Insufficient memory available or gx_system_memory_allocator has not been assigned.

**GX_INVALID_STRING_LENGTH** (0x34) Invalid string length

Allowed From

Initialization and threads

Example

```c
void my_window_draw(GX_WINDOW *window)
{
    GX_VALUE xpos = 100;
    GX_VALUE ypos = 100;
    INT dynamic_count = 1234567;
    GX_CHAR dynamic_text[10];
    GX_STRING string;

    /* Call default window draw routine. */
    gx_window_draw(window);

    /* Set font. */
    gx_context_font_set(GX_FONT_ID_SMALL_BOLD);

    /* Convert int value to string. */
    gx_utility_ltoa(dynamic_count, dynamic_text, 20);

    string.gx_string_ptr = dynamic_text;
    string.gx_string_length = strlen(dynamic_text);

    /* Draw rotate text. */
    gx_canvas_rotated_text_draw_ext(&string, xpos, ypos, 45);
}
```

See Also

gx_canvas_alpha_set, gx_canvas_drawing_complete, gx_canvas_create,
gx_canvas_drawing_initiate, gx_canvas_offset_set, gx_canvas_shift,
gx_canvas_shift

Shift canvas by x,y

Prototype

UINT  gx_canvas_shift(GX_CANVAS *canvas, GX_VALUE x, GX_VALUE y);

Description

This service shifts the specified canvas offset by the specified amount. This affects the position at which the canvas is rendered within the visible frame buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>Pointer to canvas control block</td>
</tr>
<tr>
<td>x</td>
<td>Pixels to shift on the X axis</td>
</tr>
<tr>
<td>y</td>
<td>Pixels to shift on the Y axis</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful canvas shift
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_CANVAS** (0x20) Invalid canvas

Allowed From

- Initialization and threads

Example

```c
/* Shift canvas “my_canvas”. */
status = gx_canvas_shift(&my_canvas, 10, 15);

/* If status is GX_SUCCESS the canvas has been shifted by 10 pixels on the X axis and 15 on the Y axis. */
```

See Also

gx_canvas_alpha_set, gx_canvas_create, gx_canvas_drawing_complete, gx_canvas_initiate, gx_canvas_offset_set
**gx_canvas_show**

Make a canvas visible

### Prototype

```c
UINT gx_canvas_show(GX_CANVAS *canvas);
```

### Description

This service makes a canvas visible. If the canvas has been previously bound to a hardware graphics layer using the `gx_canvas_hardware_layer_bind()` API, the `gx_canvas_show()` service is implemented directly using hardware support.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>Pointer to canvas control block</td>
</tr>
</tbody>
</table>

### Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful assignment of offset</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_CANVAS</td>
<td>(0x20) Invalid canvas</td>
</tr>
</tbody>
</table>

### Allowed From

Initialization and threads

### Example

```c
/* Make this canvas visible. */
status = gx_canvas_show(&my_canvas);
/* If status is GX_SUCCESS the canvas drawing is now visible */
```

### See Also

`gx_canvas_alpha_set, gx_canvas_create, gx_canvas_drawing_complete, gx_canvas_initiate, gx_canvas_shift, gx_canvas_hide, gx_canvas_hardware_layer_bind`
**gx_canvas_text_draw**

Draw text (deprecated)

**Prototype**

```c
UINT gx_canvas_text_draw(GX_VALUE x_start, GX_VALUE y_start,
                        GX_CONST GX_CHAR *string, INT length);
```

**Description**

This service draws text on the canvas. This API, while still supported, is deprecated and new applications should instead use `gx_canvas_text_draw_ext()`.

**Parameters**

- **x_start**: Starting x-coordinate for text
- **y_start**: Starting y-coordinate for text
- **string**: Pointer to string to draw
- **length**: If length >= 0, limits the number of characters drawn to length. If length < 0, the entire string until NULL terminator is drawn.

**Return Values**

- **GX_SUCCESS** (0x00): Successful text draw
- **GX_FAILURE** (0x1E): Failed text draw
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_CONTEXT** (0x06): No open drawing context
- **GX_INVALID_STRING_LENGTH** (0x34): Invalid string length

**Allowed From**

- Initialization and threads

**Example**

```c
/* Draw text "example" on current canvas */
status = gx_canvas_text_draw(10, 20, "example", 7);

/* Draw all of a string of unknown length on the current canvas */
status = gx_canvas_text_draw(10, 40, string_ptr, -1);

/* If status is GX_SUCCESS the text "example" has been drawn. */
```
See Also

gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
gx_display_create, gx_canvas_ellipse_draw, gx_canvas_line_draw,
gx_canvas_pie_draw, gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile,
gx_canvas_polygon_draw, gx_canvas_rectangle_draw
gx_canvas_text_draw_ext

Draw text

Prototype

UINT gx_canvas_text_draw_ext(GX_VALUE x_start, GX_VALUE y_start,
GX_CONST GX_STRING *string);

Description

This service draws text on the canvas.

Parameters

  x_start Starting x-coordinate for text
  y_start Starting y-coordinate for text
  string Pointer to string to draw

Return Values

  GX_SUCCESS   (0x00)   Successful text draw
  GX_FAILURE   (0x1E)   Failed text draw
  GX_CALLER_ERROR (0x11) Invalid caller of this function
  GX_PTR_ERROR  (0x07)   Invalid pointer
  GX_INVALID_CONTEXT (0x06) No open drawing context
  GX_INVALID_STRING_LENGTH   (0x34)   Invalid string length
  GX_INVALID_FONT  (0x16)   Invalid font

Allowed From

  Initialization and threads

Example

GX_STRING string;
string.gx_string_ptr = "example";
string.gx_string_length = 7;

/* Draw text "example" on current canvas". */
status = gx_canvas_text_draw_ext(10, 20, &string);

/* If status is GX_SUCCESS the text "example" has been drawn. */

See Also

  gx_canvas_arc_draw, gx_canvas_block_move, gx_canvas_circle_draw,
  gx_display_create, gx_canvas_ellipse_draw, gx_canvas_line_draw,
gx_canvas_pie_draw, gx_canvas_pixelmap_draw, gx_canvas_pixelmap_tile,
gx_canvas_polygon_draw, gx_canvas_rectangle_draw
gx_checkbox_create

Create checkbox

Prototype

UINT  gx_checkbox_create(GX_CHECKBOX *checkbox,
GX_CONST GX_CHAR *name,
GX_WIDGET *parent,
GX_RESOURCE_ID text_id,
ULONGLONG style, USHORT checkbox_id,
GX_CONST GX_RECTANGLE *size);

Description

This service creates a checkbox widget with the specified properties. GX_CHECKBOX is derived from GX_TEXT_BUTTON, and all gx_text_button services may be used with GX_CHECKBOX widgets.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkbox</td>
<td>Pointer to checkbox control block</td>
</tr>
<tr>
<td>name</td>
<td>Logical name of checkbox widget</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to the parent widget</td>
</tr>
<tr>
<td>text_id</td>
<td>Resource ID of checkbox text</td>
</tr>
<tr>
<td>style</td>
<td>Style of checkbox. Appendix D contains pre-defined general styles for all widgets as well as widget specific styles.</td>
</tr>
<tr>
<td>checkbox_id</td>
<td>Application-defined ID of checkbox</td>
</tr>
<tr>
<td>size</td>
<td>Dimensions of checkbox</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful checkbox create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid size</td>
</tr>
</tbody>
</table>
Allowed From

Initialization and threads

Example

/* Create "my_checkbox". */
status = gx_checkbox_create(&my_checkbox, "my_checkbox",
        &my_parent,
        MY_CHECKBOX_TEXT_RESOURCE_ID, GX_STYLE_BORDER_RAISED,
        MY_CHECKBOX_ID,
        &size);

    /* If status is GX_SUCCESS the checkbox "my_checkbox" has been created. */

See Also

gx_checkbox_draw, gx_checkbox_event_process, gx_checkbox_select
**gx_checkbox_draw**

**Draw checkbox**

**Prototype**

```c
VOID gx_checkbox_draw(GX_CHECKBOX *checkbox);
```

**Description**

This service draws the specified checkbox. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom checkbox widgets.

**Parameters**

- **checkbox** Pointer to checkbox control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom checkbox draw function. */
VOID custom_checkbox_draw(GX_CHECKBOX *checkbox)
{
    /* Call default checkbox draw. */
    gx_checkbox_draw(checkbox);

    /* Add custom drawing here. */
}
```

**See Also**

- `gx_checkbox_create`
- `gx_checkbox_event_process`
- `gx_checkbox_select`
**gx_checkbox_event_process**

Process checkbox event

**Prototype**

```c
UINT gx_checkbox_event_process(GX_CHECKBOX *checkbox,
                                 GX_EVENT *event_ptr);
```

**Description**

This service processes an event for the specified checkbox. This service should be called as the default event handler by any custom checkbox event processing functions.

**Parameters**

- **checkbox**: Pointer to checkbox control block
- **event_ptr**: Pointer to the event to process

**Return Values**

- **GX_SUCCESS** (0x00): Successful checkbox event process
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Threads
Example

/* Call generic checkbox event processing as part of custom event processing function. */

UINT custom_checkbox_event_process(GX_CHECKBOX *checkbox,
                                     GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
        case xyz:
            /* Insert custom event handling here */
            break;
        default:
            /* Pass all other events to the default checkbox event processing */
            status = gx_checkbox_event_process(checkbox, event);
            break;
    }
    return status;
}

See Also

gx_checkbox_create, gx_checkbox_draw, gx_checkbox_select
**gx_checkbox_pixelmap_set**

Set pixelmap for checkbox

### Prototype

```
UINT gx_checkbox_pixelmap_set(GX_CHECKBOX *checkbox,
    GX_RESOURCE_ID unchecked_id,
    GX_RESOURCE_ID checked_id,
    GX_RESOURCE_ID unchecked_disabled_id,
    GX_RESOURCE_ID checked_disabled_id)
```

### Description

This service assigns the pixelmaps to be displayed by the specified checkbox for each checkbox state. The resource IDs can be duplicated.

### Parameters

- **checkbox**: Pointer to checkbox control block
- **unchecked_id**: Pixelmap used for unchecked state
- **checked_id**: Pixelmap used for checked state
- **unchecked_disabled_id**: Pixelmap used for a disabled and unchecked checkbox
- **checked_disabled_id**: Pixelmap used for a disabled and checked checkbox

### Return Values

- **GX_SUCCESS** (0x00): Successful checkbox select
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

### Allowed From

- Initialization and threads
Example

/* Select "my_checkbox". */
status = gx_checkbox_pixelmap_set(&my_checkbox,
    PIXELMAP_UNCHECKED_ID,
    PIXELMAP_CHECKED_ID,
    PIXELMAP_UNCHECKED_DISABLED_ID,
    PIXELMAP_CHECKED_ENABLED_ID);

/* If status is GX_SUCCESS the pixelmaps are assigned to the checkbox "my_checkbox" */

See Also

gx_checkbox_create, gx_checkbox_draw, gx_checkbox_event_process
gx_checkbox_select

Select checkbox

Prototype

UINT gx_checkbox_select(GX_CHECKBOX *checkbox);

Description

This service forces a checkbox to the selected state.

Parameters

checkbox

Pointer to checkbox control block

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful checkbox select</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Select “my_checkbox”. */
status = gx_checkbox_select(&my_checkbox);

/* If status is GX_SUCCESS the checkbox “my_checkbox” has been toggled. */

See Also

gx_checkbox_create, gx_checkbox_draw, gx_checkbox_event_process
## gx_circular_gauge_angle_get

### Get current angle

#### Prototype

```c
UINT gx_circular_gauge_angle_get(GX_CIRCULAR_GAUGE *gauge, INT *angle);
```

#### Description

This service retrieves the current needle angle of circular gauge widget.

#### Parameters

- `gauge`: Pointer to circular gauge control block
- `angle`: Current needle angle to be retrieved

#### Return Values

- `GX_SUCCESS` (0x00): Successful circular gauge angle get
- `GX_PTR_ERROR` (0x07): Invalid pointer

#### Allowed From

Initialization and threads

#### Example

```c
INT current_angle;

/* Retrieve the current needle angle of "my_gauge". */
status = gx_circular_gauge_angle_get(&my_gauge, &current_angle);

/* If status is GX_SUCCESS the current needle angle of "my_gauge" has been retrieved. */
```

#### See Also

- `gx_circular_gauge_angle_set`, `gx_circular_gauge_animation_set`,
- `gx_circular_gauge_background_draw`, `gx_circular_gauge_create`,
- `gx_circular_gauge_draw`, `gx_circular_gauge_event_process`
**gx_circular_gauge_angle_set**

Set target angle

**Prototype**

```c
UINT gx_circular_gauge_angle_set(GX_CIRCULAR_GAUGE *gauge, INT angle);
```

**Description**

This service sets the target angle of a circular gauge widget.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gauge</td>
<td>Pointer to circular gauge control block</td>
</tr>
<tr>
<td>angle</td>
<td>Target needle angle</td>
</tr>
</tbody>
</table>

**Return Values**

- **GX_SUCCESS** (0x00) Successful angle set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Set target angle of "my_gauge" to 180. */
status = gx_circular_gauge_angle_set(&my_gauge, 180);

/* If status is GX_SUCCESS the circular gauge of "my_gauge" has been set. */
```

**See Also**

- `gx_circular_gauge_angle_get`, `gx_circular_gauge_animation_set`
- `gx_circular_gauge_background_draw`, `gx_circular_gauge_create`
- `gx_circular_gauge_draw`, `gx_circular_gauge_event_process`
**gx_circular_gauge_animation_set**

Set animation parameters

**Prototype**

```c
UINT gx_circular_gauge_animation_set(GX_CIRCULAR_GAUGE *gauge,
                                       INT steps, INT delay);
```

**Description**

This service sets animation steps and delay time for a circular gauge widget.

**Parameters**

- **gauge**  
  Pointer to circular gauge control block
- **steps**  
  Total steps for one rotation
- **delay**  
  Delay time for every step

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successful checkbox select
- **GX_CALLER_ERROR**  
  (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  
  Invalid pointer
- **GX_INVALID_VALUE**  
  (0x22)  
  Invalid value

**Allowed From**

Initialization and threads

**Example**

```c
/* Set animation steps and delay time of circular gauge “my_gauge”
to 30 and 1, the needle of “my_gauge” will rotate from current
angle to target angle by 30 steps with 1 tick delay between every
step. */
status = gx_circular_gauge_animation_set(&my_gauge, 30, 1);

/* If status is GX_SUCCESS the steps and delay time of “my_gauge”
has been set. */
```

**See Also**

- `gx_circular_gauge_angle_get`, `gx_circular_gauge_angle_set`
- `gx_circular_gauge_background_draw`, `gx_circular_gauge_create`
- `gx_circular_gauge_draw`, `gx_circular_gauge_event_process`
**gx_circular_gauge_background_draw**  
Draw circular gauge background

Prototype

```c
VOID gx_circular_gauge_background_draw(GX_CIRCULAR_GAUGE *gauge);
```

Description

This service draws background of the specified circular gauge. This service is normally called internally by the `gx_circular_gauge_draw` function, but is exposed to the application to assist in writing custom drawing functions.

Parameters

- **gauge**: Pointer to circular gauge control block

Return Values

None

Allowed From

Threads

Example

```c
/* Draw circular gauge background. */
gx_circular_gauge_background_draw(&my_circular_gauge);
```

See Also

- `gx_circular_gauge_angle_get`, `gx_circular_gauge_angle_set`, `gx_circular_gauge_animation_set`, `gx_circular_gauge_create`, `gx_circular_gauge_draw`, `gx_circular_gauge_event_process`
**gx_circular_gauge_create**

Create circular gauge

**Prototype**

```c
UINT gx_circular_gauge_create(GX_CIRCULAR_GAUGE *gauge,
GX_CONST GX_CHAR *name,
GX_WIDGET *parent,
GX_CIRCULAR_GAUGE_INFO *info,
GX_RESOURCE_ID background_id,
ULONG style,
USHORT circular_gauge_id,
GX_VALUE xpos,
GX_VALUE ypos);
```

**Description**

This service creates a circular gauge widget with the specified properties.

**Parameters**

- **gauge**: Pointer to circular gauge control block
- **name**: Logical name of circular gauge widget
- **parent**: Pointer to the parent widget
- **info**: Pointer to the gauge information structure. Appendix I contains definition to GX_CIRCULAR_GAUGE_INFO structure
- **background_id**: Resource ID of circular gauge background pixelmap
- **style**: Style of circular gauge. Appendix D contains pre-defined general styles for all widgets as well as widget specific styles.
- **circular_gauge_id**: Application-defined ID of circular gauge
- **xpos**: Gauge x-coordinate position
- **ypos**: Gauge y-coordinate position

**Return Values**

- **GX_SUCCESS** (0x00) Successful checkbox select
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_SIZE** (0x19) Invalid control block size
- **GX_ALREADY_CREATED** (0x13) Widget already created

**Allowed From**
Initialization and threads

Example

GX_CIRCULAR_GAUGE gauge_info;

gauge_info.gx_circular_gauge_info_animation_steps = 30;
gauge_info.gx_circular_gauge_info_animation_delay = 1;
gauge_info.gx_circular_gauge_info_needle_xpos = 140;
gauge_info.gx_circular_gauge_info_needle_ypos = 140;
gauge_info.gx_circular_gauge_info_NEEDLE_xcor = 20;
gauge_info.gx_circular_gauge_info_NEEDLE_ycor = 88;
gauge_info.gx_circular_gauge_info_NEEDLE_pixelmap = GX_PIXELMAP_ID_NEEDLE;

/* Create "my_gauge". */
status = gx_circular_gauge_create(&my_gauge, "my_gauge",
                                   &my_parent,
                                   &gauge_info, MY_PIXELMAP_RESOURCE_ID, GX_NULL,
                                   MY_ICON_ID, 5, 30);

/* If status is GX_SUCCESS the circular gauge "my_gauge" has been created. */

See Also

gx_circular_gauge_angle_get, gx_circular_gauge_angle_set,
gx_circular_gauge_animation_set, gx_circular_gauge_background_draw,
gx_circular_gauge_draw, gx_circular_gauge_event_process
**gx_circular_gauge_draw**

Draw circular gauge

**Prototype**

```c
VOID gx_circular_gauge_draw(GX_CIRCULAR_GAUGE *gauge);
```

**Description**

This service draws the specified circular gauge. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom gauge widgets.

**Parameters**

- `gauge` Pointer to circular gauge control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom circular gauge draw function. */
VOID custom_gauge_draw(GX_CIRCULAR_GAUGE *gauge)
{
    /* Call default circular gauge draw. */
    gx_circular_gauge_draw(gauge);

    /* Add custom drawing here. */
}
```

**See Also**

- `gx_circular_gauge_angle_get`, `gx_circular_gauge_angle_set`,
- `gx_circular_gauge_animation_set`, `gx_circular_gauge_background_draw`,
- `gx_circular_gauge_create`, `gx_circular_gauge_event_process`
**gx_circular_gauge_event_process**

Process circular gauge event

**Prototype**

```c
UINT gx_circular_gauge_event_process(GX_CIRCULAR_GAUGE *gauge,
                                     GX_EVENT *event);
```

**Description**

This service processes an event for the specified circular gauge.

**Parameters**

- `gauge` Pointer to gauge control block
- `event_ptr` Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) Successful gauge event process
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Threads

**Example**

```c
/* Call generic circular gauge event processing as part of custom event processing function. */

UINT custom_gauge_event_process(GX_CIRCULAR_GAUGE *gauge,
                                 GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;
    default:
        /* Pass all other events to the default circular gauge event processing */
        status = gx_circular_gauge_event_process(gauge, event);
        break;
    }
    return status;
```
See Also

gx_circular_gauge_angle_get, gx_circular_gauge_angle_set,
gx_circular_gauge_animation_set, gx_circular_gauge_background_draw,
gx_circular_gauge_create, gx_circular_gauge_draw
gx_context_brush_default

**Set brush of current drawing context**

**Prototype**

```c
UINT gx_context_brush_default(GX_DRAW_CONTEXT *context);
```

**Description**

This service sets the brush of the specified drawing context to default.

**Parameters**

- **context**: Pointer to context control block

**Return Values**

- **GX_SUCCESS**: (0x00) Successful creation
- **GX_PTR_ERROR**: (0x07) Invalid context pointer

**Allowed From**

Initialization and threads

**Example**

```
/* Set the brush of "my_context" to default. */
status = gx_context_brush_default(&my_context);

/* If status is GX_SUCCESS the brush of "my_context" has been set to default. */
```

**See Also**

- `gx_context_brush_define`, `gx_context_brush_get`, `gx_context_brush_set`,
- `gx_context_brush_style_set`, `gx_context_brush_pattern_set`,
- `gx_context_brush_width_set`, `gx_context_fill_color_set`, `gx_context_font_set`,
- `gx_context_line_color_set`, `gx_context_pixelmap_set`,
- `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`,
- `gx_context_raw_line_color_set`
**gx_context_brush_define**

Define brush of current drawing context

**Prototype**

```
UINT gx_context_brush_define(GX_RESOURCE_ID line_color_id,
                            GX_RESOURCE_ID fill_color_id,
                            UINT style);
```

**Description**

This service defines the brush of the current drawing context.

**Parameters**

- **line_color_id**: Resource ID of line color. Appendix B contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **fill_color_id**: Resource ID of fill color. Appendix B contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **style**: Style of brush. Appendix D describes the supported brush styles. Brush styles can be combined into one variable using bitwise OR operation.

**Return Values**

- **GX_SUCCESS**: (0x00) Successful context brush define
- **GX_INVALID_RESOURCE_ID**: (0x33) Invalid resource ID
- **GX_INVALID_CONTEXT**: (0x06) No active drawing context

**Allowed From**

Initialization and threads
Example

/* Define the brush of the current context. */
status = gx_context_brush_define(GX_COLOR_BLACK_ID,
                              GX_COLOR_BLACK_ID,
                              GX_STYLE_BORDER_NONE);

/* If status is GX_SUCCESS the brush of the current context has been defined. */

See Also

gx_context_brush_default, gx_context_brush_get, gx_context_brush_set,
gx_context_brush_pattern_set, gx_context_brush_style_set,
gx_context_brush_width_set, gx_context_fill_color_set, gx_context_font_set,
gx_context_line_color_set, gx_context_pixelmap_set,
gx_context_raw_brush_define, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
**gx_context_brush_get**

Get brush of current drawing context

**Prototype**

```
UINT gx_context_brush_get(GX_BRUSH **return_brush);
```

**Description**

This service returns a pointer to the active brush in the current drawing context. If there is no active drawing context, the service fails and returns a NULL pointer.

**Parameters**

- **return_brush**  
  Pointer to destination for brush.

**Return Values**

- **GX_SUCCESS**  
  (0x00) Successfully retrieved context brush
- **GX_INVALID_CONTEXT**  
  (0x06) No active drawing context
- **GX_PTR_ERROR**  
  (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```
GX_BRUSH *my_brush;

/* Get the brush of the current context. */
status = gx_context_brush_get(&my_brush);

/* If status is GX_SUCCESS the brush of the current context has been retrieved. */
```

**See Also**

- `gx_context_brush_default`, `gx_context_brush_define`, `gx_context_brush_set`, `gx_context_brush_style_set`, `gx_context_brush_pattern_set`, `gx_context_brush_width_set`, `gx_context_fill_color_set`, `gx_context_font_set`, `gx_context_line_color_set`, `gx_context_pixelmap_set`, `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`, `gx_context_raw_line_color_set`
**gx_context_brush_pattern_set**

Set brush pattern of current drawing context

**Prototype**

```c
UINT gx_context_brush_pattern_set(ULONG pattern);
```

**Description**

This service sets the brush pattern of the current drawing context.

The brush pattern is used for drawing dashed horizontal and dashed vertical lines. When the `gx_canvas_line_draw()` is called, and the line is horizontal or vertical, and the brush `gx_brush_line_pattern` field is non-zero, a pattern line is drawn.

The brush pattern mask is currently only supported for horizontal and vertical lines.

**Parameters**

- `pattern`  
  Pattern to be used for the brush. This is a simple hexadecimal on/off pattern to be used for pattern line drawing.

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successful context brush set

- **GX_INVALID_CONTEXT**  
  (0x06)  
  Invalid drawing context

**Allowed From**

Initialization and threads
Example

/* Set the brush pattern for the current context. */
status = gx_context_brush_pattern_set(0x80808080);

/* If status is GX_SUCCESS the brush pattern of the current context
has been set to the specified pattern. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_style_set, gx_context_brush_width_set,
gx_context_fill_color_set, gx_context_font_set, gx_context_line_color_set,
gx_context_pixelmap_set, gx_context_raw_brush_define,
gx_context_raw_fill_color_set, gx_context_raw_line_color_set
gx_context_brush_set

Set brush of current drawing context

Prototype

UINT gx_context_brush_set(GX_BRUSH *brush);

Description

This service sets the brush of the current drawing context.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brush</td>
<td>Pointer to brush to use for current context.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful context brush set</td>
</tr>
<tr>
<td>GX_INVALID_CONTEXT (0x06)</td>
<td>No active drawing context</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_BRSUH my_brush;
GX_FONT *font;
GX_COLOR fill_color;
GX_COLOR line_color;

/* Retrieve the font that associated with the specified font ID. */
gx_context_font_get(MY_FONT_ID, &font);

/* Retrieve the color that associated with the specified color ID. */
gx_context_color_get(MY_FILL_COLOR_ID, &fill_color);
gx_context_color_get(MY_LINE_COLOR, &line_color);

my_brush.gx_brush_pixelmap = MY_PIXELMAP_ID;
my_brush.gx_brush_font = font;
my_brush.gx_brush_line_pattern = 0x80808080;
my_brush.gx_brush_pattern_mask = 0x80000000;
my_brush.gx_brush_fill_color = fill_color;
my_brush.gx_brush_line_color = line_color;
my_brush.gx_brush_style = GX_BRSUH_SOLID_FILL | GX_BRUSH_ALIAS |
                          GX_BRUSH_PIECEMAP_FILL | GX_BRUSHROUND
my_brush.gx_brush_width = 2;
my_brush.gx_brush_alpha = 255;

/* Set the brush of the current context. */
status = gx_context_brush_set(my_brush);

/* If status is GX_SUCCESS the brush of the current context has been set. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_pattern_set, gx_context_brush_style_set,
gx_context_brush_width_set, gx_context_fill_color_set, gx_context_font_set,
gx_context_line_color_set, gx_context_pixelmap_set,
gx_context_raw_brush_define, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
**gx_context_brush_style_set**

Set brush style of current drawing context

**Prototype**

```c
UINT gx_context_brush_style_set(UINT style);
```

**Description**

This service sets the brush style of the current drawing context.

**Parameters**

- **style**
  
  Brush style of current context. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.

**Return Values**

- **GX_SUCCESS** (0x00) Successful context brush style set
- **GX_INVALID_CONTEXT** (0x06) No active drawing context

**Allowed From**

- Initialization and threads

**Example**

```c
/* Set the brush style of the current context. */
status = gx_context_brush_style_set(GX_BRUSH_ALIAS);

/* If status is GX_SUCCESS the brush style of the current context has been set. */
```

**See Also**

- `gx_context_brush_default`, `gx_context_brush_define`, `gx_context_brush_get`,
- `gx_context_brush_set`, `gx_context_brush_pattern_set`,
- `gx_context_brush_width_set`, `gx_context_fill_color_set`, `gx_context_font_set`,
- `gx_context_line_color_set`, `gx_context_pixelmap_set`,
- `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`,
- `gx_context_raw_line_color_set`
**gx_context_brush_width_set**

Set brush width of current drawing context

**Prototype**

```c
UINT gx_context_brush_width_set(UINT width);
```

**Description**

This service sets the width of the active brush in the current drawing context.

**Parameters**

- `width` Brush width in pixels of current context

**Return Values**

- **GX_SUCCESS** (0x00) Successful context brush width set
- **GX_INVALID_CONTEXT** (0x06) No active drawing context

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the brush width of the current context to 10 pixels. */
status = gx_context_brush_width_set(10);

/* If status is GX_SUCCESS the brush width of the current context has been set to 10. */
```

**See Also**

- `gx_context_brush_default`, `gx_context_brush_define`, `gx_context_brush_get`, `gx_context_brush_set`, `gx_context_brush_pattern_set`, `gx_context_brush_style_set`, `gx_context_fill_color_set`, `gx_context_font_set`, `gx_context_line_color_set`, `gx_context_pixelmap_set`, `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`, `gx_context_raw_line_color_set`
**gx_context_color_get**

Get color value associated with color ID in current draw context

**Prototype**

```c
UINT  gx_context_color_get(GX_RESOURCE_ID  color_id,
                         GX_COLOR *return_color);
```

**Description**

This service retrieves the color value associated with the indicated color ID. The color value is returned in the color format of the active context display. This service should only be called from within an active drawing operation.

**Parameters**

- **color_id**  Resource ID of color requested.
- **return_color**  Address of variable to hold returned color value.

**Return Values**

- **GX_SUCCESS**  (0x00)  Successful color value get
- **GX_INVALID_RESOURCE_ID**  (0x33)  Invalid resource ID
- **GX_INVALID_CONTEXT**  (0x06)  No active drawing context
- **GX_PTR_ERROR**  (0x07)  Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_COLOR color_value;

/* Get the color value. */
status = gx_context_color_get(MY_BLACK_COLOR_ID, &color_value);
```

**See Also**

`gx_context_brush_default`, `gx_context_brush_define`, `gx_context_brush_get`, `gx_context_brush_set`, `gx_context_brush_pattern_set`, `gx_context_brush_style_set`, `gx_context_brush_width_set`, `gx_context_font_set`, `gx_context_line_color_set`, `gx_context_pixelmap_set`, `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`, `gx_context_raw_line_color_set`
**gx_context_fill_color_set**  
Set fill color of current drawing context

**Prototype**

```c
UINT          gx_context_fill_color_set(GX_RESOURCE_ID  fill_color_id);
```

**Description**

This service sets the fill color of the active brush in the current drawing context.

**Parameters**

- **fill_color_id**  
  Resource ID of fill color of current context. **Appendix B** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successful context fill color set
- **GX_INVALID_RESOURCE_ID**  
  (0x33)  
  Invalid resource ID
- **GX_INVALID_CONTEXT**  
  (0x06)  
  No active drawing context

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the fill color of the current context to black. */
status = gx_context_fill_color_set(MY_BLACK_COLOR_ID);

/* If status is GX_SUCCESS the fill color of the current context has been set to black. */
```

**See Also**

- `gx_context_brush_default`
- `gx_context_brush_define`
- `gx_context_brush_get`
- `gx_context_brush_set`
- `gx_context_brush_pattern_set`
- `gx_context_brush_style_set`
- `gx_context_brush_width_set`
- `gx_context_font_set`
- `gx_context_line_color_set`
- `gx_context_pixelmap_set`
- `gx_context_raw_brush_define`
- `gx_context_raw_fill_color_set`
- `gx_context_raw_line_color_set`
**gx_context_font_get**

Get font associated with font ID in current draw context

**Prototype**

```c
UINT gx_context_font_get(GX_RESOURCE_ID font_id,
                         GX_FONT **return_font);
```

**Description**

This service retrieves the font pointer associated with the indicated font ID. This service should only be called from within an active drawing operation.

**Parameters**

- **font_id**
  Resource ID of font requested.
- **return_font**
  Address of variable to hold returned font pointer.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully retrieved font
- **GX_INVALID_RESOURCE_ID** (0x33) Invalid resource ID
- **GX_INVALID_CONTEXT** (0x06) No active drawing context
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

GX_FONT *my_font;

/* Get the font pointer. */
status = gx_context_font_get(MY_MIDSIZE_FONT, &my_font);

/* If status is GX_SUCCESS, the font that indicated with
MY_MIDSIZE_FONT has been successfully retrieved. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set, gx_context_font_set,
gx_context_line_color_set, gx_context_pixelmap_set,
gx_context_raw_brush_define, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
**gx_context_font_set**

Set font of current drawing context

**Prototype**

```c
UINT  gx_context_font_set(GX_RESOURCE_ID  font_id);
```

**Description**

This service sets the font in the active brush of the current drawing context.

**Parameters**

- **font_id**: Font resource ID of current context

**Return Values**

- **GX_SUCCESS** (0x00): Successful context font set
- **GX_INVALID_RESOURCE_ID** (0x33): Invalid resource ID
- **GX_INVALID_CONTEXT** (0x06): No active drawing context

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the font of the current context to MY_FONT_ID. */
status = gx_context_font_set(MY_FONT_ID);
/* If status is GX_SUCCESS the font of the current context has been set. */
```

**See Also**

- `gx_context_brush_default`, `gx_context_brush_define`, `gx_context_brush_get`,
- `gx_context_brush_set`, `gx_context_brush_pattern_set`,
- `gx_context_brush_style_set`, `gx_context_brush_width_set`,
- `gx_context_fill_color_set`, `gx_context_line_color_set`, `gx_context_pixelmap_set`,
- `gx_context_raw_brush_define`, `gx_context_raw_fill_color_set`,
- `gx_context_raw_line_color_set`
gx_context_line_color_set

Set line color of current drawing context

Prototype

UINT  gx_context_line_color_set(GX_RESOURCE_ID line_color_id);

Description

This service sets the line color of the active brush in the current drawing context.

Parameters

  line_color_id  Line color of current context. Appendix B contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

Return Values

  GX_SUCCESS   (0x00)  Successful context line color set
  GX_INVALID_RESOURCE_ID   (0x33)  Invalid resource ID
  GX_INVALID_CONTEXT   (0x06)  No active drawing context

Allowed From

Initialization and threads

Example

  /* Set the line color of the current context to black. */
  status = gx_context_line_color_set(GX_COLOR_BLACK_ID);
  /* If status is GX_SUCCESS the line color of the current context has been set to black. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set,
gx_context_fill_color_set, gx_context_font_set, gx_context_pixelmap_set,
gx_context_raw_brush_define, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
**gx_context_pixelmap_get**

Get pixelmap associated with pixelmap ID in current draw context

**Prototype**

```c
UINT gx_context_pixelmap_get(GX_RESOURCE_ID pixelmap_id, 
                             GX_PIXELMAP **return_map);
```

**Description**

This service retrieves the pixelmap pointer associated with the indicated pixelmap ID.

**Parameters**

- `pixelmap_id` Resource ID of pixelmap requested.
- `return_map` Address of variable to hold returned pixelmap address.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully retrieved pixelmap
- **GX_INVALID_RESOURCE_ID** (0x33) Invalid resource ID
- **GX_INVALID_CONTEXT** (0x06) No active drawing context
- **GX_PTR_ERROR** (0x07) Invalid pixelmap pointer

**Allowed From**

Initialization and threads
Example

GX_PIXELMAP *map;

/* Get the pixelmap pointer. */
status = gx_context_pixelmap_get(MY_PIXELMAP_ID, &map);

/* If status is GX_SUCCESS, the pixelmap was successfully retrieved. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set, gx_context_font_set,
gx_context_line_color_set, gx_context_pixelmap_set,
gx_context_raw_brush_define, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
**gx_context_pixelmap_set**

Set pixmap of current draw context

**Prototype**

```c
UINT  gx_context_pixelmap_set(GX_RESOURCE_ID  pixelmap_id);
```

**Description**

This service assigns the pixmap of the active brush in the current drawing context.

**Parameters**

- **pixelmap_id**: Pixmap resource ID to use for current context

**Return Values**

- **GX_SUCCESS** (0x00): Successful context pixmap set
- **GX_INVALID_RESOURCE_ID** (0x33): Invalid resource ID
- **GX_INVALID_CONTEXT** (0x06): Invalid context

**Allowed From**

- Initialization and threads

**Example**

```c
/* Set pixmap of the current context to MY_PIXELMAP_ID. */
status = gx_context_pixelmap_set(MY_PIXELMAP_ID);

/* If status is GX_SUCCESS the pixmap of the current context has been set. */
```

**See Also**

- `gx_context_brush_default`
- `gx_context_brush_define`
- `gx_context_brush_get`
- `gx_context_brush_set`
- `gx_context_brush_pattern_set`
- `gx_context_brush_style_set`
- `gx_context_brush_width_set`
- `gx_context_fill_color_set`
- `gx_context_font_set`
- `gx_context_line_color_set`
- `gx_context_raw_brush_define`
- `gx_context_raw_fill_color_set`
- `gx_context_raw_line_color_set`
**gx_context_raw_brush_define**

Define raw brush of current draw context

**Prototype**

```c
UINT gx_context_raw_brush_define(GX_COLOR line_color,
                                   GX_COLOR fill_color,
                                   UINT style);
```

**Description**

This service defines the raw brush of the current screen context. Raw definitions are used when 32-bit ARGB color values are to be passed into the brush rather than color IDs. Raw color definitions are useful when the desired color is not present in the current system color table or when the RGB color value is computed at runtime.

**Parameters**

- **line_color**
  Color of line in 32-bit raw ARGB color format. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.

- **fill_color**
  Color of fill in 32-bit raw ARGB color format. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.

- **style**
  Style of brush. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.

**Return Values**

- **GX_SUCCESS**
  (0x00) Successful context raw brush define

- **GX_INVALID_CONTEXT**
  (0x06) No active drawing context

**Allowed From**

Initialization and threads
Example

/* Define the raw brush of the current context. */
status = gx_context_raw_brush_define(GX_COLOR_BLACK,
                                        GX_COLOR_BLACK,
                                        GX_STYLE_BORDER_NONE);

/* If status is GX_SUCCESS the raw brush of the current context has
been defined. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set,
gx_context_fill_color_set, gx_context_font_set, gx_context_line_color_set,
gx_context_pixelmap_set, gx_context_raw_fill_color_set,
gx_context_raw_line_color_set
gx_context_raw_fill_color_set

Set raw fill color of current drawing context

Prototype

UINT gx_context_raw_fill_color_set(GX_COLOR line_color);

Description

This service sets the raw fill color of the current screen context. The line_color parameter is a 32-bit ARGB format raw color value, rather than a color ID value. Raw color definitions are useful when the desired color is not present in the current system color table or when the RGB color value is computed at runtime.

Parameters

| line_color | Color of line. Appendix A contains pre-defined colors. Note that the application may add custom colors as well. |

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful context raw fill color set</td>
</tr>
<tr>
<td>GX_INVALID_CONTEXT (0x06)</td>
<td>No active drawing context</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Set the raw fill color of the current context. */
status = gx_context_raw_fill_color_set(GX_COLOR_BLACK);
/* If status is GX_SUCCESS the raw fill color of the current context has been set. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set,
gx_context_fill_color_set, gx_context_font_set, gx_context_line_color_set,
gx_context_pixelmap_set, gx_context_raw_brush_define,
gx_context_raw_line_color_set
**gx_context_raw_line_color_set**  
Set raw line color of current drawing context

**Prototype**

```c
UINT gx_context_raw_line_color_set(GX_COLOR line_color);
```

**Description**

This service sets the line color of the active brush in the current drawing context. The `line_color` parameter is a 32-bit ARGB format raw color value, rather than a color ID value. Raw color definitions are useful when the desired color is not present in the current system color table or when the RGB color value is computed at runtime.

**Parameters**

| **line_color** | Color of line value. Appendix A contains pre-defined colors. Note that the application may add custom colors as well. |

**Return Values**

| **GX_SUCCESS**    | (0x00) Successful context raw line color set |
| **GX_INVALID_CONTEXT** | (0X06) No active drawing context |

**Allowed From**

Initialization and threads
Example

/* Set the raw line color of the current context. */
status = gx_context_raw_line_color_set(GX_COLOR_BLACK);

/* If status is GX_SUCCESS the raw line color of the current context has been set. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get,
gx_context_brush_set, gx_context_brush_pattern_set,
gx_context_brush_style_set, gx_context_brush_width_set,
gx_context_fill_color_set, gx_context_font_set, gx_context_line_color_set,
gx_context_pixelmap_set, gx_context_raw_brush_define,
gx_context_raw_fill_color_set
**gx_context_string_get**

Retrieve string associated with String ID (deprecated)

**Prototype**

```c
UINT gx_context_string_get(GX_RESOURCE_ID string_id, 
                          GXCONST GX_CHAR **return_string)
```

**Description**

This deprecated API returns the string associated with the given string ID. New applications should use gx_context_string_get_ext().

**Parameters**

- **string_id**
  String ID generated by the GUIX Studio application.

- **return_string**
  Address of variable to return string pointer.

**Return Values**

- **GX_SUCCESS** (0x00) Successful context raw line color set
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_CONTEXT** (0X06) No active drawing context

**Allowed From**

Initialization and threads
Example

GX_CHAR *text;

status = gx_context_string_get(GX_ID_ERROR, &text);

/* If status is GX_SUCCESS the string pointer has been returned. */

See Also

gx_context_string_get_ext
**gx_context_string_get_ext**

Retrieve string associated with given string ID.

**Prototype**

```c
UINT  gx_context_string_get_ext(GX_RESOURCE_ID string_id,
                                GX_STRING *return_string);
```

**Description**

This service returns the string associated with a given string ID. This service can only be invoked when there is an active drawing context, i.e. from within the drawing function of a widget. This service identifies the active canvas and display and retrieves the string from the located display instance.

**Parameters**

- **string_id**
  String ID used to identify the string, as generated by GUIX Studio in the application resource header file.

- **return_string**
  Address of GX_STRING variable in which the string pointer and string length will be returned.

**Return Values**

- **GX_SUCCESS**
  (0x00) Successful context raw line color set

- **GX_PTR_ERROR**
  (0x07) Invalid pointer

- **GX_CALLER_ERROR**
  (0x11) Invalid caller of this function

- **GX_INVALID_CONTEXT**
  (0x06) No active drawing context

**Allowed From**

Initialization and threads
Example

GX_STRING string;

/* Set the raw line color of the current context. */
status = gx_context_string_get_ext(ID_ERROR, &string);

/* If status is GX_SUCCESS the 
string.gx_string_ptr and string.gx_string_length 
values have been returned. */

See Also

gx_context_brush_default, gx_context_brush_define, gx_context_brush_get, 
gx_context_brush_set, gx_context_brush_pattern_set, 
gx_context_brush_style_set, gx_context_brush_width_set, 
gx_context_fill_color_set, gx_context_font_set, gx_context_line_color_set, 
gx_context_pixelmap_set, gx_context_raw_brush_set, 
gx_context_raw_fill_color_set
**gx_display_active_language_set**

Assign the display active language

**Prototype**

```c
UINT gx_display_active_language_set(GX_DISPLAY *display,
                                   GX_UBYTE language);
```

**Description**

This service assigns the currently active language for the indicated display. The language index corresponds to the languages defined in the display language table, and is not an ANSI language identifier.

Different displays in a multi display system can each run different active languages. The display language table should be assigned before this API is used. When a display is initialized using `gx_studio_display_configure`, the language table is automatically installed and the application passes in the active language index.

**Parameters**

- **display**: Pointer to display control block
- **language**: Active language index

**Return Values**

- **GX_SUCCESS** (0x00): Successful language assign
- **GX_PTR_ERROR** (0x07): Invalid display pointer
- **GX_INVALID_VALUE** (0x22): Invalid language index

**Allowed From**

Initialization and threads

**Example**

```c
/* Change value of color MY_COLOR_ID. */
status = gx_display_active_language_set(&my_display,
                                       LANGUAGE_ENGLISH);

/* If status is GX_SUCCESS the active language has been assigned. */
```

**See Also**

`gx_display_language_table_set`, `gx_studio_display_configure`
gx_display_color_set

Re-assign one color value

Prototype

UINT  gx_display_color_set(GX_DISPLAY *display,
               GX_RESOURCE_ID color_id,
               GX_COLOR new_color);

Description

This service re-assigns the color value associated with the specified
color ID. This can be used to modify the color table of a display
without providing an entirely new color table. The color value
provided must be in the native format supported by the display.

Parameters

display Pointer to display control block
color_id Color ID to reassign
new_color Color value to assign to this color_id slot

Return Values

GX_SUCCESS (0x00) Successful color reassign
GX_INVALID_Resource_ID (0x33) Invalid color ID
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_INVALID_DISPLAY (0x1D) Invalid display

Allowed From

Initialization and threads

Example

/* Change value of color MY_COLOR_ID. */
status = gx_display_color_set(&my_display, MY_COLOR_ID, 0x5454);

/* If status is GX_SUCCESS the color has been reassigned. */

See Also

gx_display_color_table_set
**gx_display_color_table_set**

Assign display color table

**Prototype**

```c
UINT  gx_display_color_table_set(GX_DISPLAY *display,
                                   GX_COLOR *color_table, INT color_count);
```

**Description**

This service re-assigns the color table to be used by the display. This service is normally invoked by the GUIX Studio generated display configuration function, but can also be called by the application software.

**Parameters**

- **display**
  Pointer to display control block
- **color_table**
  Array of color values in display native format.
- **color_count**
  Indicates number of entries in color table

**Return Values**

- **GX_SUCCESS**  (0x00)  Successful color table set
- **GX_PTR_ERROR** (0x07)  Invalid pointer
- **GX_CALLER_ERROR** (0x11)  Invalid caller of this function
- **GX_INVALID_DISPLAY** (0x1D)  Invalid display

**Allowed From**

- Initialization and threads

**Example**

```c
GX_COLOR default_table[32] = { ... };

/* Change the color table */
status = gx_display_color_table_set(&my_display, default_table, 32);

/* If status is GX_SUCCESS the color table has been reassigned. */
```

**See Also**

- `gx_display_color_set`
gx_display_create

Create display

Prototype

UINT  gx_display_create(GX_DISPLAY *display, GX_CONST CHAR *name,
                         UINT (*display_driver_setup)(GX_DISPLAY *),
                         GX_VALUE width, GX_VALUE height);

Description

This service creates a display and calls the display driver setup function. GUIX takes this display and adds it to its internal list of displays.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Pointer to display control block</td>
</tr>
<tr>
<td>name</td>
<td>Name of the display</td>
</tr>
<tr>
<td>display_driver_setup</td>
<td>Pointer to display driver setup function</td>
</tr>
<tr>
<td>optional_driver_info</td>
<td>Pointer to optional driver information</td>
</tr>
<tr>
<td>color_format</td>
<td>Color format, as defined in Appendix C</td>
</tr>
<tr>
<td>width</td>
<td>Number of pixels on the x-axis</td>
</tr>
<tr>
<td>height</td>
<td>Number of pixels on the y-axis</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)  Successful display create</td>
</tr>
<tr>
<td>GX_SYSTEM_ERROR</td>
<td>(0xFE)  Fail to setup display</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)  Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)  Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x23)  Invalid display control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_DISPLAY my_display;

UINT my_driver_setup_callback(GX_DISPLAY *display)
{
    ... 
}

/* Create screen "my_display". */
status = gx_display_create(&my_display, "my display",
                         my_driver_setup_callback, 320, 480);

/* If status is GX_SUCCESS, the screen "my_display" has been created. */

See Also

gx_display_delete
gx_display_delete

Destroy display

Prototype

UINT  gx_display_delete(GX_DISPLAY *display,
                 VOID (*display_driver_cleanup)(GX_DISPLAY *))

Description

This service shuts down a display, and cleans up allocated resources.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Pointer to display control block</td>
</tr>
<tr>
<td>display_driver_cleanup</td>
<td>Pointer to display driver cleanup function</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful display delete</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x10)</td>
<td>Created display list is NULL</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

VOID driver_cleanup_callback(GX_DISPLAY *display)
{
    ...
}

/* Delete a display “my_display”. */
status = gx_display_delete(&my_display, driver_cleanup_callback);

/* If status is GX_SUCCESS the screen “my_display” has been destroyed. */

See Also

gx_canvas_block_move, gx_canvas_line_draw, gx_canvas_pixelmap_draw,
gx_canvas_pixelmap_tile, gx_canvas_polygon_draw, gx_canvas_rectangle_draw,
gx_canvas_text_draw, gx_display_create
**gx_display_font_table_set**

Assign display font table

**Prototype**

```c
UINT  gx_display_font_table_set(GX_DISPLAY *display,
                                 GX_FONT **font_table, INT table_size);
```

**Description**

This service re-assigns the font table to be used by the display. This service is normally invoked by the GUIX Studio generated display configuration function, but can also be called by the application software.

**Parameters**

- **display**
  Pointer to display control block
- **font_table**
  Array of GX_FONT pointers.
- **table_size**
  Number of fonts in table

**Return Values**

- **GX_SUCCESS** (0x00) Successful font table set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_FONT *font_table[32] = { ... };
/* Assign font table */
status = gx_display_font_table_set(&my_display, font_table, 32);
/* If status is GX_SUCCESS, the font table has been reassigned. */
```

**See Also**

- `gx_display_color_set`
- `gx_display_color_table_set`
- `gx_display_pixels_table_set`
gx_display_language_table_get

Retrieve display language table (deprecated)

Prototype

UINT gx_display_language_table_get(GX DISPLAY *display,
                                   GX CHAR ****table, GX UBYTE *language_count,
                                   UINT *string_table_size);

Description

This service retrieves the language table from the indicated display.
This service can be used by an application to modify the display
language table, at runtime, using dynamically defined strings.

This API is deprecated and supported only for applications using the
old style language table (i.e. the Studio generated resource file is
generated for library version prior to version 5.6). New applications
should use gx_display_language_table_get_ext().

Parameters

display Pointer to display control block
table Address to receive table pointer
language_count Address to receive language count
string_table_size Address to receive string table size

Return Values

GX_SUCCESS (0x00) Successful font table set
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

GX_CHAR ***language_table;
GX UBYTE language_count;
UINT string_count;

/* Retrieve language table */
status = gx_display_language_table_get(&my_display,
                                       &language_table, &language_count, &string_count);

/* If status is GX_SUCCESS, the language table has been retrieved. */
See Also

gx_display_language_table_set, gx_display_active_language_set,
gx_display_string_get, gx_display_language_table_get_ext
**gx_display_language_table_get_ext**  
Retrieve display language table

**Prototype**

```c
UINT  gx_display_language_table_get_ext(GX_DISPLAY *display,
                GX_STRING ***table, GX_UBYTE *language_count,
                UINT *string_table_size);
```

**Description**

This service retrieves the language table from the indicated display. This service can be used by an application to modify the display language table, at runtime, using dynamically defined strings.

**Parameters**

- **display**: Pointer to display control block
- **table**: Address to receive table pointer
- **language_count**: Address to receive language count
- **string_table_size**: Address to receive string table size

**Return Values**

- **GX_SUCCESS** (0x00): Successful font table set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_STRING **language_table;
GX_UBYTE language_count;
UINT string_count;

/* Retrieve language table */
status = gx_display_language_table_get_ext(&my_display,
                                           &language_table, &language_count, &string_count);

/* If status is GX_SUCCESS, the language table has been retrieved. */
```

**See Also**

- `gx_display_language_table_set_ext`
- `gx_display_active_langauge_set`
- `gx_display_string_get_ext`
### gx_display_language_table_set

**Assign display language table (deprecated)**

**Prototype**

```c
UINT gx_display_language_table_set(GX_DISPLAY *display,
                                 GX_CHAR ***table,
                                 GX_UBYTE number_of_languages,
                                 UINT number_of_strings);
```

**Description**

This service is deprecated and new applications should use \( \text{gx_display_language_table_set_ext}() \). This service is supported only for compatibility with Studio generated resources files targeting library versions prior to version 5.6.

This service assigns the language table to be used by the display. This service is normally invoked by the GUIX Studio generated function \( \text{gx_studio_display_configure} \), but can also be called by the application software.

**Parameters**

- **display**
  - Pointer to display control block
- **table**
  - Language table
- **number_of_languages**
  - Number of columns in the provided table
- **number_of_strings**
  - Number of strings in each table column

**Return Values**

- **GX_SUCCESS** (0x00)
  - Successful font table set
- **GX_CALLER_ERROR** (0x11)
  - Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_CHAR ***language_table= my_app_language_table;

/* Assign language table */
status = gx_display_language_table_set(&my_display, language_table, 5, 232);
```
/* If status is GX_SUCCESS, the language table has been reassigned. */

See Also

gx_display_active_language_set, gx_display_string_get
gx_display_language_table_set_ext

Assign display language table

Prototype

UINT gx_display_language_table_set_ext(GX_DISPLAY *display,
            GX_STRING **table,
            GX_UBYTE number_of_languages,
            UINT number_of_strings);

Description

This service assigns the language table to be used by the display. This service is normally invoked by the GUIX Studio generated function gx_studio_display_configure, but can also be called by the application software.

Runtime language table assignment is usually done when languages are loaded from a binary resource file using gx_binres_language_table_load().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>Pointer to display control block</td>
</tr>
<tr>
<td>table</td>
<td>Language table</td>
</tr>
<tr>
<td>number_of_languages</td>
<td>Number of columns in the provided table</td>
</tr>
<tr>
<td>number_of_strings</td>
<td>Number of strings in each table column</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00)  
  Successful font table set
- **GX_CALLER_ERROR** (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)  
  Invalid pointer
- **GX_INVALID_STRING_LENGTH** (0x34)  
  Invalid string length

Allowed From

Initialization and threads

Example

GX_STRING **language_table = my_app_language_table;

/* Assign the language table */
status =
gx_display_language_table_set_ext(&my_display, language_table, 5, 132);
/ * If status is GX_SUCCESS, the language table has been reassigned. */

See Also

gx_display_active_language_set, gx_display_string_get
**gx_display_pixelmap_table_set**  
Assign display font table

**Prototype**

```c
UINT gx_display_pixelmap_table_set(GX_DISPLAY *display,
                                 GX_PIXELMAP **pixelmap_table, INT table_size);
```

**Description**

This service re-assigns the pixelmap table to be used by the display. This service is normally invoked by the Studio generated display configuration function, but can also be called by the application software.

**Parameters**

- **display**  
  Pointer to display control block
- **pixelmap_table**  
  Array of GX_PIXELMAP pointers.
- **table_size**  
  Number of pixelmaps in table

**Return Values**

- **GX_SUCCESS**  
  (0x00) Successful set pixelmap table
- **GX_CALLER_ERROR**  
  (0x11) Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x7) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_PIXELMAP *pixelmap_table[32] = { ... };

/* Assign pixelmap table */
status = gx_display_pixelmap_table_set(&my_display, pixelmap_table, 32);

/* If status is GX_SUCCESS the pixelmap table has been reassigned. */
```

**See Also**

- `gx_display_color_set`
- `gx_display_color_table_set`
- `gx_display_font_table_set`
**gx_display_string_get**

Retrieve a string from the active string table (deprecated)

**Prototype**

```c
UINT gx_display_string_get(GX_DISPLAY *display,
                            GX_RESOURCE_ID string_id,
                            GX_CONST GX_CHAR **string);
```

**Description**

This service is deprecated in favor of `gx_display_string_get_ext()`.

This service retrieves a string from the active string table for the indicated display. The active language is used to select the string from the language table assigned to the display.

String IDs are generated by GUIX Studio and are found in the application resources.h header file.

**Parameters**

- **display**
  Pointer to display control block
- **string_id**
  String ID, generated by GUIX Studio.
- **string**
  Address of string pointer variable

**Return Values**

- **GX_SUCCESS** (0x00) Successful string retrieval
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_RESOURCE_ID** (0x33) Invalid string Id
- **GX_PTR_ERROR** (0x07) Invalid display pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_CHAR *string;

/* Retrieve string value */
status = gx_display_string_get(&my_display,
                              GX_STRING_ID_MONDAY, &string);

/* If status is GX_SUCCESS, the string has been retrieved. */
```

**See Also**
gx_display_string_get_ext

Retrieve a string from the active string table

Prototype

UINT  gx_display_string_get_ext(GX_DISPLAY *display,
                                    GX_RESOURCE_ID string_id,
                                    GX_STRING *string);

Description

This service retrieves a string from the active string table for the
indicated display. The active language is used to select the string
from the language table assigned to the display.

String IDs are generated by GUIX Studio and are found in the
application resources.h header file.

Parameters

display        Pointer to display control block
string_id      String ID, generated by GUIX Studio.
string         Address of GX_STRING variable in
                which string.gx_string_ptr and
                string.gx_string_length will be returned.

Return Values

GX_SUCCESS        (0x00)   Successful string retrieval
GX_CALLER_ERROR   (0x11)   Invalid caller of this function
GX_INVALID_RESOURCE_ID (0x33) Invalid string Id
GX_PTR_ERROR      (0x07)   Invalid display pointer

Allowed From

Initialization and threads

Example

GX_STRING string;

/* Retrieve string value */
status = gx_display_string_get_ext(&my_display,
                                    GX_STRING_ID_MONDAY, &string);

/* If status is GX_SUCCESS, the string has been retrieved. */
See Also

gx_display_active_language_set, gx_display_language_table_set

gx_display_string_table_get

Retrieve the active string table (deprecated)

Prototype

UINT  gx_display_string_table_get(GX_DISPLAY *display,
       GX_UBYTE language, GX_CHAR **table, UINT
       *table_size);

Description

This service is deprecated and replaced by
gx_display_string_table_get_ext().

This service retrieves the string table associated with the active
language. This service is not frequently used, but is provided for
completeness for those applications that might need to make
runtime modifications to the string table.

Parameters

display Pointer to display control block
language Table column to retrieve.
table Address of variable to return pointer.
table_size Address of variable to return table size

Return Values

GX_SUCCESS (0x00) Successful font table set
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_NOT_FOUND (0x09) Invalid language index
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

GX_CHAR **string_table;
UINT table_size;

/* Retrieve string table */
status = gx_display_string_table_get(&my_display, LANGUAGE_ENGLISH,
       &string_table, &table_size);

/* If status is GX_SUCCESS, the string table has been retrieved. */
See Also

gx_display_color_set, gx_display_color_table_set, gx_display_pixelmap_table_set
### gx_display_string_table_get_ext

Retrieve the active string table

#### Prototype

```c
UINT gx_display_string_table_get(
    GX_DISPLAY *display,
    GX_UBYTE language, GX_STRING **table, UINT *table_size);
```

#### Description

This service retrieves the string table associated with the active language. This service is not frequently used, but is provided for completeness for those applications that might need to make runtime modifications to the string table.

#### Parameters

- **display**: Pointer to display control block
- **language**: Table column to retrieve.
- **table**: Address of variable to return pointer.
- **table_size**: Address of variable to return table size

#### Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful font table set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_NOT_FOUND</td>
<td>(0x09) Invalid language index</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

#### Allowed From

Initialization and threads

#### Example

```c
GX_STRING *string_table;
UINT table_size;

/* Retrieve string table */
status = gx_display_string_table_get_ext(&my_display,
    LANGUAGE_ENGLISH, &string_table, &table_size);

/* If status is GX_SUCCESS, the string table has been retrieved. */
```

#### See Also

- `gx_display_color_set`
- `gx_display_color_table_set`
- `gx_display_pixelmap_table_set`
**gx_display_theme_install**

Install themes to the specified display

Prototype

```c
UINT gx_display_theme_install(GX_DISPLAY *display,
                               GX_THEME *theme_table);
```

Description

This service install themes to the specified display. This service is normally invoked by the Studio generated display configuration function, but can also be called by the application software.

Parameters

- **display**
  Pointer to display control block
- **theme_table**
  Theme table to be installed

Return Values

- **GX_SUCCESS** (0x00) Successfully installed theme table
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid theme table pointer
- **GX_INVALID_DISPLAY** (0x1D) Invalid display

Allowed From

- Initialization and threads
Example

GX_THEME theme_1;
&theme_table[1] = {
    &theme_1,
    ...
}
/* Define resource tables. */
GX_COLOR     color_table[32] = {...};
GX_FONT      *font_table[32] = {...};
GX PIXELMAP  *pixelmap_table[32] = { ... };
/* Define scroll appearance. */
GX_SCROLLBARAppearance scroll_appearance;
memset(&scroll_appearance, 0, sizeof(GX_SCROLLBARAppearance));
scroll_appearance.gx_scroll_width = 20;
scroll_appearance.gx_scroll_thumb_width = 18;
scroll_appearance.gx_scroll_thumb_color =
    GX_COLOR_ID_SCROLL_BUTTON;
scroll_appearance.gx_scroll_thumb_border_color =
    GX_COLOR_ID_SCROLL_BUTTON;
scroll_appearance.gx_scroll_button_color =
    GX_COLOR_ID_SCROLL_BUTTON;
scroll_appearance.gx_scroll_thumb_travel_min = 20;
scroll_appearance.gx_scroll_thumb_travel_max = 20;
scroll_appearance.gx_scroll_thumb_border_style =
    GX_STYLE_BORDER_THIN;

theme_1.theme_color_table = color_table;
theme_1.theme_font_table = font_table;
theme_1.theme_pixelmap_table = pixelmap_table;
theme_1.theme_palette = GX_NULL;
theme_1.theme_vertical_scrollbar_appearance = scroll_appearance;
theme_1.theme_horizontal_scrollbar_appearance = scroll_appearance;
theme_1.theme_vertical_scroll_style = GX_SCROLLBAR_RELATIVE_THUMB;
theme_1.theme_horizontal_scroll_style =
    GX_SCROLLBAR_RELATIVE_THUMB;
theme_1.theme_color_table_size = 32;
theme_1.theme_font_table_size = 32;
theme_1.theme_pixelmap_table_size = 32;
theme_1.theme_palette_size = 0;

/* Install theme table. */
status = gx_display_theme_install(&my_display, theme_table);
/* If status is GX_SUCCESS the theme table has been installed. */

See Also

 gx_display_color_set, gx_display_color_table_set, gx_display_font_table_set
**gx_drop_list_close**

Close a drop list

Prototype

```c
UINT gx_drop_list_close(GX_DROP_LIST *drop_list);
```

Description

This service closes the popup list of the specified drop list.

Parameters

- **drop_list**
  Pointer to the drop list control block

Return Values

- **GX_SUCCESS** (0x00) Successful closed the drop list
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

```c
/* Close a drop list. */
status = gx_drop_list_close(&drop_list);
/* If status is GX_SUCCESS, the drop list is closed. */
```

See Also

gx_drop_list_create, gx_drop_list_event_process, gx_drop_list_open,
gx_drop_list_pixelmap_set, gx_drop_list_popup_get
gx_drop_list_create

Create a drop list

Prototype

UINT gx_drop_list_create(GX_DROP_LIST *drop_list,
                        GX_CONST GX_CHAR *name, GX_WIDGET *parent,
                        INT total_rows, INT open_height,
                        VOID (*callback)(GX_VERTICAL_LIST *, GX_WIDGET *, INT),
                        ULONG style, USHORT drop_list_id,
                        GX_CONST GX_RECTANGLE *size)

Description

This service creates a drop list. A drop list is a combination of the drop list widget, and a popup vertical list that is displayed when the drop-list is opened. The popup vertical list is created automatically when the drop-list widget is created, and displayed or hidden when the drop-list widget is opened or closed, respectively.

The drop list widget supports two associated pixelmaps. The first, described as "List Wallpaper" in the Studio properties view, is the optional wallpaper pixelmap that is displayed as the background of the vertical list that is displayed when the drop-list widget is opened. The second pixelmap, described as the "Background Image" in the Studio properties view, is an optional image displayed as the background of the drop-list itself.

A drop-list widget can have (but is not required to have) a child widget that is used to open and close the drop list. This is customarily an icon or button widget, but even a custom widget could be used as the open/close toggle for the parent drop list. The key setting which makes this child widget operate the drop list is that this child widget must have the pre-defined widget id ID_DROP_LIST_BUTTON.

To define a child widget which will open and close the drop-list, first add and child widget to the drop list, and position this child within the drop list as desired:
Then use the Studio properties view to assign this child widget the ID value ID_DROP_LIST_BUTTON, which is an internally defined ID value recognized by the parent drop list:

![Properties View](image)

### Parameters

- **drop_list** Pointer to the drop list control block
- **name** Name of the drop list
- **parent** Pointer to the parent widget
- **total_rows** Total number of rows in the drop list
- **open_height** The height of the vertical list displayed when the drop list is opened.
- **callback** Function called by the vertical list when the list is scrolled. Refer to the documentation of GX_VERTICAL_LIST for more information.
- **style** The drop-down list border style.
- **drop_list_id** Application-defined ID of the drop list
- **size** Dimensions of the drop list

### Return Values

- **GX_SUCCESS** (0x00) Successful drop list create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created

### Allowed From

Initialization and threads
Example

GX_DROP_LIST my_drop_list;
VOID list_row_create(GX_VERTICAL_LIST *list,
                GX_WIDGET *row, INT index);
{
        ...
}

GX_RECTANGLE size;

gx_utility_rectangle_define(&size, 10, 10, 80, 40);

status = gx_drop_list_create(&my_drop_list,
                "my drop list", GX_NULL,
                10, 100, list_row_create,
                GX_STYLE_BORDER_RECESSED | GX_STYLE_ENABLED,
                ID_DROP_LIST, &size)

/*! Is status is GX_SUCCESS, the drop list was successfully created.
 */

See Also:

gx_drop_list_close, gx_drop_list_event_process, gx_drop_list_open,
gx_drop_list_pixelmap_set, gx_drop_list_popup_get
gx_drop_list_event_process

Process drop list event

Prototype

UINT gx_drop_list_event_process(GX_DROP_LIST *list, GX_EVENT *event);

Description

This service processes an event for the drop list.

Parameters

- **drop_list**
  - Drop list widget control block
- **event**
  - Pointer to event to process

Return Values

- **GX_SUCCESS** (0x00)
  - Successfully processed drop list event
- **GX_CALLER_ERROR** (0x11)
  - Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer

Allowed From

Threads
Example

/* Call generic drop list event processing as part of custom event processing function. */

UINT custom_drop_list_event_process(GX_DROP_LIST *drop_list,
                                         GX_EVENT *event)
{
    UINT status = GX_SUCCESS;

    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;

    default:
        /* Pass all other events to the default drop list event processing */
        status = gx_drop_list_event_process(drop_list, event);
        break;
    }

    return status;
}

See Also

gx_drop_list_close, gx_drop_list_create, gx_drop_list_open,
gx_drop_list_pixelmap_set, gx_drop_list_popup_get
**gx_drop_list_open**

Open a drop list

**Prototype**

```c
UINT  gx_drop_list_open(GX_DROP_LIST *drop_list)
```

**Description**

This service opens a previously created drop list.

**Parameters**

- `drop_list`: Pointer to the drop list control block

**Return Values**

- **GX_SUCCESS** (0x00): Successful drop list create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_DROP_LIST mylist;

/* Open the popup list of “mylist”. */
status = gx_drop_list_open(&mylist);

/* If status == GX_SUCCESS, the popup list of “mylist” has been
displayed. */
```

**See Also**

- `gx_drop_list_close`, `gx_drop_list_create`, `gx_drop_list_event_process`, `gx_drop_list_pixelmap_set`, `gx_drop_list_popup_get`
gx_drop_list_pixelmap_set

Assign a background image to the drop list

Prototype

UINT gx_drop_list_pixelmap_set(GX_DROP_LIST *drop_list,
                               GX_RESOURCE_ID id);

Description

Assign a background image to the drop list. This pixelmap is used as the background for the drop list widget itself, and not for the popup vertical list that is displayed when the list is opened. To assign a pixelmap to the drop-list popup, you would need to first call gx_drop_list_popup_get to retrieve a pointer to the popup list, and then use gx_window_wallpaper_set() to assign a pixelmap to this popup list.

Parameters

drop_list Pointer to the drop list control block
id Resource ID to the pixlemap

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful drop list pixelmap set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GXINVALID_RESOURCE_ID</td>
<td>(0x33) Invalid pixlemap ID</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_DROP_LIST mylist;

/* create the drop list here */

/* Assign a pixmap id, which will turn on the list button */
status = gx_drop_list_pixelmap_set(&mylist,
        GX_PIXELMAP_ID_DROP_LIST_BACKGROUND);

/* If status is GX_SUCCESS, the pixmap was successfully set. */

See Also:

gx_drop_list_close, gx_drop_list_create, gx_drop_list_event_process,
gx_drop_list_open, gx_drop_list_popup_get
**gx_drop_list_popup_get**

Retrieve pointer to popup vertical list

**Prototype**

```c
UINT gx_drop_list_popup_get(GX_DROP_LIST *drop_list,
                           GX_VERTICAL_LIST **return_list)
```

**Description**

A drop-list widget is composed of the drop-list widget itself, and a popup vertical list that is shown when the drop-list widget is opened. This service retrieves a pointer to the vertical list component of the drop list, allowing the application to invoke API services directly on this vertical list.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop_list</td>
<td>Pointer to the drop list control block</td>
</tr>
<tr>
<td>return_list</td>
<td>Pointer to the vertical list stored in the drop list</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully retrieved popup vertical list</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_DROP_LIST drop_list;
GX_VERTICAL_LIST *vertical_list;

/* Retrieve the popup list of "drop_list". */
status = gx_drop_list_popup_get(&drop_list, &vertical_list)

/* If status is GX_SUCCESS, the popup list was successfully retrieved. */
```

**See Also:**

gx_drop_list_close, gx_drop_list_create, gx_drop_list_open, gx_drop_list_pixelmap_set


**gx_horizontal_list_children_position**

Position children for the horizontal list

**Prototype**

```c
UINT gx_horizontal_list_children_position(
    GX_HORIZONTAL_LIST *horizontal_list)
```

**Description**

This function positions the children for the horizontal list. This function is called automatically when the list receives the GX_EVENT_SHOW event, but should be called directly if the list is modified after it has been made visible.

**Parameters**

- **horizontal_list** Pointer to the horizontal list control block

**Return Values**

- **GX_SUCCESS** (0x00) Successful positioned the children for the horizontal list
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Position children in the horizontal list */
status = gx_horizontal_list_children_position (&horizontal_list);

/* If status is GX_SUCCESS the children in the horizontal list are positioned. */
```

**See Also**

gx_horizontal_list_create, gx_horizontal_list_event_process,
gx_horizontal_list_page_index_set, gx_horizontal_list_selected_index_get,
gx_horizontal_list_selected_widget_get, gx_horizontal_list_selected_set,
gx_horizontal_list_total_columns_set
**gx_horizontal_list_create**

Create horizontal list

**Prototype**

```c
UINT gx_horizontal_list_create(
    GX_HORIZONTAL_LIST *horizontal_list,
    GX_CONST GX_CHAR *name, GX_WIDGET *parent,
    INT total_columns,
    VOID (*callback)(GX_HORIZONTAL_LIST *, GX_WIDGET *, INT),
    ULONG style, USHORT horizontal_list_id,
    GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a horizontal list.

**Parameters**

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal_list</td>
<td>Horizontal list widget control block</td>
</tr>
<tr>
<td>name</td>
<td>Name of horizontal list</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>total_columns</td>
<td>Total number of columns in horizontal list</td>
</tr>
<tr>
<td>callback</td>
<td>This is a pointer to a callback function provided by the application. The callback function is invoked when the horizontal list is scrolled, to create the newly visible list items. In this way the horizontal list can display any user-defined widget type as the list items.</td>
</tr>
<tr>
<td>style</td>
<td>Style of scrollbar widget. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.</td>
</tr>
<tr>
<td>horizontal_list_id</td>
<td>Application-defined ID of horizontal list</td>
</tr>
<tr>
<td>size</td>
<td>Dimensions of the horizontal list</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successfully created the horizontal list</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13)</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19)</td>
<td>Invalid widget control block size</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22)</td>
<td>Number of columns not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Create horizontal list "my_list" with 5 columns. */
status = gx_horizontal_list_create(&my_list, "my_list", &my_parent,
                                   5, callback, GX_STYLE_WRAP, MY_LIST_ID,
                                   &size);

/* If status is GX_SUCCESS the horizontal list "my_list" has been created. */

See Also

gx_horizontal_list_children_position, gx_horizontal_list_event_process,
gx_horizontal_list_page_index_set, gx_horizontal_list_selected_index_get,
gx_horizontal_list_selected_widget_get, gx_horizontal_list_selected_set,
gx_horizontal_list_total_columns_set
**gx_horizontal_list_event_process**

Process horizontal list event

**Prototype**

```c
UINT  gx_horizontal_list_event_process(GX_HORIZONTAL_LIST *list,
                                       GX_EVENT *event);
```

**Description**

This service processes an event for the horizontal list.

**Parameters**

- **list**
  Horizontal list widget control block

- **event**
  Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) Successfully processed horizontal list event
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Call generic horizontal list event processing as part of custom event processing function. */

UINT custom_list_event_process(GX_HORIZONTAL_LIST *list,
   GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;
    default:
        /* Pass all other events to the default horizontal list event processing */
        status =
            gx_horizontal_list_event_process(list, event);
        break;
    }
    return status;
}

See Also

gx_horizontal_list_children_position, gx_horizontal_list_create,
gx_horizontal_list_page_index_set, gx_horizontal_list_selected_index_get,
gx_horizontal_list_selected_widget_get, gx_horizontal_list_selected_set,
gx_horizontal_list_total_columns_set
**gx_horizontal_list_page_index_set**

Set starting page index

**Prototype**

```c
UINT  gx_horizontal_list_page_index_set(GX_HORIZONTAL_LIST *list, INT *index);
```

**Description**

This service sets the starting index for the horizontal list.

**Parameters**

- **list**  
  Horizontal list widget control block
- **index**  
  The new top index

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successfully set starting page index for the horizontal list
- **GX_CALLER_ERROR**  
  (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  
  Invalid pointer
- **GX_INVALID_VALUE**  
  (0x22)  
  Invalid value

**Allowed From**

- Initialization and threads

**Example**

```c
/* Sets the starting page index of horizontal list "my_list" as 1. */
status = gx_horizontal_list_page_index_set(&my_list, 1);

/* If status is GX_SUCCESS the starting page index of "my_list" has been set. */
```

**See Also**

- `gx_horizontal_list_children_position`
- `gx_horizontal_list_create`
- `gx_horizontal_list_event_process`
- `gx_horizontal_list_selected_index_get`
- `gx_horizontal_list_selected_widget_get`
- `gx_horizontal_list_selected_set`
- `gx_horizontal_list_total_columns_set`
gx_horizontal_list_selected_index_get

Get selected entry index from horizontal list

Prototype

UINT gx_horizontal_list_selected_index_get(
    GX_horizontal_LIST *horizontal_list,
    INT *return_index);

Description

This service returns the selected list entry index of the horizontal list.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal_list</td>
<td>Horizontal list widget control block</td>
</tr>
<tr>
<td>return_index</td>
<td>Destination for return list index</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful obtained the horizontal list entry
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

```c
INT current_index_entry;

/* Get the list entry at the current index of horizontal list "my_list". */
status = gx_horizontal_list_selected_index_get(&my_list,
    &current_index_entry);

/* If status is GX_SUCCESS, "current_index_entry" contains the current list selection index. */
```

See Also

gx_horizontal_list_children_position, gx_horizontal_list_create,
gx_horizontal_list_event_process, gx_horizontal_list_page_index_set,
gx_horizontal_list_selected_widget_get, gx_horizontal_list_selected_set,
gx_horizontal_list_total_columns_set
**gx_horizontal_list_selected_set**

Assign the selected entry in a horizontal list

**Prototype**

```c
UINT gx_horizontal_list_selected_set(
    GX_HORIZONATAL_LIST *horizontal_list,
    INT index);
```

**Description**

This service assigns the selected entry in a horizontal list. If required, the horizontal list will scroll to make the selected entry visible.

**Parameters**

- `horizontal_list`: Horizontal list widget control block
- `index`: Index based position of new list entry

**Return Values**

- `GX_SUCCESS` (0x00): Successfully set the horizontal list entry
- `GX_FAILURE` (0x10): Index is not in invalid range
- `GX_CALLER_ERROR` (0x11): Invalid caller of this function
- `GX_PTR_ERROR` (0x07): Invalid pointer
- `GX_INVALID_WIDGET` (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the list entry of “my_list” to the child in line 12. */
status = gx_horizontal_list_selected_set(&my_list, 12);

/* If status is GX_SUCCESS, the list entry of “my_list” has been successfully set to 12. */
```

**See Also**

- `gx_horizontal_list_children_position`, `gx_horizontal_list_create`,
- `gx_horizontal_list_event_process`, `gx_horizontal_list_page_index_set`,
- `gx_horizontal_list_selected_index_get`, `gx_horizontal_list_selected_widget_get`,
- `gx_horizontal_list_total_columns_set`
**gx_horizontal_list_selected_widget_get**

Get selected entry from horizontal list

### Prototype

```c
UINT gx_horizontal_list_selected_widget_get(
    GX_horizontal_LIST *horizontal_list,
    GX_WIDGET **return_list_entry);
```

### Description

This service returns the selected list entry of the horizontal list. Note that if the horizontal list has more rows than child widgets, and the selected entry has been scrolled from view, this API will return GX_NULL because the child widgets are re-used as the list content is scrolled. The `gx_horizontal_list_selected_index_get` function will reliably return the index of the selected item, even if that item has been scrolled from view.

### Parameters

- **horizontal_list**
  Horizontal list widget control block
- **return_list_entry**
  Destination for return list entry widget

### Return Values

- **GX_SUCCESS** (0x00)
  Successful obtained the horizontal list entry
- **GX_FAILURE** (0x10)
  Selected widget has been scrolled from view in a list with more rows than client children.
- **GX_CALLER_ERROR** (0x11)
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  Invalid pointer

### Allowed From

Initialization and threads
Example

/* Get the list entry at the current index of horizontal list "my_list". */
status = gx_horizontal_list_selected_widget_get(&my_list, &current_list_entry);

/* If status is GX_SUCCESS, "current_list_entry" contains a pointer to the currently selected widget. */

See Also

gx_horizontal_list_children_position, gx_horizontal_list_create,
gx_horizontal_list_event_process, gx_horizontal_list_page_index_set,
gx_horizontal_list_selected_index_get, gx_horizontal_list_selected_set,
gx_horizontal_list_total_columns_set
gx_horizontal_list_total_columns_set

Assign the total number of list columns

Prototype

UINT gx_horizontal_list_total_columns_set(
    GX_HORIZONTAL_LIST *horizontal_list, INT count);

Description

This service assigns the total number of columns to be displayed by
the horizontal list.

Parameters

<table>
<thead>
<tr>
<th>horizontal_list</th>
<th>Horizontal list widget control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Number of columns to display</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful assigned column count
- **GX_CALLER_ERROR** (0x10) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid count value

Allowed From

Initialization and threads

Example

/* Tell my list to display 20 total columns. */
status = gx_horizontal_list_total_columns_set(&my_list, 20);

/* If status is GX_SUCCESS, the total columns of “my_list” was
successfully set to 20. */

See Also

gx_horizontal_list_children_position, gx_horizontal_list_create,
gx_horizontal_list_event_process, gx_horizontal_list_page_index_set,
gx_horizontal_list_selected_index_get, gx_horizontal_list_selected_widget_get,
gx_horizontal_list_selected_set
**gx_horizontal_scrollbar_create**

Create horizontal scrollbar

**Prototype**

```c
UINT gx_horizontal_scrollbar_create(GX_SCROLLBAR *scrollbar,
    GX_CONST GX_CHAR *name,
    GX_WINDOW *parent,
    GX_SCROLLBAR_APPEARANCE *appearance
    ULONG style);
```

**Description**

This service creates a horizontal scrollbar. The ID for a horizontal scrollbar is pre-defined (because a window has to know how to catch events from it), and the size is automatic (because it has to fill the parent window’s client width). If we decide to allow client area scrollbars, we will need to add another create function with the id and size parameters.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrollbar</td>
<td>Scrollbar widget control block</td>
</tr>
<tr>
<td>name</td>
<td>Name of scrollbar</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>appearance</td>
<td>The appearance structure defines the appearance of the scroll bar. If this value is GX_NULL, the scrollbar will use the default scrollbar appearance defined by gx_system_scroll_appearance_get. Refer to <strong>Appendix I</strong> for the definition of the GX_SCROLLBAR_APPEARANCE structure.</td>
</tr>
</tbody>
</table>

**Style**

Style of scrollbar widget.

*Appendix D* contains pre-defined general styles for all widgets as well as widget-specific styles.
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful horizontal scrollbar create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid widget control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Create horizontal scrollbar "my_scrollbar". */
status = gx_horizontal_scrollbar_create(&my_scrollbar,
                                          "my_horizontal_scrollbar", &my_parent,
                                          GX_NULL,
                                          GX_STYLE_SCROLLBAR_END_BUTTONS);

/* If status is GX_SUCCESS the horizontal scrollbar "my_scrollbar"
   has been created. */

See Also

gx_scrollbar_draw, gx_scrollbar_event_process, gx_scrollbar_limit_check,
gx_scrollbar_reset, gx_vertical_scrollbar_create
gx_icon_button_create

Prototype

UINT  gx_icon_button_create(GX_ICON_BUTTON *button,
                         GX_CONST GX_CHAR *name,
                         GX_WIDGET *parent,
                         GX_RESOURCE_ID icon_id,
                         ULONG style,
                         USHORT icon_button_id,
                         GX_CONST GX_RECTANGLE *size);

Description

This service creates the specified icon button widget.

GX_ICON_BUTTON is derived from GX_BUTTON and supports all
gx_button API services.

Parameters

button Pointer to icon button control block
name Logical name of icon button widget
parent Pointer to the parent widget
icon_id Resource ID of icon
style Style of icon. Appendix D contains pre-
defined general styles for all widgets as well
as widget-specific styles.
icon_button_id Application-defined ID of icon button
size Dimensions of icon button

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully created icon button</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid widget control block size</td>
</tr>
</tbody>
</table>
Allowed From

Initialization and threads

Example

/* Create “my_icon_button”. */
status = gx_icon_button_create(&my_icon_button, “my_icon_button”,
    &my_parent,
    MY_ICON_RESOURCE_ID, GX_STYLE_BORDER_RAISED,
    MY_ICON_BUTTON_ID,
    &size);

/* If status is GX_SUCCESS the icon button “my_icon_button” has been created. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select, gx_icon_create,
gx_icon_draw, gx_icon_pixelmap_set, gx_pixelmap_button_create,
gx_pixelmap_button_draw, gx_radio_button_create, gx_radio_button_draw
 gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
**gx_icon_button_draw**

**Draw an icon button**

**Prototype**

```c
VOID gx_icon_button_draw(GX_ICON_BUTTON *button);
```

**Description**

This service draws the icon button. This function is normally called internally by GUIX as part of a canvas refresh operation, but it also exposed to the application that might want to provide a custom drawing function and invoke the default icon button drawing as custom drawing base.

**Parameters**

- **button**: Pointer to icon button control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom icon draw function. */
void MyIconButtonDraw(GX_ICON_BUTTON *button)
{
    /* Do the normal drawing first */
    gx_icon_button_draw(button);

    /* Add custom drawing here */
}
```

**See Also**

- `gx_button_background_draw`, `gx_button_create`, `gx_button_deselect`, `gx_button_draw`, `gx_button_event_process`, `gx_button_select`, `gx_icon_create`, `gx_icon_draw`, `gx_icon_pixelmap_set`, `gx_pixelmap_button_create`, `gx_pixelmap_button_draw`, `gx_radio_button_create`, `gx_radio_button_draw`, `gx_text_button_create`, `gx_text_button_color_set`, `gx_text_button_draw`
**gx_icon_button_pixelmap_set**

Set pixelmap to the icon button widget

**Prototype**

```c
UINT gx_icon_button_pixelmap_set(GX_ICON_BUTTON *button,
                                GX_RESOURCE_ID icon_id);
```

**Description**

This service assigns a new pixelmap to the icon button widget.

**Parameters**

- **button**
  Pointers to icon button control block
  
- **icon_id**
  Resource ID of pixelmap

**Return Values**

- **GX_SUCCESS**
  (0x00)
  Successfully set icon button pixelmap

- **GX_CALLER_ERROR**
  (0x11)
  Invalid caller of this function

- **GX_PTR_ERROR**
  (0x07)
  Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Set pixelmap for “my_icon_button”. */
status = gx_icon_button_pixelmap_set(&my_icon_button,
                                     GX_PIXELMAP_ID_ICON);

/* If status is GX_SUCCESS, the pixelmap of the icon button “my_icon_button” has been set. */
```

**See Also**

- `gx_button_background_draw`
- `gx_button_create`
- `gx_button_deselect`
- `gx_button_draw`
- `gx_button_event_process`
- `gx_button_select`
- `gx_icon_create`
- `gx_icon_draw`
- `gx_icon_pixelmap_set`
- `gx_pixelmap_button_create`
- `gx_pixelmap_button_draw`
- `gx_radio_button_create`
- `gx_radio_button_draw`
- `gx_text_button_create`
- `gx_text_button_color_set`
- `gx_text_button_draw`
gx_icon_background_draw

Draw icon background

Prototype

VOID gx_icon_background_draw(GX_ICON *icon);

Description

This service draws background of the specified icon widget. This service is normally called internally by the gx_icon_button_draw function, but is exposed to the application to assist in writing custom drawing functions.

Parameters

icon Pointer to icon widget control block

Return Values

None

Allowed From

Initialization and threads

Example

/* Write a custom icon draw function. */
void MyIconButtonDraw(GX_ICON *icon)
{
    /* Call icon background draw. */
    gx_icon_background_draw(icon);

    /* Add custom drawing here */
    /* Draw child widgets. */
    gx_widget_children_draw(icon);
}

See Also

gx_icon_create, gx_icon_draw, gx_icon_event_process, gx_icon_pixelmap_set
**gx_icon_create**

Create icon

Prototype

```c
UINT  gx_icon_create(GX_ICON *icon, GX_CONST GX_CHAR *name, 
                     GX_WIDGET *parent, 
                     GX_RESOURCE_ID pixelmap_id, ULONG style, 
                     USHORT icon_id, GX_VALUE x, GX_VALUE y);
```

Description

This service creates the specified icon widget.

Parameters

- **icon** Pointer to icon control block
- **name** Logical name of icon widget
- **parent** Pointer to the parent widget
- **pixelmap_id** Resource ID of pixelmap
- **style** Style of icon
- **icon_id** Application-defined ID of icon
- **x** Starting x-coordinate position
- **y** Starting y-coordinate position

Return Values

- **GX_SUCCESS** (0x00) Successful icon create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size

Allowed From

Initialization and threads
Example

/* Create "my_icon". */
status = gx_icon_create(&my_icon, "my_icon", &my_parent,
    MY_PIXELMAP_RESOURCE_ID, GX_STYLE_BORDER_RAISED,
    MY_ICON_ID,
    5, 30);

/* If status is GX_SUCCESS the icon "my_icon" has been created. */

See Also

gx_icon_button_create, gx_icon_draw, gx_icon_pixelmap_set
**gx_icon_draw**

**Draw icon**

**Prototype**

```c
VOID gx_icon_draw(GX_ICON *icon);
```

**Description**

This service draws the specified icon widget. This service is normally called internally by GUIX as part of a canvas refresh operation, but it also exposed to the application that might want to provide a custom drawing function and invoke the default icon drawing as custom drawing base.

**Parameters**

- **icon**
  Pointer to icon widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom icon drawing function. */

VOID my_icon_draw(GX_MENU *menu)
{
    /* Call default icon draw. */
    gx_icon_draw(menu);

    /* Add your own drawing here. */
}
```

**See Also**

gx_icon_button_create, gx_icon_create, gx_icon_pixelmap_set
**gx_icon_event_process**

Icon widget event processing

**Prototype**

```c
UINT gx_icon_event_process(GX_ICON *icon, GX_EVENT *event_ptr);
```

**Description**

This service handles events sent to a GX_ICON widget.

**Parameters**

- **icon**: Pointer to icon widget control block
- **event_ptr**: Pointer to GX_EVENT structure

**Return Values**

- **GX_SUCCESS** (0x00): Successful processed icon event
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Threads

**Example**

```c
switch(event_ptr->gx_event_type)
{
    case GX_EVENT_SHOW:
        /* Do default handling. */
        status = gx_icon_event_process(icon, event_ptr);
        /* add your own handling here */
        break;
}
```

**See Also**

- `gx_icon_button_create`, `gx_icon_create`, `gx_icon_pixelmap_set`
**gx_icon_pixelmap_set**

Set pixelmap for icon

**Prototype**

```c
UINT  gx_icon_pixelmap_set(GX_ICON *icon,
                        GX_RESOURCE_ID normal_id,
                        GX_RESOURCE_ID selected_id);
```

**Description**

This service sets the pixelmap for the specified icon widget.

**Parameters**

- `icon` Pointer to icon widget control block
- `normal_id` Normal state Resource ID
- `selected_id` Selected state Resource ID

**Return Values**

- **GX_SUCCESS** (0x00) Successful icon pixelmap set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Set pixelmap for "my_icon". */
status = gx_icon_pixelmap_set(&my_icon,
                                MY_NOT_SELECTED_RESOURCE_ID,
                                MY_SELECTED_ID);

/* If status is GX_SUCCESS, the icon "my_icon" has a pixelmap set. */
```

**See Also**

- `gx_icon_button_create`, `gx_icon_create`, `gx_icon_draw`
**gx_image_reader_create**

Create image reader module instance

**Prototype**

```c
UINT gx_image_reader_create(
    GX_IMAGE_READER *image_reader,
    GX_CONST GX_UBYTE *data, INT data_size,
    GX_UBYTE color_format, GX_UBYTE mode);
```

**Description**

This function creates a runtime raw image reader / decoder. Currently only jpeg and png raw image types are supported. This service requires GX_SOFTWARE_DECODER_SUPPORT to be defined.

**Parameters**

- `image_reader`: Image reader control block
- `data`: Pointer to raw input data.
- `data_size`: Size of raw input data.
- `color_format`: The requested output color format.
- `mode`: Compress, dither and alpha modes flags.

**Return Values**

- **GX_SUCCESS** (0x00) Successful image reader create
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid data size

**Allowed From**

Initialization and Threads
Example

GX_IMAGE_READER my_image_reader;

"color format" could be set to:
    GX_COLOR_FORMAT_32ARGB
    GX_COLOR_FORMAT_24RGB
    GX_COLOR_FORMAT_565RGB
    GX_COLOR_FORMAT_8BIT_PALETTE

/* Create image reader "my_image_reader". */
status = gx_image_reader_create(&my_image_reader, decoder_data,
                        decoder_data_size,
                        GX_COLOR_FORMAT_32ARGB,
                        GX_IMAGE_READER_MODE_COMPRESS)

/* If status is GX_SUCCESS, "my_image_reader" has been successfully
   created. */

See Also

gx_image_reader_start, gx_image_reader_palette_set
**gx_image_reader_palette_set**

Define image reader palette

**Prototype**

```c
UINT  gx_image_reader_palette_set(
    GX_IMAGE_READER *image_reader,
    GX_COLOR *pal,
    UINT palsize);
```

**Description**

This service sets palette for image reader control block. This service requires GX_SOFTWARE_DECODER_SUPPORT to be defined.

**Parameters**

- `image_reader` Image reader control block
- `pal` Pointer to palette
- `palsize` The size of the palette

**Return Values**

- **GX_SUCCESS** (0x00) Successful image reader palette set
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid palette size

**Allowed From**

Initialization and Threads

**Example**

```c
/* Set “my_palette” as the palette of “my_image_reader”. */
status = gx_image_reader_palette_set(&my_image_reader, my_palette, my_palette_size);

/* If status is GX_SUCCESS the palette of “my_image_reader” has been set to “my_palette”. */
```

**See Also**

gx_image_reader_create, gx_image_reader_start
**gx_image_reader_start**

Start the decompress and conversion process

**Prototype**

```c
UINT gx_image_reader_start(GX_IMAGE_READER *image_reader,
                           GX_PIXELMAP *outmap);
```

**Description**

This service decodes a raw image to a specified color format. Currently only jpeg and png raw image types are supported. This requires GXSOFTWARE_DECODER_SUPPORT to be defined.

**Parameters**

- `image_reader` Image reader control block
- `pixelmap` Output pixelmap

**Return Values**

- **GX_SUCCESS** (0x00) Successful image decoding
- **GX_SYSTEM_MEMORY_ERROR** (0x30) Memory allocator is not defined or memory allocation failed
- **GX_NOT_SUPPORTED** (0x28) Input image type or output color format is not supported
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and Threads
Example

GX_IMAGE_READER my_image_reader;
GX_PIXELMAP output_map;

/* Create my_image_reader here. */

/* Decoding a raw image according to the settings of
 "my_image_reader". */
status = gx_image_reader_start(&my_image_reader, output_map)

/* If status is GX_SUCCESS "output_map" have been loaded with the
 requested pixelmap. */

See Also

gx_image_reader_create, gx_image_reader_palette_set
**gx_line_chart_axis_draw**

Draw line chart x,y axis

**Prototype**

VOID  gx_line_chart_axis_draw(GX_LINE_CHART *chart)

**Description**

This service draws the x,y axis of a line chart. The axis colors and line width parameters are retrieved from the line chart information structure.

This service is normally called internally by the gx_line_chart_draw function, but is exposed to the application to assist in writing custom drawing functions.

**Parameters**

- **chart**
  Line chart control block.

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom chart drawing function. */

VOID my_chart_draw(GX_LINE_CHART *chart)
{
   /* Call default window draw. */
   gx_window_draw((GX_WINDOW *)chart);

   /* Draw the line chart axis. */
   gx_line_chart_axis_draw(&chart->gx_line_chart_info);

   /* Draw the data line */
   gx_line_chart_data_draw(&chart->gx_line_chart_info);

   /* Add your own drawing here. */
}
```

**See Also**

gx_line_chart_create, gx_line_chart_data_draw, gx_line_chart_draw,
gx_line_chart_update, gx_line_chart_y_scale_calculate
**gx_line_chart_create**

Create GX_LINE_CHART widget

### Prototype

```c
UINT gx_line_chart_create(GX_LINE_CHART *chart,
GX_CONST GX_CHAR *name,
GX_WIDGET *parent,
GX_LINE_CHART_INFO *info,
ULONG style,
USHORT chart_id,
GX_RECTANGLE *size)
```

### Description

This service creates a line chart window. The chart drawing parameters and chart data are passed in via the GX_LINE_CHART_INFO structure.

GX_LINE_CHART is based on GX_WINDOW, and supports all of the GX_WINDOW APIs.

### Parameters

- **chart**: Pointer to the GX_LINE_CHART control block.
- **name**: Optional line chart name
- **parent**: Parent widget, or GX_NULL
- **info**: Structure defining line chart drawing parameters. Appendix I contains definition to GX_LINE_CHART_INFO structure.
- **style**: Widget style flags
- **chart_id**: Chart logical ID value
- **size**: Chart window bounding rectangle

### Return Values

- **GX_SUCCESS** (0x00): Successful line chart create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
- **GX_ALREADY_CREATED** (0x13): Widget already created
Allowed From

Initialization and threads

Example

/* Create a line chart */
GX_LINE_CHART chart;
GX_RECTANGLE chart_size;
GX_LINE_CHART_INFO gx_chart_info;
GX_WINDOW_ROOT *root_window;

chart_size = root_window->gx_widget_size;

/* Initialize params for the GUIX base chart. */
gx_chart_info.gx_line_chart_min_val = DATA_MIN_VAL;
gx_chart_info.gx_line_chart_max_val = DATA_MAX_VAL;
gx_chart_info.gx_line_chart_max_data_count = MAX_DATA_COUNT;
gx_chart_info.gx_line_chart_active_data_count = 0;
gx_chart_info.gx_line_chart_axis_line_width = AXIS_LINE_WIDTH;
gx_chart_info.gx_line_chart_data_line_width = DATA_LINE_WIDTH;
gx_chart_info.gx_line_chart_data = chart_data;
gx_chart_info.gx_line_chart_line_color = GX_COLOR_ID_DATA_LINE;
gx_chart_info.gx_line_chart_axis_color = GX_COLOR_ID_AXIS_LINE;

/* Leave room for labels on bottom and right. */
gx_chart_info.gx_line_chart_left_margin = 0;
gx_chart_info.gx_line_chart_top_margin = 0;
gx_chart_info.gx_line_chart_right_margin = 80;
gx_chart_info.gx_line_chart_bottom_margin = 32;

status = gx_line_chart_create(&chart, "Line Chart", root_window,
&gx_chart_info, GX_STYLE_NONE, &chart_size);

/* If status is GX_SUCCESS, the "chart" has been successfully created. */

See Also

gx_line_chart_create, gx_line_chart_data_draw, gx_line_chart_draw,
gx_line_chart_update, gx_line_chart_y_scale_calculate
**gx_line_chart_data_draw**

**Prototype**

```c
VOID gx_line_chart_data_draw(GX_LINE_CHART *chart)
```

**Description**

This service draws the line chart data line. The line colors and line width parameters are retrieved from the line chart information structure.

This service is normally called internally by the `gx_line_chart_draw` function, but is exposed to the application to assist in writing custom drawing functions.

**Parameters**

- **chart**
  Line chart control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom chart drawing function. */

VOID my_chart_draw(GX_LINE_CHART *chart)
{
    /* Call default window draw. */
    gx_window_draw((GX_WINDOW *)chart);

    /* Draw the line chart axis. */
    gx_line_chart_axis_draw(chart);

    /* Draw the data line. */
    gx_line_chart_data_draw(chart);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_line_chart_create`
- `gx_line_chart_draw`
- `gx_line_chart_update`
- `gx_line_chart_y_scale_calculate`
**gx_line_chart_draw**

**Draw the line chart**

**Prototype**

```c
UINT gx_line_chart_draw(GX_LINE_CHART *chart)
```

**Description**

This is the default line chart drawing function, which draws the chart axis and data line. Applications usually provide a custom drawing function to replace the default drawing to add things such as tickmarks, scale, or other information to the chart axis and data line drawn by the base line chart widget.

**Parameters**

- **chart**
  Pointer to the line chart control block.

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom chart drawing function. */

VOID my_chart_draw(GX_LINE_CHART *chart)
{
    /* Call default line chart draw. */
    gx_line_chart_draw(chart);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_line_chart_create`
- `gx_line_chart_draw`
- `gx_line_chart_update`
- `gx_line_chart_y_scale_calculate`
**gx_line_chart_update**

Update line chart data line

**Prototype**

```c
UINT gx_line_chart_update(GX_LINE_CHART *chart, INT *data,
                           INT data_count)
```

**Description**

This service updates the data array plotted by the line chart window, and forces the window to redraw.

**Parameters**

- **chart**: Line chart control block
- **data**: Data array to be plotted
- **data_count**: Size of the data array

**Return Values**

- **GX_SUCCESS** (0x00): Successful text button create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
INT chart_data[100];
GX_LINE_CHART chart;

/* Update the data array associated with “chart”. */
status = gx_line_chart_update(&chart, chart_data, 100);

/* If status is GX_SUCCESS, the line chart data has been updated. */
```

**See Also**

- `gx_line_chart_create`
- `gx_line_chart_data_draw`
- `gx_line_chart_draw`
- `gx_line_chart_y_scale_calculate`
**gx_line_chart_y_scale_calculate**

Calculate fixed-point y axis scaling value

**Prototype**

```c
UINT gx_line_chart_y_scale_calculate(GX_LINE_CHART *chart,
                                      INT *return_val)
```

**Description**

This service calculates the fixed-point scaling value used to plot data values on the chart Y axis. The chart_info parameters and chart bounding rectangle are used to calculate this scaling value.

**Parameters**

- **chart**: Line chart control block
- **return_val**: Address of value to hold fixed-point return value.

**Return Values**

- **GX_SUCCESS** (0x00): Successful y scale value calculate
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_LINE_CHART chart;
INT y_scale;

/* Calculate y scale value of "chart". */
status = gx_line_chart_y_scale_calculate(&chart, &y_scale);

/* If status is GX_SUCCESS, y scale value of "chart" has been calculated. */
```

**See Also**

gx_line_chart_create, gx_line_chart_data_draw, gx_line_chart_draw, gx_line_chart_update
gx_menu_create

Create a menu

Prototype

UINT gx_menu_create(GX_MENU *menu, GX_CONST GX_CHAR *name,
                     GX_WIDGET *parent, GX_RESOURCE_ID text_id,
                     GX_RESOURCE_ID fill_id, ULONG style,
                     USHORT menu_id, GX_CONST GX_RECTANGLE *size);

Description

This service creates a menu as specified and associates the menu with the supplied parent widget. It accepts all types of widget as child menu item. To insert a widget as a child menu item, call gx_menu_insert.

GX_MENU is derived from GX_PIXELMAP_PROMPT and supports all gx_pixelmap_prompt API services.

Parameters

- **menu**: Pointer to menu control block
- **name**: Name of the menu
- **parent**: Pointer to parent widget
- **text_id**: Resource ID of text
- **fill_id**: Resource ID of fill
- **style**: Style of the widget. Appendix D contains pre-defined general styles for all widgets as well as widget specific styles.
- **menu_id**: Application-defined ID of the menu
- **size**: Size of the menu

Return Values

- **GX_SUCCESS** (0x00): Successful menu creation
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size

Allowed From

Initialization and threads
Example

GX_MENU my_menu;

/* Create "my_menu". */
status = gx_menu_create(&my_menu, "my_menu", parent, MY_TEXT_ID,
                  MY_FILL_ID, GX_STYLE_ENABLED, MY_MENU_ID,
                  &size);

/* If status is GX_SUCCESS, the menu was successfully created. */

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_accordion_menu_position,
gx_menu_draw, gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set
gx_menu_draw

Prototype

VOID gx_menu_draw(GX_MENU *menu);

Description

This service draws the specified menu. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom menu widgets.

Parameters

menu Pointer to menu control block

Return Values

None

Allowed From

Threads

Example

/* Write a custom menu drawing function. */

VOID my_menu_draw(GX_MENU *menu)
{
    /* Call default menu draw. */
    gx_menu_draw(menu);
    /* Add your own drawing here. */
}

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_accordion_menu_position,
gx_menu_create, gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set
gx_menu_insert

Insert a new item

Prototype

UINT gx_menu_insert(GX_MENU *menu, GX_WIDGET *insert);

Description

This service inserts a new item to the menu.

Parameters

- **menu**: Pointer to menu control block
- **widget**: Pointer to the widget to insert

Return Values

- **GX_SUCCESS** (0x00): Successfully inserted new item into menu
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

Allowed From

- Initialization and threads

Example

/* Insert new item "my_widget" to the menu "my_menu". */
status = gx_menu_insert(&my_menu, &my_widget);

/* If status is GX_SUCCESS the new item "my_widget" has been inserted to the menu "my_menu". */

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_accordion_menu_position,
gx_menu_create, gx_menu_draw, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set
**gx_menu_remove**

Remove an item

**Prototype**

```c
UINT gx_menu_remove(GX_MENU *menu, GX_WIDGET *widget);
```

**Description**

This service removes an item from the menu.

**Parameters**

- **menu**: Pointer to menu control block
- **widget**: Pointer to widget to remove

**Return Values**

- **GX_SUCCESS**: (0x00) Successfully removed menu item
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Remove item "my_widget" from menu "my_menu" */
status = gx_menu_remove(&my_menu, &my_widget);

/* If status is GX_SUCCESS the item "my_widget" has been removed
from menu "my_menu". */
```

**See Also**

- `gx_accordion_menu_create`, `gx_accordion_menu_draw`,
- `gx_accordion_menu_event_process`, `gx_accordion_menu_position`,
- `gx_menu_create`, `gx_menu_draw`, `gx_menu_insert`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`
gx_menu_text_draw

Draw menu text

Prototype

VOID gx_menu_text_draw(GX_MENU *menu);

Description

This service draws the text of a menu. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom menu widgets.

Parameters

menu Pointer to menu control block

Return Values

None

Allowed From

Threads

Example

/* Write a custom menu drawing function. */

VOID my_menu_draw(GX_MENU *menu)
{
    /* Call default menu background draw. */
    gx_pixelmap_prompt_background_draw(
        (GX_PIXELMAP_PROMPT *)menu);

    /* Draw menu text. */
    gx_menu_text_draw(menu);

    /* Add your own drawing here. */
}

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_accordion_menu_position,
gx_menu_create, gx_menu_draw, gx_menu_insert, gx_menu_remove,
gx_menu_text_offset_set
gx_menu_text_offset_set

Set menu text draw offset

Prototype

UINT  gx_menu_text_offset_set(GX_MENU *menu, GX_VALUE x_offset,
                                GX_VALUE y_offset);

Description

This service sets x, y display offset for menu text.

Parameters

- **menu**: Pointer to menu control block
- **x_offset**: X coordinate of offset
- **y_offset**: Y coordinate of offset

Return Values

- **GX_SUCCESS**: (0x00) Successful set menu text draw offset
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

/* Set text draw offset of menu "my_menu" to (20, 10). */
status = gx_menu_text_offset_set(&my_menu, 20, 10);

/* If status is GX_SUCCESS the text draw offset of menu "my_menu"
has been set to (20, 10). */

See Also

gx_accordion_menu_create, gx_accordion_menu_draw,
gx_accordion_menu_event_process, gx_accordion_menu_position,
gx_menu_create, gx_menu_draw, gx_menu_insert, gx_menu_remove,
gx_menu_text_draw
gx_multi_line_text_button_create

Create multi line text button

Prototype

UINT  gx_multi_line_text_button_create(
    GX_MULTI_LINE_TEXT_BUTTON *text_button,
    GX_CONST GX_CHAR *name,
    GX_WIDGET *parent,
    GX_RESOURCE_ID text_id,
    ULONG style,
    USHORT text_button_id,
    GX_CONST GX_RECTANGLE *size);

Description

This service creates a multi-line text button widget. A multi-line text button displays the button text over 1-n lines. The maximum number of lines is defined by the constant GX_MULTI_LINE_TEXT_BUTTON_MAX_LINES, which defaults to 4. The line breaks are set by carriage return and/or carriage return + line feed pairs within the text string assigned to the multi-line text button.

GX_MULTI_LINE_TEXT_BUTTON is derived from GX_TEXT_BUTTON and supports all gx_text_button API services.

Parameters

text_button  Pointer to text button control block
name  Logical name of text button
parent  Pointer to parent widget of the button
text_id  Resource ID of text
style  Text button style. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
text_button_id  Application-defined ID of the text button
size  Size of the button

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful multi line text button create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
</tbody>
</table>
**GX_INVALID_SIZE**

(0x19)

Invalid widget control block size

**Allowed From**

Initialization and threads

**Example**

```c
/* Create multi-line text button "my_text_button" */
status = gx_multi_line_text_button_create(&my_text_button,
                                          "my text button",
                                          &my_parent_window, MY_TEXT_RESOURCE_ID,
                                          GX_STYLE_BUTTON_TOGGLE, MY_TEXT_BUTTON_ID,
                                          &size);

/* If status is GX_SUCCESS, the multi-line text button "my_text_button" was created. */
```

**See Also**

gx_text_button_create, gx_button_create, gx_multi_line_text_button_draw,
gx_multi_line_text_button_event_process, gx_multi_line_text_button_text_set,
gx_multi_line_text_button_text_id_set
guix\_multi\_line\_text\_button\_draw

**Prototype**

\texttt{VOID \textit{gx\_multi\_line\_text\_button\_draw} (GX\_MULTI\_LINE\_TEXT\_BUTTON *button);}

**Description**

This service draws the multi-line text button. This function is normally called internally by GUIX as part of a canvas refresh operation, but it also exposed to the application that might want to provide a custom drawing function and invoke the default multi-line text button drawing as custom drawing base.

**Parameters**

\texttt{button} \hspace{1cm} Pointer to text button control block

**Return Values**

None

**Allowed From**

Threads

**Example**

\begin{verbatim}
/* Draw the text button “my_text_button”. */
void MyButtonDraw(GX\_MULTI\_LINE\_TEXT\_BUTTON *button) {
  /* Do the normal drawing first */
  gx\_multi\_line\_text\_button\_draw(&my\_text\_button);
  /* Add custom drawing here. */
}
\end{verbatim}

**See Also**

\texttt{gx\_text\_button\_create, gx\_button\_create, gx\_multi\_line\_text\_button\_draw,}
\texttt{gx\_multi\_line\_text\_button\_event\_process, gx\_multi\_line\_text\_button\_text\_set,}
\texttt{gx\_multi\_line\_text\_button\_text\_id\_set}
**gx_multi_line_text_button_event_process**  
Default event handling for multi-line text button

**Prototype**

```c
UINT gx_multi_line_text_button_event_process(
    GX_MULTI_LINE_TEXT_BUTTON *button,
    GX_EVENT *event_ptr);
```

**Description**

This service is the default event handling function for the multi line text button widget. This function is made accessible to applications that want to provide custom event handling for a text button widget.

**Parameters**

- **button**: Pointer to text button control block
- **event_ptr**: Event to be processed

**Return Values**

- **GX_SUCCESS**: (0x00) Successfully handled event
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

```c
UINT MyEventHandler(GX_MULTI_LINE_TEXT_BUTTON *button,
                     GX_EVENT *event_ptr)
{
    switch(event->gx_event_type)
    {
    case GX_EVENT_SHOW:
        gx_multi_line_text_button_event_process(button, event_ptr);
        /* add custom actions here */
        break;
    default:
        gx_multi_line_text_button_event_process(button, event_ptr);
        break;
    }
    return GX_SUCCESS;
}
```

See Also

gx_text_button_create, gx_button_create, gx_multi_line_text_button_draw,
gx_multi_line_text_button_event_process, gx_multi_line_text_button_text_set,
gx_multi_line_text_button_text_id_set
**gx_multi_line_text_button_text_draw**

**Drawing support function**

**Prototype**

```c
VOID gx_multi_line_text_button_text_draw(
    GX_MULTI_LINE_TEXT_BUTTON *text_button)
```

**Description**

This support function draws the text portion of a multi-line text button. This function is called internally by `gx_multi_line_text_button_draw()`, and is provided as a separate API as a convenience for applications that define a custom multi-line text button drawing function. Applications that want to customize the button background drawing can provide their custom drawing function, and invoke the `multi_line_text_button_text_draw` service as part of their custom drawing to draw the button text over the background.

**Parameters**

- **text_button**
  
  Pointer to text button control block

**Return Values**

None

**Allowed From**

Initialization and threads

**Example**

```c
/* Define a custom drawing function */
VOID my_button_draw(GX_MULTI_LINE_TEXT_BUTTON *button)
{
    /* insert code here to draw button background */

    /* call support function to do text drawing */
    gx_multi_line_text_button_text_draw();

    /* draw child widgets */
    gx_widget_children_draw((GX_WIDGET *) button);
}
```
See Also

gx_text_button_create, gx_button_create, gx_multi_line_text_button_draw,
gx_multi_line_text_button_event_process, gx_multi_line_text_button_text_set,
gx_multi_line_text_button_text_id_set
### gx_multi_line_text_button_text_id_set

Set text resource ID to the text button

#### Prototype

```c
UINT gx_multi_line_text_button_text_id_set(
    GX_MULTI_LINE_TEXT_BUTTON *text_button,
    RESOURCE_ID string_id);
```

#### Description

This service sets the specified string resource ID to the text button. The string may contain newline characters which act to display the text on multiple lines within the button area.

#### Parameters

- **text_button**: Pointer to text button control block
- **string_id**: Resource ID of the string

#### Return Values

- **GX_SUCCESS** (0x00): Successfully set the string resource ID to the text button
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

#### Allowed From

Initialization and threads

#### Example

```c
/* Set the string ID "MY_STRING_ID" to the text button
   "my_text_button". */
status = gx_multi_line_text_button_text_id_set(
    &my_text_button, MY_STRING_ID);

/* If status is GX_SUCCESS, the string ID MY_STRING_ID was set to
   "my_text_button". */
```

#### See Also

- `gx_text_button_create`
- `gx_button_create`
- `gx_multi_line_text_button_draw`
- `gx_multi_line_text_button_event_process`
- `gx_multi_line_text_button_text_set`
- `gx_multi_line_text_button_text_id_set`
**gx_multi_line_text_button_text_set**

Assign text to the text button (deprecated)

**Prototype**

```c
UINT gx_multi_line_text_button_text_set(
    GX_MULTI_LINE_TEXT_BUTTON *text_button,
    GX_CHAR *text)
```

**Description**

This service is deprecated in favor of `gx_multi_line_text_button_text_set_ext()`. This service assigns the specified string to the text button. If the `text_button` widget was created with style `GX_STYLE_TEXT_COPY`, the widget creates a private copy of the text string assigned, and therefore the `gx_system_memory_allocate_set` API must be invoked once before this service is requested. If `GX_STYLE_TEXT_COPY` is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

**Parameters**

- `text_button` Pointer to text button control block
- `text` pointer to the NULL-terminated string

**Return Values**

- `GX_SUCCESS` (0x00) Successfully set the text to the button
- `GX_CALLER_ERROR` (0x11) Invalid caller of this function
- `GX_PTR_ERROR` (0x07) Invalid pointer
- `GX_MEMORY_ERROR` (0x30) Memory allocator is not defined
- `GX_INVALID_STRING_LENGTH` (0x34) Invalid string length

**Allowed From**

Initialization and threads
Example

static GX_CHAR text[] = “my\rstring”;

/* Set text to the text button “my_text_button”. */
status = gx_multi_line_text_button_text_set(&my_text_button, text);

/* If status is GX_SUCCESS, the text of “my_text_button” was set. */

See Also

gx_text_button_create, gx_button_create, gx_multi_line_text_button_draw, 
gx_multi_line_text_button_event_process, gx_multi_line_text_button_text_set, 
gx_multi_line_text_button_text_id_set
**gx_multi_line_text_button_text_set_ext**

Assign text to the text button

Prototype

```c
UINT  gx_multi_line_text_button_text_set_ext(
    GX_MULTI_LINE_TEXT_BUTTON *text_button,
    GX_STRING *string)
```

Description

This service assigns the specified string to the text button. If the text button widget was created with style GX_STYLE_TEXT_COPY, the widget creates a private copy of the text string assigned, and therefore the `gx_system_memory_allocate_set` API must be invoked once before this service is requested. If GX_STYLE_TEXT_COPY is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_button</td>
<td>Pointer to text button control block</td>
</tr>
<tr>
<td>string</td>
<td>Pointer to GX_STRING variable</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successfully set the text to the button</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_MEMORY_ERROR</td>
<td>(0x30)</td>
<td>Memory allocator is not defined</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>(0x34)</td>
<td>Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example
static GX_CHAR text[] = "my\rstring";
GX_STRING string;

string.gx_string_ptr = text;
string.gx_string_length = strlen(text);

/* Set text to the text button "my_text_button". */
status = gx_multi_line_text_button_text_set_ext(&my_text_button,
                                                string);

/* If status is GX_SUCCESS, the text of "my_text_button" was set. */

See Also

gx_multi_line_text_button_draw, gx_multi_line_text_button_event_process,
gx_multi_line_text_button_text_set, gx_multi_line_text_button_text_id_set
**gx_multi_line_text_input_backspace**

Delete a character before multi line text input cursor position

**Prototype**

```c
UINT gx_multi_line_text_input_backspace(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service deletes the character before multi line text input cursor position. This service is called internally when a backspace key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**: Multi-line text input widget control block

**Return Values**

- **GX_SUCCESS**: (0x00) Successful multi-line text input backspace
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x23) Widget not valid
- **GX_FAILURE**: (0x10) Invalid font

**Allowed From**

Initialization and threads
Example

/* Delete a character before the cursor of "my_text_input". */
status = gx_multi_line_text_input_backspace(&my_text_input);

/* If status is GX_SUCCESS the character before the cursor has been deleted. */

See Also

gx_multi_line_text_input_buffer_clear, gx_multi_line_text_input_buffer_get,
gx_multi_line_text_input_char_insert, gx_multi_line_text_input_create,
gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_input_buffer_clear

Deletes all characters from the text input buffer

Prototype

UINT gx_multi_line_text_input_buffer_clear(GX_MULTI_LINE_TEXT_INPUT *text_input);

Description

This service deletes all characters from the text input buffer.

Parameters

text_input Multi-line text input widget control block

Return Values

GX_SUCCESS (0x00) Successful multi-line text input buffer clear
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

/* clear input buffer of ”my_text_input”. */
status = gx_multi_line_text_input_buffer_clear(&my_text_input);

/* If status is GX_SUCCESS the text input widget has emptied its input buffer. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_get,
gx_multi_line_text_input_char_insert, gx_multi_line_text_input_create,
gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_buffer_get**

Retrieves buffer information of text input widget

**Prototype**

```c
UINT gx_multi_line_text_input_buffer_get(
    GX_MULTI_LINE_TEXT_INPUT *text_input, GX_CHAR **buffer_address, UINT *content_size, UINT *buffer_size);
```

**Description**

This service retrieves buffer information of a multi-line text input widget.

**Parameters**

- **text_input**: Multi-line text input widget control block
- **buffer_address**: The address of the input buffer
- **content_size**: The byte count of the input data
- **buffer_size**: The size of the input buffer

**Return Values**

- **GX_SUCCESS** (0x00): Successful multi-line text get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

- Initialization and threads

**Example**

```c
GX_CHAR *buffer_address;
UINT context_size;
UINT buffer_size;

/* Retrieves buffer information of ”my_text_input” widget. */
status = gx_multi_line_text_input_buffer_get(&my_text_input,
    &buffer_address, &string_size,
    &buffer_size);

/* If status is GX_SUCCESS the value of buffer_address, string_size and buffer_size has been retrieved. */
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_char_insert, gx_multi_line_text_input_create,
gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_char_insert**

Insert a character string at current multi line text input cursor position
(deprecated)

**Prototype**

```c
UINT gx_multi_line_text_input_char_insert(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_UBYTE *insert_str,
    UINT insert_size);
```

**Description**

This API is deprecated and replaced by
gx_multi_line_text_input_char_insert_ext().

This service inserts a character string into the multi line text input
string buffer at the current cursor position. This service is called
internally when specific key down event is received, but can also be
invoked by the application.

**Parameters**

- **text_input**: Multi-line text input widget control block
- **insert_str**: UTF-8 format character string to be inserted
- **insert_size**: Byte count to be inserted

**Return Values**

- **GX_SUCCESS** (0x00): Successfully inserted the character string
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_VALUE** (0x22): Invalid string size
- **GX_FAILURE** (0x10): Invalid font or out of buffer size

**Allowed From**

Initialization and threads
Example

/* Insert characters at current cursor position. */
GX_CHAR insert_text[10] = "insert";
status = gx_multi_line_text_input_char_insert(&my_text_input,
                                          insert_text,
                                          GX_STRLEN(insert_text));

/* If status is GX_SUCCESS the multi line text input widget has
   successfully insert the string. */

See Also

gx_multi_line_text_input_char_insert_ext
gx_multi_line_text_input_char_insert_ext

Insert a character string at current multi line text input cursor position (deprecated)

Prototype

UINT  gx_multi_line_text_input_char_insert_ext(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_CONST GX_STRING *string);

Description

This service inserts a character string into the multi line text input string buffer at the current cursor position. This service is called internally when specific key down events are received, but can also be invoked by the application.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_input</td>
<td>Multi-line text input widget control block</td>
</tr>
<tr>
<td>string</td>
<td>UTF-8 encoded character string to be inserted</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully inserted the character string</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22) Invalid string size</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x10) Invalid font or out of buffer size</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>(0x34) Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Insert characters at current cursor position. */
GX_CHAR insert_text[10] = “insert”;
GX_STRING string;

string.gx_string_ptr = insert_text;
string.gx_string_length = strlen(insert_text);

status = gx_multi_line_text_input_char_insert_ext(&my_text_input, &string);

/* If status is GX_SUCCESS the multi line text input widget has successfully inserted the string. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_create,
gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set

gx_multi_line_text_input_create

Create multi-line text input

Prototype

UINT gx_multi_line_text_input_create(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_CONST GX_CHAR *name, GX_WINDOW *parent,
    GX_CHAR *input_buffer, UINT buffer_size,
    ULONG style, USHORT text_input_id,
    GX_CONST GX_RECTANGLE *size);

Description

This service creates a multi-line text input widget.
GX_MULTI_LINE_TEXT_INPUT is derived from GX_MULTI_LINE_TEXT_VIEW and supports all gx_multi_line_text_view services.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_input</td>
<td>Multi-line text input widget control block</td>
</tr>
<tr>
<td>name</td>
<td>Name of text input widget</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>input_buffer</td>
<td>Pointer to text input buffer</td>
</tr>
<tr>
<td>buffer_size</td>
<td>Size of text input buffer in bytes</td>
</tr>
<tr>
<td>style</td>
<td>Style of text input widget. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.</td>
</tr>
<tr>
<td>text_input_id</td>
<td>Application-defined ID of text input</td>
</tr>
<tr>
<td>size</td>
<td>Dimensions of text input widget</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful multi-line text input create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Parent widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid widget control block size</td>
</tr>
</tbody>
</table>
Allowed From

Initialization and threads

Example

GX_MULTI_LINE_TEXT_INPUT my_text_input;
GX_CHAR my_buffer[100];
GX_RECTANGLE size;

/* Define widget size. */
gx_utility_rectangle_define(&size, 10, 10, 100, 200);

/* Create multi-line text input widget "my_text_input". */
status = gx_multi_line_text_input_create(&my_text_input,
                           "my_text_input", &my_parent,
                           my_buffer, 100, GX_STYLE_BORDER_RAISED,
                           MY_TEXT_INPUT_ID, &size);

/* If status is GX_SUCCESS, the text input "my_text_input" has been created. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_input_cursor_pos_get

Retrieve multi line text input cursor position

Prototype

UINT gx_multi_line_text_input_cursor_pos_get(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_POINT cursor_pos);

Description

This service retrieves the multi-line text input cursor position.

Parameters

text_input Multi-line text input widget control block
cursor_pos Retrieved cursor position

Return Values

GX_SUCCESS (0x00) Successfully retrieved cursor position
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads
Example

/* Retrieve the cursor position of “my_text_input”. */
GX_POINT cursor_pos;
status = gx_multi_line_text_input_cursor_pos_get(&my_text_input,
&cursor_pos);

/* If status is GX_SUCCESS the cursor position has been retrieved. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_line_space_set,
gx_multi_line_text_view_scroll_info_get, gx_multi_line_text_view_text_color_set,
gx_multi_line_text_view_text_id_set, gx_multi_line_text_view_text_set,
gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_delete**

Delete the character at the multi line text input cursor position

**Prototype**

```c
UINT gx_multi_line_text_input_delete(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service deletes the character after the multi line text input cursor position. This service is called internally when a delete key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**
  
  Multi-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00)
  
  Successfully delete a character after the cursor

- **GX_CALLER_ERROR** (0x11)
  
  Invalid caller of this function

- **GX_PTR_ERROR** (0x07)
  
  Invalid pointer

- **GX_INVALID_WIDGET** (0x12)
  
  Widget not valid

- **GX_FAILURE** (0x10)
  
  Invalid font

**Allowed From**

- Initialization and threads
Example

/* Delete the character after the cursor of "my_text_input". */
status = gx_multi_line_text_input_delete(&my_text_input);

/* If status is GX_SUCCESS the character after the cursor has been
deleted. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_input_down_arrow

Move the multi line text input cursor to the next line

Prototype

UINT  gx_multi_line_text_input_down_arrow(
    GX_MULTI_LINE_TEXT_INPUT *text_input);

Description

This service positions the multi line text input widget cursor to the next line. This service is called internally when a down arrow key down event is received, but can also be invoked by the application.

Parameters

text_input  Multi-line text input widget control block

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully moved text input cursor to the next line</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET (0x12)</td>
<td>Widget not valid</td>
</tr>
<tr>
<td>GX_FAILURE (0x10)</td>
<td>Invalid font or line height</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Move input cursor to the next line. */
status = gx_multi_line_text_input_down_arrow(&my_text_input);

/* If status is GX_SUCCESS, text input cursor has been moved to the next line. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_end,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
**gx_multi_line_text_input_end**

Move the multi line text input cursor to the end of the current line

**Prototype**

```c
UINT gx_multi_line_text_input_end(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service positions the multi line text input widget cursor to the end of the current string line. This service is called internally when an end key down event is received, but can also be invoked by the application.

**Parameters**

- `text_input` Multi-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully moved text input cursor to end of the current line
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

- Initialization and threads
Example

/* Move input cursor to the end of current line. */
status = gx_multi_line_text_input_end(&my_text_input);

/* If status is GX_SUCCESS, the multi line text input cursor has been moved to the end of the current line. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_event_process**

Default event handling for multi-line text input

Prototype

```c
UINT gx_multi_line_text_input_event_process(
    GX_MULTI_LINE_TEXT_INPUT *input,
    GX_EVENT *event_ptr);
```

Description

This service is the default event handling function for the multi line text input widget. This function is made accessible to applications that want to provide custom event handling for a multi line text input widget.

Parameters

- **button**: Pointer to multi line text input control block
- **event_ptr**: Event to be processed

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GX_SUCCESS</strong></td>
<td>(0x00)</td>
<td>Successfully handled event</td>
</tr>
<tr>
<td><strong>GX_CALLER_ERROR</strong></td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td><strong>GX_PTR_ERROR</strong></td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td><strong>GX_INVALID_WIDGET</strong></td>
<td>(0x12)</td>
<td>Parent widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

- Initialization and threads

Example

```c
UINT MyEventHandler(GX_MULTI_LINE_TEXT_INPUT *input,
    GX_EVENT *event_ptr)
{
    switch(event->gx_event_type)
    {
    case xyz:
        /* insert custom event handling here */
        break;

    default:
        /* pass all other events to the generic multi line text input event processing */
        gx_multi_line_text_input_event_process(input, event_ptr);
        break;
    }
    return GX_SUCCESS;
}
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_fill_color_set,
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_fill_color_set**

Set multi line text input background color

**Prototype**

```c
UINT gx_multi_line_text_input_fill_color_set(  
    GX_MULTI_LINE_TEXT_INPUT *text_input,  
    GX_RESOURCE_ID normal_fill_color_id,  
    GX_RESOURCE_ID selected_fill_color_id,  
    GX_RESOURCE_ID disabled_fill_color_id,  
    GXRESOURCE_ID readonly_fill_color_id);
```

**Description**

This service assigns fill colors for the multi-line text input widget.

**Parameters**

- **text_input**
  Multi-line text input widget control block
- **normal_fill_color_id**
  Resource ID of the normal fill color that used in normal state
- **selected_fill_color_id**
  Resource ID of the selected fill color that used when the widget gain focus
- **disabled_fill_color_id**
  Resource ID of the disabled fill color that used when GX_STYLE_ENABLED is not active
- **readonly_fill_color_id**
  Resource ID of the read only fill color that used when both GX_STYLE_ENABLED and GX_STYLE_INPUT_READONLY are active.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set colors for the multi-line text input
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Set fill colors for the multi-line text input widget "my_text_input". */
status = gx_multi_line_text_input_fill_color_set(&my_text_input, 
    GX_COLOR_ID_NORMAL_FILL, 
    GX_COLOR_ID_SELECTED_FILL, 
    GX_COLOR_ID_DISABLED_FILL, 
    GX_COLOR_ID_READONLY_FILL);

/* If status is GX_SUCCESS, the fill color of "my_text_input" has been successfully set. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear, 
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert, 
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get, 
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow, 
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process, 
gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow, 
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add, 
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, 
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select, 
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow, 
gx_multi_line_text_view_create, gx_multi_line_text_view_draw, 
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, 
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get, 
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set, 
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_home**

Move the text input cursor to the start of the current line

**Prototype**

```c
UINT gx_multi_line_text_input_home(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service moves the text input cursor position to the start of the current line. This service is called internally when a home key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**
  Multi-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully moved cursor to start of the current line
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Move cursor to the start of the current line. */
status = gx_multi_line_text_input_home(&my_text_input);

/* If status is GX_SUCCESS the cursor has been moved to the start of the current line. */
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_left_arrow,
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_left_arrow**

Move multi line text input cursor one character to the left

**Prototype**

```c
UINT  gx_multi_line_text_input_left_arrow(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service moves the multi line text input cursor one character to the left. This service is called internally when a left key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**
  Single-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully moved cursor to the left
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid
- **GX_FAILURE** (0x10) Invalid font

**Allowed From**

- Initialization and threads

**Example**

```c
/* Move the cursor one character to the left. */
status = gx_multi_line_text_input_left_arrow(&my_text_input);

/* If status is GX_SUCCESS the multi line text input cursor has been moved one character to the left. */
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear, 
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert, 
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get, 
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow, 
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process, 
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home, 
gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add, 
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, 
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select, 
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow, 
gx_multi_line_text_view_create, gx_multi_line_text_view_draw, 
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, 
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get, 
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set, 
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
### gx_multi_line_text_input_right_arrow

**Move multi line text input cursor one character to the right**

**Prototype**

```c
UINT gx_multi_line_text_input_right_arrow(
    GX_MULTI_LINE_TEXT_INPUT *text_input);
```

**Description**

This service moves the multi line text input cursor one character to the right. This service is called internally when a right key down event is received, but can also be invoked by the application.

**Parameters**

- `text_input`: Multi-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully moved cursor to the right
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Move cursor one character to the right. */
status = gx_multi_line_text_input_right_arrow(&my_text_input);

/* If status is GX_SUCCESS the text input cursor has been moved one character to the right. */
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_style_add,
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_style_add**

Add multi line text input styles

**Prototype**

```c
UINT gx_multi_line_text_input_style_add(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    ULONG style);
```

**Description**

This service adds styles to a multi-line text input widget.

**Parameters**

- `text_input`: Multi-line text input widget control block
- `style`: Styles to add. Appendix D contains predefined general styles for all widgets

**Return Values**

- **GX_SUCCESS** (0x00): Successful multi-line text input style add
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Add style GX_STYTLE_CURSOR_ALWAYS to multi-line text input widget "my_text_input". */
status = gx_multi_line_text_input_style_add(&my_text_input, GX_STYLE_CURSOR_ALWAYS);

/* If status is GX_SUCCESS the style GX_STYLE_CURSOR_ALWAYS has been successfully added. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear, 
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert, 
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get, 
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow, 
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process, 
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home, 
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow, 
gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, 
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select, 
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow, 
gx_multi_line_text_view_create, gx_multi_line_text_view_draw, 
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, 
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get, 
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set, 
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_style_remove**

Remove styles

**Prototype**

```c
UINT gx_multi_line_text_input_remove(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    ULONG style);
```

**Description**

This service removes the specified styles from the multi-line text input widget.

**Parameters**

- **text_input**
  - Multi-line text input widget control block
- **style**
  - Styles to remove. **Appendix D** contains pre-defined general styles for all widgets

**Return Values**

- **GX_SUCCESS** (0x00) Successful multi-line text input create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Remove style GX_STYLE_CURSOR_ALWAYS from text input widget "my_text_input". */
status = gx_multi_line_text_input_style_remove(&my_text_input,
                                        GX_STYLE_CURSOR_ALWAYS);

/* If status is GX_SUCCESS, the style GX_STYLE_CURSOR_ALWAYS has been successfully removed. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_set,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_style_set**

Set multi line text input styles

**Prototype**

```c
UINT gx_multi_line_text_input_style_set(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    ULONG style);
```

**Description**

This service sets styles for a multi-line text input widget.

**Parameters**

- **text_input**
  Multi-line text input widget control block

- **style**
  Styles to set. **Appendix D** contains pre-defined general styles for all widgets

**Return Values**

- **GX_SUCCESS** (0x00) Successful multi-line text input style set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Set style GX_STYLE_CURSOR_ALWAYS for text input widget "my_text_input". */
status = gx_multi_line_text_input_style_set(&my_text_input,
    GX_STYLE_CURSOR_ALWAYS);

/* If status is GX_SUCCESS the text input style has been set */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_text_color_set**

Set multi line text input text color

**Prototype**

```c
UINT gx_multi_line_text_input_text_color_set(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_RESOURCE_ID normal_text_color_id,
    GX_RESOURCE_ID selected_text_color_id,
    GX_RESOURCE_ID disabled_text_color_id,
    GX_RESOURCE_ID readonly_text_color_id);
```

**Description**

This service assigns text colors for the multi-line text input widget.

**Parameters**

- `text_input` Multi-line text input widget control block
- `normal_fill_color_id` Resource ID of the normal text color that used in normal state
- `selected_text_color_id` Resource ID of the selected text color that used when the widget gain focus
- `disabled_text_color_id` Resource ID of the disabled text color that used when `GX_STYLE_ENABLED` is not active
- `readonly_text_color_id` Resource ID of the read only text color that used when both `GX_STYLE_ENABLED` and `GX_STYLE_TEXT_INPUT_READONLY` are active

**Return Values**

- `GX_SUCCESS` (0x00) Successfully set colors for the multi-line text input
- `GX_CALLER_ERROR` (0x11) Invalid caller of this function
- `GX_PTR_ERROR` (0x07) Invalid pointer

**Allowed From**

Initialization and threads
**Example**

```c
/* Set text colors for the multi-line text input widget
"my_text_input". */
status = gx_multi_line_text_input_text_color_set(&my_text_input,
    GX_COLOR_ID_NORMAL_TEXT,
    GX_COLOR_ID_SELECTED_TEXT,
    GX_COLOR_ID_DISABLED_TEXT,
    GX_COLOR_ID_READONLY_TEXT);

/* If status is GX_SUCCESS, the fill colors of "my_text_view" has
been successfully set. */
```

**See Also**

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_select,
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_input_text_select**

Select text

**Prototype**

```c
UINT  gx_multi_line_text_input_text_select(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    UINT  start_index, UINT  end_index);
```

**Description**

This service selects multi line text input text with specified start mark and end mark index and highlights the selected text with the selected fill and text colors.

**Parameters**

- `text_input` Pointer to multi line text input control block
- `start_index` Index of the first selected character
- `end_index` Index of the last selected character

**Return Values**

- **GX_SUCCESS** (0x00) Successful multi line text input text selection
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid
- **GX_INVALID_VALUE** (0x22) Index value not valid

**Allowed From**

Initialization and threads
Example

```c
/* Select text between index [0, 9]. */
status = gx_multi_line_text_input_text_select(&my_text_input, 0, 9);

/* If status is GX_SUCCESS, the text between index [0, 9] "my_text_input" was selected. */
```

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear, 
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert, 
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get, 
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow, 
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process, 
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home, 
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow, 
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove, 
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set, 
gx_multi_line_text_input_text_set, gx_multi_line_text_input_up_arrow, 
gx_multi_line_text_view_create, gx_multi_line_text_view_draw, 
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, 
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get, 
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set, 
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_input_text_set
Assign text to the text input (deprecated)

Prototype

UINT gx_multi_line_text_input_text_set(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_CHAR *text)

Description

This API is deprecated and replace by
gx_multi_line_text_input_text_set_ext().

This service assigns the specified string to the multi line text input. If
the multi_line_text_input widget’s input buffer size is smaller than
string length, the string will be truncated.

Parameters

- text_input: Pointer to multi line text input control
  block
- text: pointer to the NULL-terminated string

Return Values

- GX_SUCCESS: (0x00) Successfully set the text
to the multi line text input
- GX_CALLER_ERROR: (0x11) Invalid caller of this function
- GX_PTR_ERROR: (0x07) Invalid pointer
- GX_INVALID_STRING_LENGTH: (0x34) Invalid string length

Allowed From

- Initialization and threads

Example

/* Set the string "my string" to the text button "my_text_input". */
status = gx_multi_line_text_input_text_set(&my_text_input,
                                           "my\rstring");

/* If status is GX_SUCCESS, the content of "my_text_input" has been reset. */
See Also

gx_multi_line_text_input_text_set_ext
**gx_multi_line_text_input_text_set_ext**
Assign text to the text input

**Prototype**

```c
UINT  gx_multi_line_text_input_text_set(
    GX_MULTI_LINE_TEXT_INPUT *text_input,
    GX_CONST GX_STRING *string)
```

**Description**

This service assigns the specified string to the multi line text input. If the multi_line_text_input widget’s input buffer size is smaller than string length, the string will be truncated.

**Parameters**

- **text_input**: Pointer to multi line text input control block
- **string**: Pointer to GX_STRING to assign

**Return Values**

- **GX_SUCCESS** (0x00): Successfully set the text to the multi line text input
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_STRING_LENGTH** (0x34): Invalid string length

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the string "my string" to the text button "my_text_input".
 */
GX_STRING string;
string.gx_string_ptr = "my\rstring";
string.gx_string_length = strlen(string.gx_string_ptr);
status = gx_multi_line_text_input_text_set_ext(&my_text_input,
    &string);

/* If status is GX_SUCCESS, the content of "my_text_input" has been reset. */
```
See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_up_arrow,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_view_create**

Create multi-line text view

**Prototype**

```c
UINT  gx_multi_line_text_view_create(GX_MULTI_LINE_TEXT_VIEW *text_view,
                                      GX_CONST GX_CHAR *name,
                                      GX_WINDOW *parent,
                                      GX_RESOURCE_ID text_id,
                                      ULONG style,
                                      USHORT text_view_id,
                                      GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a GX_MULTI_LINE_TEXT_VIEW widget. This widget type is derived from GX_WINDOW, and therefore all gx_window API services may also be utilized with this widget type.

**Parameters**

- **text_view**: Multi-line text view widget control block
- **name**: Name of the text view widget
- **parent**: Pointer to parent widget
- **text_id**: Resource ID of the text string
- **style**: Style of text view widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
- **text_view_id**: Application-defined ID of text view
- **size**: Dimensions of text view widget

**Return Values**

- **GX_SUCCESS** (0x00): Successfully created multi-line text view widget
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY.Created** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size

**Allowed From**

Initialization and threads
Example

```c
/* Create multi-line text view widget ”my_text_view”.
status = gx_multi_line_text_view_create(&my_text_view,
    ”my_text_view”, &my_parent,
    TEXT_STRING_ID, GX_STYLE_BORDER_RAISED,
    MY_TEXT_VIEW_ID, &size);

/* If status is GX_SUCCESS the text view ”my_text_view” has been
   successfully created. */
```

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_draw,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_view_draw**

Draw a multi line text view widget

**Prototype**

```c
VOID gx_multi_line_text_view_draw (GX_MULTI_LINE_TEXT_VIEW *text_view);
```

**Description**

This service draws a multi line text view widget. This service is normally called internally during canvas refresh, but can also be called from custom multi line text view drawing functions.

**Parameters**

- `text_view` Multi-line text view widget control block

**Return Values**

None

**Allowed From**

Initialization and threads

**Example**

```c
/* Write a custom multi line text view drawing function. */

VOID my_multi_line_text_view_draw(GX_MULTI_LINE_TEXT_VIEW *view)
{
    /* Call default multi line text view draw. */
    gx_multi_line_text_view_draw(view);
    /* Add your own drawing here. */
}
```

**See Also**

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear, gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert, gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get, gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow, gx_multi_line_text_input_end, gx_multi_line_text_input_event_process, gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home, gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow, gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set, gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_view_event_process

Process multi-line text view event

Prototype

UINT gx_multi_line_text_view_event_process(
    GX_MULTI_LINE_TEXT_VIEW *text_view,
    GX_EVENT  *event);

Description

This service processes an event for a multi-line text view widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_view</td>
<td>Multi-line text view widget control block</td>
</tr>
<tr>
<td>event</td>
<td>Pointer to event to process</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful multi-line text view event process</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Threads
Example

```c
/* Write a custom event handler. */
UINT my_event_handler(GX_MULTI_LINE_TEXT_VIEW *view, GX_EVENT *event_ptr)
{
    switch(event->gx_event_type)
    {
    case GX_EVENT_SHOW:
        gx_multi_line_text_view_event_process(view, event_ptr);
        /* Add custom actions here. */
        break;
    default:
        gx_multi_line_text_view_event_process (view, event_ptr);
        break;
    }
    return GX_SUCCESS;
}
```

See Also

`gx_multi_line_text_input_backspace`, `gx_multi_line_text_input_buffer_clear`,
`gx_multi_line_text_input_buffer_get`, `gx_multi_line_text_input_char_insert`,
`gx_multi_line_text_input_create`, `gx_multi_line_text_input_cursor_pos_get`,
`gx_multi_line_text_input_delete`, `gx_multi_line_text_input_down_arrow`,
`gx_multi_line_text_input_end`, `gx_multi_line_text_input_event_process`,
`gx_multi_line_text_input_fill_color_set`, `gx_multi_line_text_input_home`,
`gx_multi_line_text_input_left_arrow`, `gx_multi_line_text_input_right_arrow`,
`gx_multi_line_text_input_style_add`, `gx_multi_line_text_input_style_remove`,
`gx_multi_line_text_input_style_set`, `gx_multi_line_text_input_text_color_set`,
`gx_multi_line_text_input_text_select`, `gx_multi_line_text_input_text_set`,
`gx_multi_line_text_view_draw`, `gx_multi_line_text_view_font_set`,
`gx_multi_line_text_view_line_space_set`, `gx_multi_line_text_view_scroll_info_get`,
`gx_multi_line_text_view_text_color_set`, `gx_multi_line_text_view_text_id_set`,
`gx_multi_line_text_view_text_set`, `gx_multi_line_text_view_whitespace_set`
__gx_multi_line_text_view_font_set__

Set font used in multi-line text view

**Prototype**

```c
UINT gx_multi_line_text_view_text_id_set(
    GX_MULTI_LINE_TEXT_VIEW *text_view,
    GX_RESOURCE_ID font_id);
```

**Description**

This service sets the font of a multi-line text view widget.

**Parameters**

- `text_view`: Multi-line text view widget control block
- `font_id`: Resource ID for the font

**Return Values**

- **GX_SUCCESS** (0x00): Successfully set font for the multi-line text view
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Set font ID FONT_ID to the multi-line text view widget
   “my_text_view”. */
status = gx_multi_line_text_view_font_set(&my_text_view, FONT_ID);

/* If status is GX_SUCCESS, the text view “my_text_view” will use
the specified font to display its text. */
```

**See Also**

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_line_space_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_view_line_space_set

Set multi-line text view line space

Prototype

UINT gx_multi_line_text_view_line_space_set(
    GX_MULTI_LINE_TEXT_VIEW *text_view,
    GX_BYTE line_space);

Description

This service sets the spacing between lines of text for the multi-line text view widget.

Parameters

<table>
<thead>
<tr>
<th>view</th>
<th>Multi-line text view widget control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>line_space</td>
<td>Value to set</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>GX_SUCCESS</th>
<th>(0x00) Successfully set line space value for the multi-line text view</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Set line space of "my_text_view" to 2. */
status = gx_multi_line_text_view_line_space_set(&my_text_view, 2);

/* If status is GX_SUCCESS, the line space of "my_text_view" has been successfully set to 2. */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_view_create, gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_scroll_info_get,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_view_scroll_info_get**

Get multi-line text view scroll info

**Prototype**

```c
UINT gx_multi_line_text_view_scroll_info_get(
    GX_MULTI_LINE_TEXT_VIEW *text_view, ULONG style,
    GX_SCROLL_INFO *info);
```

**Description**

This service gets the multi-line text view scroll information.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_view</td>
<td>Multi-line text view widget control block</td>
</tr>
<tr>
<td>Style</td>
<td>GX_SCROLLBAR_HORIZONTAL or GX_SCROLLBAR_VERTICAL</td>
</tr>
<tr>
<td>Info</td>
<td>Pointer to destination for scroll info. Appendix I contains definition to GX_SCROLL_INFO structure.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully retrieved text view scroll info</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x10) Widget is not visible or text view font id is not valid</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads
Example

GX_SCROLL_INFO scroll_info;

/* Get scroll information for multi-line text view "my_text_view". */
status = gx_multi_line_text_view_scroll_info_get(&my_text_view,
&scroll_info);

/* If status is GX_SUCCESS the "scroll_info" contains the scroll
information for multi-line text view "my_text_view". */

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_line_space_set,
gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_view_text_color_set**

Set the text color for the multi line text view

**Prototype**

```c
UINT gx_multi_line_text_view_text_color_set(  
    GX_MULTI_LINE_TEXT_VIEW *text_view,  
    GX_RESOURCE_ID normal_text_color_id,  
    GX_RESOURCE_ID selected_text_color_id,  
    GX_RESOURCe_ID disabled_text_color_id);
```

**Description**

This service assigns text color to the multi-line text view widget.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_view</td>
<td>Multi-line text view widget control block</td>
</tr>
<tr>
<td>normal_text_color_id</td>
<td>Resource ID of the normal text color that used in normal state</td>
</tr>
<tr>
<td>selected_text_color_id</td>
<td>Resource ID of the selected text color that used when the widget gain focus</td>
</tr>
<tr>
<td>disabled_text_color_id</td>
<td>Resource ID of the disabled text color that used GX_STYLE_ENABLED is not active</td>
</tr>
</tbody>
</table>

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set colors for the multi-line text view
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

- Initialization and threads
Example

/* Set text colors for the multi-line text view widget "my_text_view". */
status = gx_multi_line_text_view_text_color_set(&my_text_view,
    GX_COLOR_ID_NORMAL_TEXT,
    GX_COLOR_ID_SELECTED_TEXT,
    GX_COLOR_ID_DISABLED_TEXT);

/* If status is GX_SUCCESS the text color of "my_text_view" has been successfully set. */

See Also
gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_line_space_set,
gx_multi_line_text_view_scroll_info_get, gx_multi_line_text_view_text_id_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_view_text_id_set

Set system text string in multi line text view

Prototype

UINT gx_multi_line_text_view_text_id_set(
    GX_MULTI_LINE_TEXT_VIEW *text_view,
    GXRESOURCE_ID text_id);

Description

This service sets the resource ID of a string to the multi-line text view widget.

Parameters

text_view Multi-line text view widget control block
text_id Resource ID for the text string

Return Values

GX_SUCCESS (0x00) Successfully set string id for the multi-line text view
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_RESOURCE_ID (0x33) Invalid resource ID

Allowed From

Initialization and threads
Example

```c
/* Set string ID STRING_ID to the multi-line text view widget
   "my_text_view". */
status = gx_multi_line_text_view_text_id_set(&my_text_view,
                                          STRING_ID);

/* If status is GX_SUCCESS the text id of "my_text_view" has been
   successfully set. */
```

See Also

gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_line_space_set,
gx_multi_line_text_view_scroll_info_get, gx_multi_line_text_view_text_color_set,
gx_multi_line_text_view_text_set, gx_multi_line_text_view_whitespace_set
gx_multi_line_text_view_text_set

Set user-defined string in multi line text view

Prototype

UINT gx_multi_line_text_view_text_set(
    GX_MULTI_LINE_TEXT_VIEW *text_view,
    GX_CONST GX_CHAR *text);

Description

This service assigns a text string to the multi-line text view widget. If
the text_view widget was created with style
GX_STYLE_TEXT_COPY, the widget creates a private copy of the
text string assigned, and therefore the
gx_system_memory_allocate_set API must be invoked once before
this service is requested. If GX_STYLE_TEXT_COPY is not active,
the widget does not make a private copy of the incoming string, and
therefore the assigned string must be statically or globally allocated,
i.e. it may not be an automatic or temporary variable.

Parameters

text_view  Multi-line text view widget control block
text  NULL-terminated text string

Return Values

GX_SUCCESS  (0x00)  Successfully set string for
the multi-line text view

GX_SYSTEM_MEMORY_ERROR  (0x30)  Memory allocator is not
defined or memory
allocation failed

GX_CALLER_ERROR  (0x11)  Invalid caller of this function
GX_PTR_ERROR  (0x07)  Invalid pointer

Allowed From

Initialization and threads
Example

/* Set string “my string” to the multi-line text view widget "my_text_view". */
status = gx_multi_line_text_view_text_set(&my_text_view,
                                           "my string");

/* If status is GX_SUCCESS the text of "my_text_view" has been successfully set. */

See Also
gx_multi_line_text_input_backspace, gx_multi_line_text_input_buffer_clear,
gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_char_insert,
gx_multi_line_text_input_create, gx_multi_line_text_input_cursor_pos_get,
gx_multi_line_text_input_delete, gx_multi_line_text_input_down_arrow,
gx_multi_line_text_input_end, gx_multi_line_text_input_event_process,
gx_multi_line_text_input_fill_color_set, gx_multi_line_text_input_home,
gx_multi_line_text_input_left_arrow, gx_multi_line_text_input_right_arrow,
gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove,
gx_multi_line_text_input_style_set, gx_multi_line_text_input_text_color_set,
gx_multi_line_text_input_text_select, gx_multi_line_text_input_text_set,
gx_multi_line_text_input_up_arrow, gx_multi_line_text_view_create,
gx_multi_line_text_view_draw, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_font_set, gx_multi_line_text_view_line_space_set,
gx_multi_line_text_view_scroll_info_get, gx_multi_line_text_view_text_color_set,
gx_multi_line_text_view_text_id_set, gx_multi_line_text_view_whitespace_set
**gx_multi_line_text_view_whitespace_set**

Set multi-line text view whitespace

**Prototype**

```
UINT gx_multi_line_text_view_whitespace_set(
    GX_MULTI_LINE_TEXT_VIEW *text_view, GX_UBYTE whitespace);
```

**Description**

This service sets spacing between widget outlines and client area for a multi-line text view widget.

**Parameters**

- `text_view`: Multi-line text view widget control block
- `whitespace`: Width of margin between text_view widget and the displayed text, in pixels.

**Return Values**

- `GX_SUCCESS` (0x00): Successfully set whitespace for the multi-line text view
- `GX_CALLER_ERROR` (0x11): Invalid caller of this function
- `GX_PTR_ERROR` (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
/* Set whitespace of "my_text_view" to 2. */
status = gx_multi_line_text_view_whitespace_set(&my_text_view, 2);

/* If status is GX_SUCCESS the whitespace of "my_text_view" has been successfully set to 2. */
```

**See Also**

- `gx_multi_line_text_input_backspace`
- `gx_multi_line_text_input_buffer_clear`
- `gx_multi_line_text_input_buffer_get`
- `gx_multi_line_text_input_char_insert`
- `gx_multi_line_text_input_create`
- `gx_multi_line_text_input_cursor_pos_get`
- `gx_multi_line_text_input_delete`
- `gx_multi_line_text_input_down_arrow`
- `gx_multi_line_text_input_end`
- `gx_multi_line_text_input_event_process`
- `gx_multi_line_text_input_fill_color_set`
- `gx_multi_line_text_input_home`
- `gx_multi_line_text_input_left_arrow`
- `gx_multi_line_text_input_right_arrow`
- `gx_multi_line_text_input_style_add`
- `gx_multi_line_text_input_style_remove`
**gx_numeric_pixelmap_prompt_create**

Create numeric pixelmap prompt

**Prototype**

```c
UINT gx_numeric_pixelmap_prompt_create(
    GX_NUMERIC_PIXELMAP_PROMPT *prompt,
    GX_CONST GX_CHAR name, GX_WIDGET *parent,
    GXRESOURCE_ID text_id, GXRESOURCE_ID fill_id,
    ULONG style, USHORT pixelmap_prompt_id,
    GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a numeric pixelmap prompt widget. A numeric_pixelmap_prompt is just a pixelmap_prompt that keeps its own buffer and provides a gx_numeric_pixelmap_prompt_value_set(INT) API, the buffer size is defined by the constant GX_NUMERIC_PROMPT_BUFFER_SIZE, which defaults to 16.

GX_NUMERIC_PIXELMAP_PROMPT is derived from GX_PIXELMAP_PROMPT and supports all gx_pixelmap_prompt API services.

**Parameters**

- `prompt`  
  Numeric pixelmap prompt control block
- `name`  
  Name of prompt
- `parent`  
  Parent widget control block
- `text_id`  
  Resource string id
- `fill_id`  
  Pixelmap id for fill area
- `style`  
  Style of numeric pixelmap prompt, Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- `pixelmap_prompt_id`  
  Application-defined ID of prompt
- `size`  
  Dimensions of numeric pixelmap prompt
Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successfully create numeric pixlemap prompt</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>0x13</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>0x19</td>
<td>Invalid widget control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
/* Create "my_numeric_pix_prompt". */
status = gx_numeric_pixelmap_prompt_create(&my_numeric_pix_prompt,
                                           "my_numeric_pix_prompt", &my_parent,
                                           MY_TEXT_RESOURCE_ID, MY_FEEL_RESOURCE_ID,
                                           GX_STYLE_BORDER_RAISED, MY_NUMERIC_PIXELMAP_PROMPT_ID,
                                           &size);

/* If status is GX_SUCCESS the numeric pixlemap prompt
"my_numeric_pix_prompt" has been created. */
```

See Also

- `gx_numeric_pixelmap_format_function_set`
- `gx_numeric_pixelmap_prompt_value_set`
gx_numeric_pixelmap_prompt_format_function_set

Override format function of numeric pixelmap prompt

Prototype

UINT gx_numeric_pixelmap_prompt_format_function_set(
    GX_NUMERIC_PIXELMAP_PROMPT *prompt,
    VOID (*format_func)(GX_NUMERIC_PIXELMAP_PROMPT *, INT));

Description

This service overrides the default format function of the numeric pixelmap prompt widget. The default format function converts the numeric pixelmap prompt value to a string and stores it in the widget’s private buffer. This service allows the application to define its own format function to format and store the numeric pixelmap prompt value in the widget’s private buffer.

Parameters

| prompt | Numeric pixelmap prompt control block |
| format_func | Format function to be set |

Return Values

| GX_SUCCESS | (0x00) Successfully set numeric pixelmap prompt format function |
| GX_PTR_ERROR | (0x07) Invalid pointer |

Allowed From

Initialization and threads
Example

/* Define my numeric pixelmap format function. */
VOID my_format_function(GX_NUMERIC_PIXELMAP_PROMPT *prompt,
                     INT value)
{
    /* If the value is “1234”, the new format will be “12.34”. */
    INT length;
    gx_utility_ltoa(value / 100,
                     prompt->gx_numeric_pixelmap_prompt_buffer,
                     GX_NUMERIC_PROMPT_BUFFER_SIZE);
    Length = GX_STRLEN(prompt->gx_numeric_pixelmap_prompt_buffer);
    prompt->gx_numeric_pixelmap_prompt_buffer[length++] = '.';
    gx_utility_ltoa(value % 100,
                     prompt->gx_numeric_pixelmap_prompt_buffer + length,
                     GX_NUMERIC_PROMPT_BUFFER_SIZE - length);
}

/* Override default format function of “my_numeric_pix_prompt”. */
status = gx_numeric_pixelmap_prompt_format_function_set(
            &my_numeric_pix_prompt,
            my_format_function);

/* If status is GX_SUCCESS the format function of “my_numeric_pix_prompt” has been override. */

See Also

gx_numeric_pixelmap_prompt_create, gx_numeric_pixelmap_prompt_value_set
**gx_numeric_pixelmap_prompt_value_set**

Set numeric pixelmap prompt value

Prototype

```c
UINT gx_numeric_pixelmap_prompt_value_set(
    GX_NUMERIC_PIXELMAP_PROMPT *prompt,
    INT value);
```

Description

This service an integer value to a numeric pixelmap prompt.

Parameters

- **prompt**: Numeric pixelmap prompt control block
- **value**: Integer value to be set

Return Values

- **GX_SUCCESS** (0x00): Successfully set numeric pixelmap prompt value
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

Allowed From

Initialization and threads

Example

```c
/* Set a value to “my_numeric_pix_prompt”. */
status = gx_numeric_pixelmap_prompt_value_set(&my_numeric_pix_prompt, 1000);

/* If status is GX_SUCCESS the value of the numeric pixelmap prompt “my_numeric_pix_prompt” has been set. */
```

See Also

- `gx_numeric_pixelmap_prompt_create`
- `gx_numeric_pixelmap_format_function_set`
gx_numeric_prompt_create

Create numeric prompt

Prototype

UINT gx_numeric_prompt_create(
    GX_NUMERIC_PROMPT *prompt,
    GX_CONST GX_CHAR name, GX_WIDGET *parent,
    GX_RESOURCE_ID text_id,
    ULONG style, USHORT prompt_id,
    GX_CONST GX_RECTANGLE *size);

Description

This service creates a numeric prompt widget. A numeric prompt is just a prompt that keeps its own buffer and provides a gx_numeric_prompt_value_set(INT) API, the buffer size is defined by the constant GX_NUMERIC_PROMPT_BUFFER_SIZE, which defaults to 16.

GX_NUMERIC_PROMPT is derived from GX_PROMPT and supports all gx_prompt API services.

Parameters

- prompt: Numeric prompt control block
- name: Name of prompt
- parent: Parent widget control block
- text_id: Resource string id
- style: Style of numeric prompt, Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- prompt_id: Application-defined ID of prompt
- size: Dimensions of numeric prompt

Return Values

- **GX_SUCCESS** (0x00): Successfully create numeric prompt
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY_CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
Allowed From

Initialization and threads

Example

/* Create "my_numeric_prompt". */
status = gx_numeric_prompt_create(&my_numeric_prompt,
   "my_numeric_prompt", &my_parent,
   MY_TEXT_RESOURCE_ID, GX_STYLE_BORDER_RAISED,
   MY_NUMERIC_PROMPT_ID,
   &size);

/* If status is GX_SUCCESS the numeric prompt "my_numeric_prompt"
has been created. */

See Also

gx_numeric_format_function_set, gx_numeric_prompt_value_set
**gx_numeric_prompt_format_function_set**

Override format function of numeric prompt

**Prototype**

```c
UINT gx_numeric_format_function_set(
    GX_NUMERIC_PROMPT *prompt,
    VOID (*format_func)(GX_NUMERIC_PROMPT *, INT)
);
```

**Description**

This service overrides the default format function of a numeric prompt widget. The default format function converts the numeric prompt value to a string and stores it in the widget's private buffer. This service allows the application to define its own format function to format and store the numeric prompt value in the widget's private buffer.

**Parameters**

- **prompt**
  - Numeric prompt control block
- **format_func**
  - Format function to be set

**Return Values**

- **GX_SUCCESS** (0x00)
  - Successfully set numeric prompt format function
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer

**Allowed From**

- Initialization and threads
Example

/* Define my numeric format function. */
VOID my_format_function(GX_NUMERIC_PROMPT *prompt, INT value)
{
    /* If the value is “1234”, the new format will be “12.34”. */

    INT length;
    gx_utility_ltoa(value / 100, prompt->gx_numeric_prompt_buffer,
                     GX_NUMERIC_PROMPT_BUFFER_SIZE);
    Length = GX_STRLEN(prompt->gx_numeric_prompt_buffer);
    prompt->gx_numeric_prompt_buffer[length++] = '.';
    gx_utility_ltoa(value % 100,
                     prompt->gx_numeric_prompt_buffer + length,
                     GX_NUMERIC_PROMPT_BUFFER_SIZE - length);
}

/* Override the default format function of “my_numeric_prompt”. */
status = gx_numeric_prompt_format_function_set(&my_numeric_prompt,
                                                my_format_function);

/* If status is GX_SUCCESS, the format function of “my_numeric_prompt” has been override. */

See Also

gx_numeric_prompt_create, gx_numeric_prompt_value_set
gx_numeric_prompt_value_set

Set numeric prompt value

Prototype

UINT  gx_numeric_prompt_value_set(
    GX_NUMERIC_PROMPT *prompt, INT value);

Description

This service sets an integer value to a numeric prompt.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompt</td>
<td>Numeric prompt control block</td>
</tr>
<tr>
<td>value</td>
<td>Integer value to be set</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully set numeric prompt value
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

/* Set a value to “my_numeric_prompt”. */
status = gx_numeric_prompt_value_set(&my_numeric_prompt, 1000);

/* If status is GX_SUCCESS the value of the numeric prompt
“my_numeric_prompt” has been set. */

See Also

gx_numeric_prompt_create, gx_numeric_format_function_set
**gx_numeric_scroll_wheel_create**

Create numeric scroll wheel

### Prototype

```
UINT  gx_numeric_scroll_wheel_create(
    GX_NUMERIC_SCROLL_WHEEL *wheel,
    GX_CONST GX_CHAR *name,
    GX_WIDGET *parent,
    INT start_val,
    INT end_val,
    ULONG style,
    USHORT wheel_id,
    GX_CONST GX_RECTANGLE *size);
```

### Description

This service creates a numeric scroll wheel widget.

A numeric scroll wheel is a type of scroll wheel widget that is specifically used for displaying a range of numbers. Other types of scroll wheel widgets are also available. Refer to the `gx_scroll_wheel_create()` API for more information about the scroll wheel widget hierarchy, widget types, and widget derivation.

GX_NUMERIC_SCROLL_WHEEL is derived from GX_TEXT_SCROLL_WHEEL and supports all gx_text_scroll_wheel services.

All scroll wheel types generate GX_EVENT_LIST_SELECT events to their parent when the scroll wheel is scrolled.

A numeric scroll wheel will default to having \( \text{abs}(\text{end\_val} - \text{start\_val}) + 1 \) rows. In other words, the scroll wheel will display every value between \( \text{start\_val} \) and \( \text{end\_val} \), incrementing or decrementing by 1 with each row. Note that \( \text{start\_val} \) can be greater or less than \( \text{end\_val} \), depending on which way the application wants the range to appear.

If the application wants to change the row increment, it can do this by calling `gx_scroll_wheel_total_rows_set()` after creating the numeric scroll wheel. For example, an application wanting to create a scroll wheel that displays the values years 1980 to 2020, incrementing by 5, might do this:

```
    gx_numeric_scroll_wheel_create(&wheel, GX_NULL, parent, 1980, 2020, style, id, &size);
```
/* the years 1980 through 2020, inclusive, incrementing by 5 years,
yields 9 total rows */

gx_scroll_wheel_total_rows_set(&wheel, 9);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel</td>
<td>Pointer to numeric scroll wheel control block</td>
</tr>
<tr>
<td>name</td>
<td>Logical name of pixelmap button widget</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to the parent widget</td>
</tr>
<tr>
<td>start_val</td>
<td>Starting numeric value</td>
</tr>
<tr>
<td>end_val</td>
<td>Ending numeric value</td>
</tr>
<tr>
<td>style</td>
<td>Style of checkbox. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.</td>
</tr>
<tr>
<td>wheel_id</td>
<td>Application-defined ID of scroll wheel</td>
</tr>
<tr>
<td>size</td>
<td>Dimensions of scroll wheel widget</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully created numeric scroll wheel
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size

Allowed From

 Initialization and threads

Example

```c
/* Create "year_wheel". */
status = gx_numeric_scroll_wheel_create(&year_wheel,
                                   "year_selector",
                                   &parent,
                                   1980, 2040,
                                   GX_STYLE_ENABLED | GX_STYLE_TEXT_CENTER |
                                   GX_STYLE_TRANSPARENT | GX_STYLE_WRAP |
                                   GX_STYLE_TEXT_SCROLL_WHEEL_ROUND,
                                   YEAR_WHEEL_ID,
                                   &size);

/* If status is GX_SUCCESS the scroll wheel "year_wheel" has been created. */
```
See Also

gx_numeric_scroll_wheel_range_set, gx_scroll_wheel_create,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_text_scroll_wheel_callback_set,
gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set,
gx_string_scroll_wheel_create, gx_string_scroll_wheel_text_get
gx_numeric_scroll_wheel_range_set

Assign value range of numeric scroll wheel

Prototype

UINT gx_numeric_scroll_wheel_range_set(GX_NUMERIC_SCROLL_WHEEL *wheel, INT start_val, INT end_val);

Description

This service modifies the range of values allowed and displayed by a numeric scroll wheel widget.

A numeric scroll wheel is a type of scroll wheel widget that is specifically used for displaying a range of numbers. Other types of scroll wheel widgets are also available. Refer to the gx_scroll_wheel_create() API for more information about the scroll wheel widget hierarchy, widget types, and widget derivation.

Invoking this API resets the scroll wheel total rows to

\[ \text{abs}(\text{end_val} - \text{start_val}) + 1 \]

meaning the scroll wheel will increment by 1 for each row. To change this, the application can call gx_scroll_wheel_total_rows_set() to change the total number of row, effectively changing the value increment between rows.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel</td>
<td>Pointer to numeric scroll wheel control block</td>
</tr>
<tr>
<td>start_val</td>
<td>Starting numeric value</td>
</tr>
<tr>
<td>end_val</td>
<td>Ending numeric value</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully set numeric scroll wheel range
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

Allowed From

Initialization and threads
Example

/* Change range of “rate” scroll wheel. */

status = gx_numeric_scroll_wheel_range_set(&year_wheel, 0, 200);

/* If status is GX_SUCCESS the scroll wheel range has been modified. */

See Also

gx_numeric_scroll_wheel_create, gx_scroll_wheel_create,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_text_scroll_wheel_callback_set,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set,
gx_string_scroll_wheel_create, gx_string_scroll_wheel_text_get
### gx_pixelmap_button_create

Create pixelmap button

#### Prototype

```c
UINT  gx_pixelmap_button_create(GX PIXELMAP BUTTON *button,
GX CONST GX CHAR *name,
GX_WIDGET *parent,
GX_RESOURCE ID normal_id,
GX_RESOURCE ID selected_id,
GX_RESOURCE ID disabled_id,
ULONG style,
USHORT pixelmap_button_id,
GX CONST GX_RECTANGLE *size);
```

#### Description

This service creates a pixelmap button widget.

GX_PIXELMAP_BUTTON is derived from GX_BUTTON and supports all gx_button services.

#### Parameters

- **button**: Pointer to pixelmap button control block
- **name**: Logical name of pixelmap button widget
- **parent**: Pointer to the parent widget
- **normal_id**: Normal state Resource ID
- **selected_id**: Selected state Resource ID
- **disabled_id**: Disabled state Resource ID
- **style**: Style of checkbox. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **pixelmap_button_id**: Application-defined ID of pixelmap button
- **size**: Dimensions of pixelmap button

#### Return Values

- **GX_SUCCESS** (0x00): Successfully created pixelmap button
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY_CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
Allowed From

Initialization and threads

Example

/* Create "my_pixelmap_button". */
status = \texttt{gx\_pixelmap\_button\_create}(&my\_pixelmap\_button,
    "my\_pixelmap\_button", &my\_parent,
    MY\_NORMAL\_RESOURCE\_ID, MY\_SELECTED\_RESOURCE\_ID,
    MY\_DESELECTED\_RESOURCE\_ID, GX\_STYLE\_BORDER\_RAISED,
    MY\_PIXELMAP\_BUTTON\_ID,
    &size);

/* If status is GX\_SUCCESS the pixelmap button "my\_pixelmap\_button" has been created. */

See Also

gx\_button\_background\_draw, gx\_button\_create, gx\_button\_deselect,
gx\_button\_draw, gx\_button\_event\_process, gx\_button\_select,
gx\_pixelmap\_button\_draw, gx\_pixelmap\_button\_pixelmap\_set,
gx\_pixelmap\_prompt\_create, gx\_pixelmap\_prompt\_draw,
gx\_pixelmap\_prompt\_pixelmap\_set, gx\_pixelmap\_slider\_create,
gx\_pixelmap\_slider\_draw, gx\_pixelmap\_slider\_event\_process,
gx\_radio\_button\_create, gx\_radio\_button\_draw, gx\_icon\_button\_create,
gx\_text\_button\_create, gx\_text\_button\_color\_set, gx\_text\_button\_draw
gx_pixelmap_button_draw

Draw pixelmap button

Prototype

VOID gx_pixelmap_button_draw(GX_PIXELMAP_BUTTON *button);

Description

This service draws a pixelmap button widget. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom pixelmap button widgets.

Parameters

button Pointer to pixelmap button control block

Return Values

None

Allowed From

Threads

Example

/* Write a custom pixelmap button drawing function. */

VOID my_pixelmap_button_draw(GX_PIXELMAP_BUTTON *button)
{
    /* Call default pixelmap button draw. */
    gx_pixelmap_button_draw(button);

    /* Add your own drawing here. */
}

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_pixelmap_button_create, gx_pixelmap_button_pixelmap_set,
gx_pixelmap_prompt_create, gx_pixelmap_prompt_pixelmap_set,
gx_pixelmap_slider_create, gx_pixelmap_slider_slider_event_process,
gx_radio_button_create, gx_radio_button_draw, gx_icon_button_create,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
**gx_pixelmap_button_event_process**

Pixelmap button event processing

**Prototype**

```c
UINT  gx_pixelmap_button_event_process(GX_PIXELMAP_BUTTON *button, 
                                         GX_EVENT *event_ptr);
```

**Description**

This service provides default event handling for the pixelmap button widget type.

**Parameters**

- `button` Pointer to pixelmap button control block
- `event_ptr` Pointer to GX_EVENT structure

**Return Values**

- `GX_SUCCESS` (0x00) Successful pixelmap button draw
- `GX_CALLER_ERROR` (0x11) Invalid caller of this function
- `GX_PTR_ERROR` (0x07) Invalid pointer

**Allowed From**

Initialization and threads
Example

```c
switch(event_ptr->gx_event_type)
{
    case GX_EVENT_SHOW:
        /* Do default handling. */
        status = gx_pixelmap_button_event_process(icon, event_ptr);
        /* add my own handling here */
        break;
}
```

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect, 
gx_button_draw, gx_button_event_process, gx_button_select, 
gx_pixelmap_button_create, gx_pixelmap_button_pixelmap_set,  
gx_pixelmap_prompt_create, gx_pixelmap_prompt_pixelmap_set,  
gx_pixelmap_prompt_draw, gx_pixelmap_slider_create, 
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process,  
gx_radio_button_create, gx_radio_button_draw, gx_icon_button_create,  
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
gx_pixelmap_button_pixelmap_set

Assign pixelmaps to button

Prototype

UINT gx_pixelmap_button_pixelmap_set(GX_PIXELMAP_BUTTON *button, 
    GX_RESOURCE_ID normal_id, 
    GXRESOURCE_ID selected_id, 
    GXRESOURCE_ID disabled_id);

Description

This service sets pixelmaps to the pixelmap button.

Parameters

- **button**: Pointer to pixelmap button control block
- **normal_id**: Resource ID of the pixelmap to be used as normal state
- **selected_id**: Resource ID of the pixelmap to be used when the button is selected
- **disabled_id**: Resource ID of the pixelmap to be used when the button is disabled

Return Values

- **GX_SUCCESS** (0x00): Successful sets the pixelmap to the button
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

Allowed From

Initialization and threads
Example

/* Draw “my_pixels_button”. */
status = gx_pixelmap_button_pixelmap_set (&my_pixelmap_button,
              NORMAL_ID, SELECTED_ID,
              DISABLED_ID);

/* If status is GX_SUCCESS the pixelmap button is properly
configured with the specified pixelmaps. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_pixelmap_slider_create,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process,
gx_radio_button_create, gx_radio_button_draw, gx_icon_button_create,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
**gx_pixelmap_prompt_create**

Create pixelmap prompt

**Prototype**

```c
UINT gx_pixelmap_prompt_create(GX_PIXELMAP_PROMPT *prompt,
                                 GX_CONST GX_CHAR *name,
                                 GX_WIDGET *parent,
                                 GX_RESOURCE_ID text_id,
                                 GX_RESOURCE_ID fill_pixelmap_id,
                                 ULONG style,
                                 USHORT pixelmap_prompt_id,
                                 GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a pixelmap prompt widget. A pixelmap prompt differs from a standard GX_PROMPT in that it paints the background of the prompt using pixelmaps. The create function accepts one pixelmap id, the normal state fill pixelmap. Up to six pixelmaps may be assigned to the pixelmap prompt.

**Parameters**

- **prompt**: Pointer to pixelmap prompt control block
- **name**: Logical name of pixelmap prompt widget
- **parent**: Pointer to the parent widget
- **text_id**: Resource ID of text
- **fill_id**: Resource ID of fill
- **style**: Style of checkbox. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **pixelmap_prompt_id**: Application-defined ID of pixelmap prompt
- **size**: Dimensions of pixelmap prompt

**Return Values**

- **GX_SUCCESS** (0x00): Successful pixelmap prompt create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
- **GX_ALREADY CREATED** (0x13): Widget already created
Allowed From

Initialization and threads

Example

/* Create "my_pixelmap_prompt". */
status = gx_pixelmap_prompt_create(&my_pixelmap_prompt,
   "my_pixelmap_prompt", &my_parent,
   MY_TEXT_RESOURCE_ID, MY_LEFT_RESOURCE_ID,
   MY_FILL_RESOURCE_ID, MY_RIGHT_RESOURCE_ID,
   GX_STYLE_BORDER_RAISED, MY_PIXELMAP_PROMPT_ID,
   &size);

/* If status is GX_SUCCESS the pixelmap prompt "my_pixelmap_prompt"
has been created. */

See Also

gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_pixelmap_slider_create,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process, gx_prompt_create,
gx_prompt_draw, gx_prompt_font_set, gx_prompt_text_color_set,
gx_prompt_text_get, gx_prompt_text_set
**gx_pixelmap_prompt_draw**

Draw pixelmap prompt

**Prototype**

VOID  gx_pixelmap_prompt_draw(GX_PIXELMAP_PROMPT *prompt);

**Description**

This service draws a pixelmap prompt widget. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom pixelmap prompt widgets.

**Parameters**

**prompt**

Pointer to pixelmap prompt control block

**Return Values**

None

**Allowed From**

Initialization and threads

**Example**

/* Write a custom pixelmap prompt drawing function. */

VOID my_pixelmap_button_draw(GX_PIXELMAP_PROMPT *prompt)
{
    /* Call default pixelmap prompt draw. */
    gx_pixelmap_prompt_draw(prompt);

    /* Add your own drawing here. */
}

**See Also**

gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_create,
gx_pixelmap_prompt_pixelmap_set, gx_pixelmap_slider_create,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process, gx_prompt_create,
gx_prompt_draw, gx_prompt_font_set, gx_prompt_text_color_set,
gx_prompt_text_get, gx_prompt_text_set
gx_pixelmap_prompt_pixelmap_set

Assign pixelmaps to prompt

Prototype

UINT gx_pixelmap_prompt_pixelmap_set(GX_PIXELMAP_PROMPT *prompt,
GX_RESOURCE_ID normal_left_pixelmap,
GX_RESOURCE_ID normal_fill_pixelmap,
GX_RESOURCE_ID normal_right_pixelmap,
GX_Resource_ID selected_left_pixelmap,
GX_RESOURCE_ID selected_fill_pixelmap,
GX_RESOURCE_ID selected_right_pixelmap);

Description

This service assigns pixelmap ids to the pixelmap prompt. The left, fill, and right pixelmap ids are used to allow the application to use one set of pixelmaps for prompts of various widths but a common height to save on storage requirements. If the left and right IDs are not used, they should be set to 0. If the prompt should draw itself differently when it gains input focus, the selected pixelmap ids are used for that purpose. If the selected ids are not used or are the same as the normal ids, set them to 0.

Parameters

prompt Pointer to pixelmap prompt control block
normal_left_id Resource ID of the pixelmap to be used on the left side in the normal state
normal_fill_id Resource ID of the pixelmap to be used as a tiled fill in the normal state
normal_right_id Resource ID of the pixelmap to be used on the right side in the normal state
selected_left_id Resource ID of the pixelmap to be used on the left side in the selected state
selected_fill_id Resource ID of the pixelmap to be used as a tiled fill in the selected state
selected_right_id Resource ID of the pixelmap to be used on the right side in the selected state

Return Values

GX_SUCCESS (0x00) Successful sets the pixelmap to the prompt
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_RESOURCE_ID (0x33) Resource ID not valid
Allowed From

Initialization and threads

Example

/* Assign pixmap IDs to “my_prompt”. Only the normal state
   pixelmaps are used in this case */
status = gx_pixelmap_prompt_pixelmap_set(&my_prompt,
   normal_left_id, normal_fill_id,
   normal_right_id, 0, 0, 0);

/* If status is GX_SUCCESS the pixmap prompt is properly
   configured with pixelmaps. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_create,
gx_pixelmap_prompt_draw, gx_pixelmap_slider_create,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process,
gx_radio_button_create, gx_radio_button_draw, gx_icon_button_create,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
**gx_pixelmap_slider_create**

Create pixmap slider

**Prototype**

```c
UINT gx_pixelmap_slider_create(GX PIXELMAP_SLIDER *slider,
GX CONST GX_CHAR *name, GX_WIDGET *parent,
GX SLIDER_INFO *info,
GX PIXELMAP_SLIDER_INFO *pixelmap_info,
ULONG style, USHORT pixelmap_slider_id,
GX CONST GX_RECTANGLE *size);
```

**Description**

This service creates a pixmap slider widget.

**Parameters**

- **slider**
  Pointer to pixmap slider control block

- **name**
  Logical name of pixmap slider widget

- **parent**
  Pointer to the parent widget

- **info**
  Pointer to a GX_SLIDER_INFO structure which contains values defining the slider minimum value, maximum value, current value, and needle limits. **Appendix I** contains definition for GX_SLIDER_INFO structure.

- **pixelmap_info**
  Pointer to a GX_PIXELMAP_SLIDER_INFO structure which defines the pixelmaps used to draw the slider background and needle. **Appendix I** contains definition for GX_PIXELMAP_SLIDER_INFO structure. The slider background can use one or two pixelmaps. If one, the background does not change as the needle moves. If two backgrounds are defined, the background before the needle uses the first background pixelmap, and the background after the needle uses the second background pixelmap.

- **style**
  Style of slider. **Appendix D** contains predefined general styles for all widgets as well as widget-specific styles.

- **pixelmap_slider_id**
  Application-defined ID of pixmap slider
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully created pixelmap slider</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED (0x13)</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE (0x19)</td>
<td>Invalid widget control block size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_SLIDER_INFO info;
GX_PIXELMAP_SLIDER_INFO pixelmap_info;

/* Initiate slider information structure. */
info.gx_slider_info_min_val = 0;
info.gx_slider_info_max_val = 100;
info.gx_slider_info_current_val = 50;
info.gx_slider_info_min_travel = 10;
info.gx_slider_info_max_travel = 10;
info.gx_slider_info_needle_width = 5;
info.gx_slider_info_needle_height = 10;
info.gx_slider_info_needle_inset = 5;
info.gx_slider_info_needle_hotspot_offset;

/* Initiate pixelmap slider information structure. */
pixelmap_info.gx_pixelmap_slider_info_lower_background_pixelmap = GX_PIXELMAP_ID_ORANGE;
pixelmap_info.gx_pixelmap_slider_info_upper_background_pixelmap = GX_PIXELMAP_ID_EMPTY;
pixelmap_info.gx_pixelmap_slider_info_needle_pixelmap = GX_PIXELMAP_ID_NEEDLE;

/* Create “my_pixelmap_slider”. */
status = gx_pixelmap_slider_create(&my_pixelmap_slider, "my_pixelmap_slider", &my_parent,
info, &pixelmap_info,
GX_STYLE_BORDER_RAISED,
MY_PIXELMAP_SLIDER_ID, &size);

/* If status is GX_SUCCESS the pixelmap slider “my_pixelmap_slider” has been created. */
See Also

gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_create,
gx_pixelmap_prompt_draw, gx_pixelmap_prompt_pixelmap_set,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process, gx_slider_create,
gx_slider_draw, gx_slider_event_process, gx_slider_needle_draw,
gx_slider_needle_position_get, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set


**gx_pixelmap_slider_draw**

**Draw pixelmap slider**

**Prototype**

VOID  **gx_pixelmap_slider_draw**(GX_PIXELMAP_SLIDER *slider);

**Description**

This service draws a pixelmap slider widget. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom pixelmap slider widgets.

**Parameters**

- **slider**: Pointer to pixelmap slider control block

**Return Values**

None

**Allowed From**

Threads

**Example**

/* Write a custom pixelmap slider drawing function. */

VOID my_pixelmap_slider_draw(GX_PIXELMAP_SLIDER *pixelmap_slider)
{
    /* Call default pixelmap slider draw. */
    gx_pixelmap_slider_draw(pixelmap_slider);
    /* Add your own drawing here. */
}

**See Also**

- gx_pixelmap_button_create, gx_pixelmap_button_draw,
- gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_create,
- gx_pixelmap_prompt_draw, gx_pixelmap_prompt_pixelmap_set,
- gx_pixelmap_slider_create, gx_pixelmap_slider_event_process,
- gx_pixelmap_slider_pixelmap_set, gx_slider_create, gx_slider_draw,
- gx_slider_event_process, gx_slider_needle_draw, gx_slider_needle_position_get,
- gx_slider_needle_position_get, gx_slider_tickmarks_draw, gx_slider_travel_get,
- gx_slider_value_calculate, gx_slider_value_set
**gx_pixelmap_slider_event_process**

**Process pixelmap slider event**

**Prototype**

```c
UINT gx_pixelmap_slider_event_process(GX_PIXELMAP_SLIDER *slider, GX_EVENT *event);
```

**Description**

This service processes an event for the specified pixelmap slider widget.

**Parameters**

- `slider` Pointer to pixelmap slider control block
- `event` Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) Successful pixelmap slider event process
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads
Example

/* Write a custom event processing function. */
UINT my_event_handler(GX_PIXELMAP_SLIDER *pixelmap_slider, GX_EVENT *event_ptr)
{
    switch(event_ptr->gx_event_type)
    {
    case GX_EVENT_SHOW:
        /* Do default handling. */
        status = gx_pixelmap_slider_event_process(pixelmap_slider, event_ptr);
        /* add my own handling here */
        break;
    default:
        status = gx_pixelmap_slider_event_process(pixelmap_slider, event_ptr);
        break;
    }
    return status;
}

See Also

gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_button_pixelmap_set, gx_pixelmap_prompt_create,
gx_pixelmap_prompt_draw, gx_pixelmap_prompt_pixelmap_set,
gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_pixelmap_set, gx_slider_create, gx_slider_draw,
gx_slider_event_process, gx_slider_needle_draw,
gx_slider_needle_position_get, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set
gx_pixelmap_slider_pixelmap_set

Assign pixelmaps to slider

Prototype

UINT  gx_pixelmap_slider_pixelmap_set(GX_PIXELMAP_SLIDER *slider,
                                  GX_PIXELMAP_SLIDER_INFO *pixinfo);

Description

This service sets pixelmaps to the pixelmap slider.

Parameters

<table>
<thead>
<tr>
<th>slider</th>
<th>Pointer to pixelmap slider control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixinfo</td>
<td>Pointer to aGX_PIXELMAP_SLIDER_INFO structure which defines the pixelmaps used to draw the slider background and needle. Appendix I contains definition for GX_PIXELMAP_SLIDER_INFO structure. The slider background can use one or two pixelmaps. If one, the background does not change as the needle moves. If two backgrounds are defined, the background before the needle uses the first background pixelmap, and the background after the needle uses the second background pixelmap.</td>
</tr>
</tbody>
</table>

Return Values

| GX_SUCCESS       | (0x00) Successful sets the pixelmap to the slider |
| GX_CALLER_ERROR | (0x11) Invalid caller of this function |
| GX_PTR_ERROR     | (0x07) Invalid pointer |

Allowed From

Initialization and threads
Example

GX_PIXELMAP_SLIDER_INFO pixelmap_info;

/* Initiate pixelmap slider information structure. */
pixelmap_info.gx_pixelmap_slider_info_lower_background_pixelmap = GX_PIXELMAP_ID_ORANGE;
pixelmap_info.gx_pixelmap_slider_info_upper_background_pixelmap = GX_PIXELMAP_ID_EMPTY;
pixelmap_info.gx_pixelmap_slider_info_needle_pixelmap = GX_PIXELMAP_ID_NEEDLE;

/* Draw "my_pixelmap_button". */
status = gx_pixelmap_slider_pixelmap_set(&my_pixelmap_slider, &pixelmap_info);

/* If status is GX_SUCCESS the pixelmap slider is properly configured with "pixelmap_info". */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_pixelmap_slider_create,
gx_pixelmap_slider_draw, gx_pixelmap_slider_event_process,
gx_radio_button_create, gx_radio_button_draw, gx_icon_button_create,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw
**gx_progress_bar_background_draw**  
Draw progress bar background

**Prototype**

```c
VOID  gx_progress_bar_background_draw(
    GX_PROGRESS_BAR *progress_bar)
```

**Description**

This service draws the background of the specified progress bar. This function is called internally as part of the `gx_progress_bar_draw()`, but is exposed to the application to support those cases where the application defines a custom progress bar drawing function.

**Parameters**

- **progress bar**  
  Pointer to progress bar control block

**Return Values**

- None

**Allowed From**

- Threads

**Example**

```c
/* Write a custom progress bar drawing function. */

VOID my_progress_bar_draw(GX_PROGRESS_BAR *progress_bar)
{
    /* Call default progress bar background draw. */
    gx_progress_bar_background_draw(progress_bar);

    /* Call default progress bar text draw. */
    gx_progress_bar_text_draw(progress_bar);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_progress_bar_create`, `gx_progress_bar_draw`,  
- `gx_progress_bar_event_process`, `gx_progress_bar_font_set`,  
- `gx_progress_bar_info_set`, `gx_progress_bar_pielmap_set`,  
- `gx_progress_bar_range_set`, `gx_progress_bar_text_color_set`,  
- `gx_progress_bar_value_set`
**gx_progress_bar_create**

Create a progress bar

### Prototype

```c
UINT  gx_progress_bar_create(GX_PROGRESS_BAR *progress_bar,
                              GX_CONST GX_CHAR *name,
                              GX_WIDGET *parent,
                              GX_PROGRESS_BAR_INFO *progress_bar_info,
                              ULONG style, USHORT progress_bar_id,
                              GX_CONST GX_RECTANGLE *size);
```

### Description

This service creates a progress bar widget.

### Parameters

- **progress_bar**: Progress bar control block
- **name**: Logical name
- **parent**: Pointer to the parent widget
- **progress_bar_info**: Pointer to a GX_PROGRESS_BAR_INFO structure. [Appendix I](#) contains definition for GX_PROGRESS_BAR_INFO structure.
- **style**: Style of progress bar. [Appendix D](#) contains pre-defined general styles for all widgets as well as widget-specific styles.
- **progress_bar_id**: Application-defined ID of progress bar
- **size**: Dimensions of progress bar

### Return Values

- **GX_SUCCESS**: (0x00) Successful progress bar create
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_ALREADY.Created**: (0x13) Widget already created
- **GX_INVALID_SIZE**: (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET**: (0x12) Parent widget not valid

### Allowed From

Initialization and threads
Example

GX_PROGRESS_BAR_INFO info;
GX_RECTANGLE size;

info.gx_progress_bar_info_min_val = 0;
info.gx_progress_bar_info_max_val = 100;
info.gx_progress_bar_info_current_val = 0;
info.gx_progress_bar_font_id = GX_FONT_ID_SYSTEM_FONT;
info.gx_progress_bar_normal_text_color = GX_COLOR_ID_WHITE;
info.gx_progress_bar_selected_text_color = GX_COLOR_ID_BLUE;
info.gx_progress_bar_fill_pixelmap = 0;

size.gx_rectangle_left = 10;
size.gx_rectangle_top = 10;
size.gx_rectangle_right = 110;
size.gx_rectangle_bottom = 140;

/* Create a progress bar with the specified information. */
status = gx_progress_bar_create(&my_progress_bar, GX_NULL, GX_NULL,
                                &info, GX_STYLE_BORDER_THIN,
                                0, &size);

/* If status is GX_SUCCESS the progress bar "my_progress_bar" has
been successfully created. */

See Also

gx_progress_bar_draw, gx_progress_bar_event_process,
gx_progress_bar_font_set, gx_progress_bar_info_set,
gx_progress_bar_pixmap_set, gx_progress_bar_range_set,
gx_progress_bar_text_color_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set
gx_progress_bar_draw

Draw a progress bar

Prototype

VOID gx_progress_bar_draw(GX_PROGRESS_BAR *progress_bar);

Description

This service draws a progress bar widget. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom progress bar widgets.

Parameters

progress_bar Progress bar control block

Return Values

None

Allowed From

Threads

Example

/* Write a custom progress bar drawing function. */

VOID my_progress_bar_draw(GX_PROGRESS_BAR *progress_bar)
{
    /* Call default progress bar draw. */
    gx_progress_bar_draw(progress_bar);

    /* Add your own drawing here. */
}

See Also

gx_progress_bar_create, gx_progress_bar_event_process, gx_progress_bar_font_set, gx_progress_bar_info_set, gx_progress_bar_pielmap_set, gx_progress_bar_range_set, gx_progress_bar_text_color_set, gx_progress_bar_text_draw, gx_progress_bar_value_set
**gx_progress_bar_event_process**  
Progress a progress bar event

**Prototype**

```c
UINT gx_progress_bar_event_process(GX_PROGRESS_BAR *progress_bar,  
                                   GX_EVENT *event_ptr);
```

**Description**

This service processes a progress bar event.

**Parameters**

- **progress_bar**: Progress bar control block
- **event_ptr**: Pointer to GX_EVENT structure

**Return Values**

- **GX_SUCCESS** (0x00) Successful prompt create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Invalid widget

**Allowed From**

Initialization and threads

**Example**

```c
/* Write a custom event processing function. */

UINT my_event_process (GX_PROGRESS_BAR *progress_bar, GX_EVENT *event_ptr)
{
    switch(event_ptr->gx_event_type)
    {
        case GX_EVENT_SHOW:
            /* Do default handling. */
            status = gx_progress_bar_event_process(progress_bar,  
                                                    event_ptr);

            /* add my own handling here */
            break;
        default:
            status = gx_progress_bar_event_process(progress_bar,  
                                                    event_ptr);
            break;
    }

    return status;
}
```
See Also

gx_progress_bar_create, gx_progress_bar_event_draw,
gx_progress_bar_font_set, gx_progress_bar_info_set,
gx_progress_bar_pielmap_set, gx_progress_bar_range_set,
gx_progress_bar_text_color_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set
**gx_progress_bar_font_set**

Set font of progress bar text

**Prototype**

```c
UINT gx_progress_bar_font_set(GX_PROGRESS_BAR *progress_bar,
                               GXRESOURCE_ID font_id);
```

**Description**

This service sets the font of a progress bar widget.

**Parameters**

- **progress_bar**: Progress bar control block
- **font_id**: Font resource id

**Return Values**

- **GX_SUCCESS** (0x00) Successful progress bar font set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
UINT status = gx_progress_bar_font_set(&progress_bar,
                                        GX_FONT_ID_MEDIUM);
/* if status is GX_SUCCESS, the font was successfully assigned. */
```

**See Also**

- `gx_progress_bar_create`, `gx_progress_bar_draw`,
- `gx_progress_bar_event_process`, `gx_progress_bar_info_set`,
- `gx_progress_bar_pixmap_set`, `gx_progress_bar_range_set`,
- `gx_progress_bar_text_color_set`, `gx_progress_bar_text_draw`,
- `gx_progress_bar_value_set`
gx_progress_bar_info_set

Set progress bar information structure

Prototype

UINT gx_progress_bar_info_set(GX_PROGRESS_BAR *progress_bar,
                              GX_PROGRESS_BAR_INFO *info);

Description

This service resets the information structure of a progress bar widget.

Parameters

progress_bar: Progress bar control block
info: Pointer to a GX_PROGRESS_BAR_INFO structure. Appendix I contains definition for GX_PROGRESS_BAR_INFO structure.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully reset progress bar info</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_PROGRESS_BAR_INFO info;

info.gx_progress_bar_info_min_val = 0;
info.gx_progress_bar_info_max_val = 100;
info.gx_progress_bar_info_current_val = 0;
info.gx_progress_bar_font_id = GX_FONT_ID_SYSTEM_FONT;
info.gx_progress_bar_normal_text_color = GX_COLOR_ID_WHITE;
info.gx_progress_bar_selected_text_color = GX_COLOR_ID_BLUE;
info.gx_progress_bar_fill_pixelmap = 0;

status = gx_progress_bar_info_set(&progress_bar, &info);

/* if status == GX_SUCCESS the progress bar info was re-assigned.
 */

See Also

gx_progress_bar_info_create, gx_progress_bar_draw,
gx_progress_bar_event_process, gx_progress_bar_font_set,
gx_progress_bar_pixmap_set, gx_progress_bar_range_set,
gx_progress_bar_text_color_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set


**gx_progress_bar_pixelmap_set**

Set pixelmap used to draw progress bar

**Prototype**

```c
UINT  gx_progress_bar_pixelmap_set(GX_PROGRESS_BAR *progress_bar,
                                   GX_RESOURCE_ID pixelmap_id);
```

**Description**

This service sets the pixelmap used to fill the progress bar background.

**Parameters**

- `progress_bar`: Progress bar control block
- `pixelmap_id`: Pixelmap resource id

**Return Values**

- **GX_SUCCESS** (0x00): Successful progress bar pixelmap set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
UINT status = gx_progress_bar_pixelmap_set(&progress_bar,
                                           GX_PIXELMAP_ID_PROGRESS_FILL);
/* if status is GX_SUCCESS, the pixelmap was successfully assigned. */
```

**See Also**

gx_progress_bar_pixelmap_create, gx_progress_bar_draw,
gx_progress_bar_event_process, gx_progress_bar_font_set,
gx_progress_bar_info_set, gx_progress_bar_range_set,
gx_progress_bar_text_color_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set
gx_progress_bar_range_set

Set value range of a progress bar

Prototype

UINT  gx_progress_bar_range_set(GX_PROGRESS_BAR *progress_bar,
                                 INT min_value, INT max_value);

Description

This service sets the progress bar value range.

Parameters

progress_bar  Progress bar control block
min_value     Progress bar minimum value
max_value     Progress bar maximum value

Return Values

GX_SUCCESS   (0x00)  Successful progress bar range set
GX_CALLER_ERROR  (0x11)  Invalid caller of this function
GX_PTR_ERROR   (0x07)  Invalid pointer

Allowed From

Initialization and threads

Example

UINT status = gx_progress_bar_range_set(progress_bar, 0, 100);
/* if status is GX_SUCCESS, the progress bar range was successfully assigned. */

See Also

gx_progress_bar_range_create, gx_progress_bar_draw,
gx_progress_bar_event_process, gx_progress_bar_font_set,
gx_progress_bar_info_set, gx_progress_bar_pielmap_set,
gx_progress_bar_text_color_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set
**gx_progress_bar_text_color_set**

Set the text color of a progress bar

**Prototype**

```c
UINT gx_progress_bar_text_color_set(GX_PROGRESS_BAR *progress_bar,
                                    GX_RESOURCE_ID normal_text_color,
                                    GX_RESOURCE_ID selected_text_color,
                                    GX_RESOURCE_ID disabled_text_color);
```

**Description**

This service sets the text color of a progress bar widget.

**Parameters**

- `progress_bar`: Progress bar control block
- `normal_text_color`: Resource ID of normal text color that used in normal state
- `selected_text_color`: Resource ID of selected text color that used when the widget gain focus
- `disabled_text_color`: Resource ID of disabled text color that used when GX_STYLE_ENABLED is not active

**Return Values**

- **GX_SUCCESS** (0x00): Successful progress bar text color set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads
Example

/* Set text colors for the progress bar “my_progress_bar”*/
UINT status = gx_progress_bar_text_color_set(&my_progress_bar,
    GX_COLOR_ID_NORMAL_TEXT,
    GX_COLOR_ID_SELECTED_TEXT,
    GX_COLOR_ID_DISABLED_TEXT);

/* if status is GX_SUCCESS, the progress bar text colors were
 successfully assigned. */

See Also

gx_progress_bar_create, gx_progress_bar_draw,
gx_progress_bar_event_process, gx_progress_bar_font_set,
gx_progress_bar_info_set, gx_progress_bar_pielmap_set,
gx_progress_bar_range_set, gx_progress_bar_text_draw,
gx_progress_bar_value_set
**gx_progress_bar_text_draw**

*Draw progress bar text*

**Prototype**

```c
VOID gx_progress_bar_text_draw(GX_PROGRESS_BAR *progress_bar)
```

**Description**

This service draws the text of specified progress bar. This function is called internally as part of the `gx_progress_bar_draw()`, but is exposed to the application to support those cases where the application defines a custom progress bar drawing function.

**Parameters**

- **progress bar**: Pointer to progress bar control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom progress bar drawing function. */

VOID my_progress_bar_draw(GX_PROGRESS_BAR *progress_bar)
{
    /* Call default progress bar background draw. */
    gx_progress_bar_background_draw(progress_bar);

    /* Call default progress bar text draw. */
    gx_progress_bar_text_draw(progress_bar);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_progress_bar_create`, `gx_progress_bar_draw`,
- `gx_progress_bar_event_process`, `gx_progress_bar_font_set`,
- `gx_progress_bar_info_set`, `gx_progress_bar_pielmap_set`,
- `gx_progress_bar_range_set`, `gx_progress_bar_text_color_set`,
- `gx_progress_bar_value_set`
gx_progress_bar_value_set

Set current value of a progress bar

Prototype

UINT gx_progress_bar_value_set(GX_PROGRESS_BAR *progress_bar,
                                INT value);

Description

This service assigns the progress bar current value. The progress
bar widget will automatically invalidate and redraw itself when the
progress bar value is changed.

Parameters

progress_bar
value

Progress bar control block
Progress bar current value

Return Values

GX_SUCCESS (0x00) Successful set the value of
the progress bar
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

UINT status = gx_progress_bar_value_set(progress_bar, 50);

/* if status == GX_SUCCESS the progress bar value was successfully
assigned. */

See Also

gx_progress_bar_value_create, gx_progress_bar_draw,
gx_progress_bar_event_process, gx_progress_bar_font_set,
gx_progress_bar_info_set, gx_progress_bar_pielmap_set,
gx_progress_bar_range_set, gx_progress_bar_text_color_set,
gx_progress_bar_text_draw
**gx_prompt_create**

Create prompt

**Prototype**

```c
UINT gx_prompt_create(GX_PROMPT *prompt, GX_CONST GX_CHAR *name, GX_WIDGET *parent, GX_RESOURCE_ID text_id, ULONG style, USHORT prompt_id, GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a prompt widget.

GX_PROMPT is derived from GX_WIDGET and supports all gx_widget services.

**Parameters**

- **prompt**: Pointer to prompt control block
- **name**: Logical name of prompt widget
- **parent**: Pointer to the parent widget
- **text_id**: Resource ID of prompt text
- **style**: Style of prompt. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **prompt_id**: Application-defined ID of prompt
- **size**: Dimensions of prompt

**Return Values**

- **GX_SUCCESS** (0x00): Successful prompt create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY_CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size

**Allowed From**

Initialization and threads
Example

/* Create "my_prompt", */
status = gx_prompt_create(&my_prompt, "my_promPt", &my_parent,
                        MY_PROMPT TEXT RESOURCE ID,
                        GX_STYLE BORDER RAISED, MY_PROPMT ID, &size);

/* If status is GX_SUCCESS the prompt "my_prompt" has been created.
 */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_draw, gx_prompt_font_set,
gx_prompt_text_color_set, gx_prompt_text_get, gx_prompt_text_id_set,
gx_prompt_text_set
**gx_prompt_draw**

**Draw prompt**

**Prototype**

```c
VOID gx_prompt_draw(GX_PROMPT *prompt);
```

**Description**

This service draws a prompt widget. This service is called internally by GUIX during canvas refresh, but can also be called by custom drawing functions.

**Parameters**

- **prompt**
  Pointer to prompt widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom prompt drawing function. */

VOID my_prompt_draw(GX_PROMPT *prompt)
{
    /* Call default prompt draw. */
    gx_prompt_draw(prompt);
    /* Add your own drawing here. */
}
```

**See Also**

- `gx_pixelmap_prompt_create`, `gx_pixelmap_prompt_draw`
- `gx_pixelmap_prompt_pixelmap_set`, `gx_prompt_create`, `gx_prompt_font_set`
- `gx_prompt_text_color_set`, `gx_prompt_text_get`, `gx_prompt_text_id_set`
- `gx_prompt_text_set`
gx_prompt_font_set

Set prompt font

Prototype

UINT  gx_prompt_font_set(GX_PROMPT *prompt,
                          GX_RESOURCE_ID font_id);

Description

This service sets the font of a prompt widget.

Parameters

<table>
<thead>
<tr>
<th>prompt</th>
<th>Pointer to prompt widget control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>font_id</td>
<td>Resource ID of font</td>
</tr>
</tbody>
</table>

Return Values

| GX_SUCCESS      | (0x00) Successful prompt font set       |
| GX_CALLER_ERROR | (0x11) Invalid caller of this function  |
| GX_PTR_ERROR    | (0x07) Invalid pointer                  |
| GX_INVALID_WIDGET | (0x12) Widget not valid            |

Allowed From

Initialization and threads

Example

/* Set the font of "my_prompt". */
status = gx_prompt_font_set(&my_prompt, MY_PROMPT_FONT_ID);

/* If status is GX_SUCCESS the font for prompt "my_prompt" has been set. */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_create, gx_prompt_draw,
gx_prompt_text_color_set, gx_prompt_text_get, gx_prompt_text_id_set,
gx_prompt_text_set
gx_prompt_text_color_set

Set prompt text color

Prototype

UINT gx_prompt_text_color_set(GX_PROMPT *prompt,
                               GX_RESOURCE_ID normal_color,
                               GX_RESOURCE_ID selected_color,
                               GX_RESOURCE_ID disabled_color);

Description

This service sets the text color of a prompt widget.

Parameters

prompt

Pointer to prompt widget control block

normal_color

Resource ID of color for normal text.

Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

selected_color

Resource ID of color for selected text, used when the widget gain focus.

Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

disabled_color

Resource ID of color for disabled text, used when GX_STYLE_ENABLED is not active. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

Return Values

<table>
<thead>
<tr>
<th>Enum</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successful prompt text color set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET_SIZE</td>
<td>0x14</td>
<td>Invalid widget size</td>
</tr>
</tbody>
</table>
Allowed From

Initialization and threads

Example

/* Set the text color of “my_prompt”. */
status = gx_prompt_text_color_set(&my_prompt,
    GX_COLOR_ID_BLACK,
    GX_COLOR_ID_LIGHTGRAY,
    GX_COLOR_ID_DISABLED_TEXT);

/* If status is GX_SUCCESS the text color for prompt “my_prompt”
has been set. */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_create, gx_prompt_draw,
gx_prompt_font_set, gx_prompt_text_get, gx_prompt_text_id_set,
gx_prompt_text_set
**gx_prompt_text_draw**

Drawing support function

**Prototype**

```c
VOID gx_prompt_text_draw(GX_PROMPT *prompt)
```

**Description**

This support function draws the text portion of a prompt. This function is called internally by `gx_prompt_draw()`, and is provided as a separate API as a convenience for applications that define a custom prompt drawing function. Applications that want to customize the prompt background drawing can provide their custom drawing function, and invoke the `gx_prompt_text_draw` service as part of their custom drawing to draw the prompt text over the background.

**Parameters**

- **prompt**: Pointer to the prompt control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Define a custom drawing function */
VOID my_prompt_draw(GX_PROMPT *prompt)
{
    /* insert code here to draw prompt background */

    /* call support function to do text drawing */
    gx_prompt_text_draw();

    /* draw child widgets */
    gx_widget_children_draw((GX_WIDGET *) prompt);
}
```

**See Also**

- `gx_pixelmap_prompt_create`, `gx_pixelmap_prompt_draw`,
- `gx_pixelmap_prompt_pixelmap_set`, `gx_prompt_create`, `gx_prompt_draw`,
- `gx_prompt_font_set`, `gx_prompt_text_color_set`, `gx_prompt_text_id_set`,
- `gx_prompt_text_set`
gx_prompt_text_get
Get prompt text (deprecated)

Prototype

UINT gx_prompt_text_get(GX_PROMPT *prompt,
                        GX_CHAR **return_text);

Description

This service is deprecated in favor of gx_prompt_text_get_ext().

This service gets the text of a prompt widget.

Parameters

prompt Pointer to prompt widget control block
return_text Pointer to destination for text

Return Values

GX_SUCCESS (0x00) Successful prompt text get
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads

Example

GX_PROMPT my_prompt;
GX_CHAR *my_prompt_text;

/* Get the text of “my_prompt”. */
status = gx_prompt_text_get(&my_prompt, &my_prompt_text);

/* If status is GX_SUCCESS the pointer “my_prompt_text” points to
the text displayed by “my_prompt”. */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_create, gx_prompt_draw,
gx_prompt_font_set, gx_prompt_text_color_set, gx_prompt_text_id_set,
gx_prompt_text_set
**gx_prompt_text_get_ext**

Get prompt text

**Prototype**

```c
UINT gx_prompt_text_get(GX_PROMPT *prompt,
                         GX_STRING *return_string);
```

**Description**

This service gets the string of a prompt widget.

**Parameters**

```
prompt      Pointer to prompt widget control block
return_string Pointer to destination for string
```

**Return Values**

```
GX_SUCCESS (0x00) Successful prompt text get
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid
```

**Allowed From**

Initialization and threads

**Example**

```c
GX_PROMPT my_prompt;
GX_STRING my_prompt_string;

/* Get the text of ”my_prompt”. */
status = gx_prompt_text_get_ext(&my_prompt, &my_prompt_string);

/* If status is GX_SUCCESS then my_prompt_string has been initialize to hold a copy of the prompt string. */
```

**See Also**

- `gx_pixelmap_prompt_create`, `gx_pixelmap_prompt_draw`
- `gx_pixelmap_prompt_pixelmap_set`, `gx_prompt_create`, `gx_prompt_draw`
- `gx_prompt_font_set`, `gx_prompt_text_color_set`, `gx_prompt_text_id_set`
- `gx_prompt_text_set`
**gx_prompt_text_id_set**

Set prompt text ID

**Prototype**

```c
UINT gx_prompt_text_id_set(GX_PROMPT *prompt,
                           GX_RESOURCE_ID string_id)
```

**Description**

This service sets the string ID for the text prompt widget.

**Parameters**

- `prompt`: Pointer to prompt widget control block
- `string_id`: Resource ID of the string

**Return Values**

- **GX_SUCCESS** (0x00): Successful prompt text ID set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_RESOURCE_ID** (0x33): Invalid resource ID
- **GX_SYSTEM_MEMORY_ERROR** (0x30): Memory free function is not defined

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the string ID of "my_prompt". */
status = gx_prompt_text_id_set(&my_prompt, MY_STRING_ID);

/* If status is GX_SUCCESS the text ID for prompt "my_prompt" has been set. */
```

**See Also**

- `gx_pixelmap_prompt_create`, `gx_pixelmap_prompt_draw`
- `gx_pixelmap_prompt_pixelmap_set`, `gx_prompt_create`, `gx_prompt_draw`
- `gx_prompt_font_set`, `gx_prompt_text_get`, `gx_prompt_text_set`
gx_prompt_text_set

Set prompt text (deprecated)

Prototype

UINT gx_prompt_text_set(GX_PROMPT *prompt, GX_CHAR *text);

Description

This service has been deprecated in favor of gx_prompt_text_set_ext().

This service sets the text of a prompt widget. If the prompt widget was created with style GX_STYLE_TEXT_COPY, the widget creates a private copy of the text string assigned. If GX_STYLE_TEXT_COPY is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

GX_PROMPT is derived from GX_WIDGET, and therefore all gx_widget API services may be used with GX_PROMPT.

Parameters

prompt

Pointer to prompt widget control block

text

Pointer to text

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful prompt text set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_SYSTEM_MEMORY_ERROR</td>
<td>Memory allocate function is not defined</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Set the text of “my_prompt” to “my_text”. */
status = gx_prompt_text_set(&my_prompt, “my_text”);

/* If status is GX_SUCCESS the text for “my_prompt” has been set. */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_create, gx_prompt_draw,
gx_prompt_font_set, gx_prompt_text_color_set, gx_prompt_text_id_set,
gx_prompt_text_get
gx_prompt_text_set_ext

Set prompt text

Prototype

UINT gx_prompt_text_set_ext(GX_PROMPT *prompt, GX_STRING *string);

Description

This service sets the text of a prompt widget. If the prompt widget was created with style GX_STYLE_TEXT_COPY, the widget creates a private copy of the text string assigned. If GX_STYLE_TEXT_COPY is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

GX_PROMPT is derived from GX_WIDGET, and therefore all gx_widget API services may be used with GX_PROMPT.

Parameters

<table>
<thead>
<tr>
<th>prompt</th>
<th>Pointer to prompt widget control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Pointer to text</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful prompt text set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_SYSTEM_MEMORY_ERROR** (0x30) Memory allocate function is not defined
- **GX_INVALID_STRING_LENGTH** (0x34) Invalid string length

Allowed From

Initialization and threads
Example

GX_STRING new_string;
new_string.gx_string_ptr = "my_text";
new_string.gx_string_length = strlen(new_string.gx_string_ptr);

/* Set the text of “my_prompt” to “new_string”. */
status = gx_prompt_text_set(&my_prompt, &new_string);

/* If status is GX_SUCCESS the text for “my_prompt” has been set. */

See Also

gx_pixelmap_prompt_create, gx_pixelmap_prompt_draw,
gx_pixelmap_prompt_pixelmap_set, gx_prompt_create, gx_prompt_draw,
gx_prompt_font_set, gx_prompt_text_color_set, gx_prompt_text_id_set,
gx_prompt_text_get
**gx_radial_progress_bar_anchor_set**

Set starting angle

**Prototype**

```c
UINT gx_radial_progress_bar_anchor_set(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_VALUE angle);
```

**Description**

This service sets the starting angle for radial progress bar.

**Parameters**

- **progress bar**: Pointer to radial progress bar control block
- **angle**: Starting angle of the circular arc

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful radial progress bar anchor set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>Invalid widget</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_VALUE start_angle = 90;

/* Set the start angle of "my_progress_bar" to 90 degree. */
status = gx_radial_progress_bar_anchor_set(&my_progress_bar,
                                           start_angle);

/* If status is GX_SUCCESS the anchor value of "my_progress_bar" has been set. */
```

**See Also**

- `gx_radial_progress_bar_background_draw`
- `gx_radial_progress_bar_create`
- `gx_radial_progress_bar_draw`
- `gx_radial_progress_bar_event_process`
- `gx_radial_progress_bar_font_set`
- `gx_radial_progress_bar_info_set`
- `gx_radial_progress_bar_text_color_set`
- `gx_radial_progress_bar_text_draw`
- `gx_radial_progress_bar_value_set`
**gx_radial_progress_bar_background_draw**

**Draw background**

**Prototype**

```c
VOID gx_radial_progress_bar_background_draw(
    GX_RADIAL_PROGRESS_BAR *progress_bar);
```

**Description**

This service draws a radial progress bar background. This service is internally referenced by the `gx_radial_progress_bar_draw` function, but is exposed for use by the application in those cases where the application defines a custom radial progress bar drawing function.

**Parameters**

- **progress bar**
  Pointer to radial progress bar control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom radial progress bar drawing function. */

VOID my_radial_progress_bar_draw(GX_RADIAL_PROGRESS_BAR *radial_progress)
{
    /* Call default radial progress bar background draw. */
    gx_radial_progress_bar_background_draw(radial_progress);

    /* Add your own drawing here. */

    /* Draw child widgets. */
    gx_widget_children_draw((GX_WIDGET *)radial_progress);
}
```

**See Also**

- `gx_radial_progress_bar_anchor_set`, `gx_radial_progress_bar_create`, `gx_radial_progress_bar_draw`, `gx_radial_progress_bar_event_process`, `gx_radial_progress_bar_font_set`, `gx_radial_progress_bar_info_set`, `gx_radial_progress_bar_text_color_set`, `gx_radial_progress_bar_text_draw`, `gx_radial_progress_bar_value_set`
gx_radial_progress_bar_create

Create radial progress bar

Prototype

```c
UINT gx_radial_progress_bar_create(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_CONST GX_CHAR *name,
    GX_WIDGET *parent,
    GX_RADIAL_PROGRESS_BAR_INFO *info,
    ULONG style
    USHORT id);
```

Description

This service creates a radial progress bar.

If the widget style GX_STYLE_ENABLED is applied to the progress bar, the progress bar will accept pen_down, pen_drag, and pen_up input to modify the progress bar current value.

The widget style GX_STYLE_PROGRESS_TEXT_DRAW can be used to enable drawing the progress bar value as text within the progress bar area. If this style is used in combination with the style GX_STYLE_PROGRESS_PERCENT, the progress bar value is displayed as a percentage. Otherwise the progress bar value is displayed as the current angular value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progress bar</td>
<td>Pointer to radial progress bar control block</td>
</tr>
<tr>
<td>name</td>
<td>Name of radial progress bar</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>info</td>
<td>Pointer to a GX_RADIAL_PROGRESS_BAR structure. Appendix I contains definition for GX_RADIAL_PROGRESS_BAR structure.</td>
</tr>
<tr>
<td>style</td>
<td>Style of radial progress bar</td>
</tr>
<tr>
<td>id</td>
<td>Application-defined ID of progress bar</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful radial progress bar create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13)</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Invalid parent widget</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
GX_RAIDAL_PROGRESS_BAR_INFO info;

info.gx_radial_progress_bar_info_xcenter = 200;
info.gx_radial_progress_bar_info_ycenter = 200;
info.gx_radial_progress_bar_info_radius = 100;
info.gx_radial_progress_bar_info_current_angle = 180;
info.gx_radial_progress_bar_info_anchor_val = -180;
info.gx_radial_progress_bar_info_font_id = GX_FONT_ID_SYSTEM;
info.gx_radial_progress_bar_info_normal_text_color =
    GX_COLOR_ID_TEXT;
info.gx_radial_progress_bar_info_selected_text_color =
    GX_COLOR_ID_TEXT;
info.gx_radial_progress_bar_info_disabled_text_color =
    GX_COLOR_ID_DISABLED_TEXT;
info.gx_radial_progress_bar_info_normal_brush_width = 20;
info.gx_radial_progress_bar_info_selected_brush_width = 16;
info.gx_radial_progress_bar_info_normal_brush_color =
    GX_COLOR_ID_WIDGET_FILL;
info.gx_radial_progress_bar_info_selected_brush_color =
    GX_COLOR_ID_SELECTED_FILL;

/* Create a radial progress bar "my_progress_bar". */
status = gx_radial_progress_bar_create(&my_progress_bar,
    "my_progress_bar", parent, &info,
    GX_STYLE_ENABLED | GX_STYLE_TRANSPARENT |
    GX_STYLE_PROGRESS_TEXT_DRAW,
    ID_MY_RADIAL_PROGRESS);

/* If status is GX_SUCCESS the radial progress bar
"my_progress_bar" has been created. */
```

See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw,
gx_radial_progress_bar_draw, gx_radial_progress_bar_event_process,
gx_radial_progress_bar_font_set, gx_radial_progress_bar_info_set,
gx_radial_progress_bar_text_color_set, gx_radial_progress_bar_text_draw,
gx_radial_progress_bar_value_set
**gx_radial_progress_bar_draw**

Draw a radial progress bar

### Prototype

```
VOID gx_radial_progress_bar_draw(
   GX_RADIAL_PROGRESS_BAR *progress_bar);
```

### Description

This service draws a radial progress bar. This service is used internally referenced by the `gx_radial_progress_bar_create` function, but is exposed for use by the application in those cases where the application defines a custom radial progress bar drawing function.

### Parameters

- **progress bar**
  Pointer to radial progress bar control block

### Return Values

None

### Allowed From

Threads

### Example

```
/* Write a custom radial progress bar drawing function. */

VOID my_radial_progress_bar_draw(GX_RADIAL_PROGRESS_BAR *radial_progress)
{
    /* Call default radial progress bar draw. */
    gx_radial_progress_bar_draw(radial_progress);

    /* Add your own drawing here. */

    /* Draw child widgets. */
    gx_widget_children_draw((GX_WIDGET *)radial_progress);
}
```

### See Also

- `gx_radial_progress_bar_anchor_set`, `gx_radial_progress_bar_background_draw`,
- `gx_radial_progress_bar_create`, `gx_radial_progress_bar_event_process`,
- `gx_radial_progress_bar_font_set`, `gx_radial_progress_bar_info_set`,
- `gx_radial_progress_bar_size_calculate`, `gx_radial_progress_bar_text_color_set`,
- `gx_radial_progress_bar_text_draw`, `gx_radial_progress_bar_value_set`
**gx_radial_progress_bar_event_process**  
Process radial progress bar event

**Prototype**

```c
UINT gx_radial_progress_bar_event_process(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_EVENT *event_ptr);
```

**Description**

This service processes a radial progress bar event. This function is internally referenced by the `gx_radial_progress_bar_create` function, but is exposed for use by the application in those cases where the application defines a custom radial progress event processing function.

**Parameters**

- **progress_bar** Point to radial progress bar control block
- **event_ptr** Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) Successful radial progress bar event process
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Invalid widget

**Allowed From**

Initialization and threads
Example

/* Write a custom event processing function. */

UINT my_event_process (GX_RADIAL_PROGRESS_BAR *radial_progress,
GX_EVENT *event_ptr)
{
    switch(event_ptr->gx_event_type)
    {
    case GX_EVENT_SHOW:
        /* Do default handling. */
        status = gx_radial_progress_bar_event_process(
            radial_progress, event_ptr);

        /* add my own handling here */
        break;
    default:
        status = gx_radial_progress_bar_event_process(
            radial_progress, event_ptr);
        break;
    }

    return status;
}

See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw,
gx_radial_progress_bar_create, gx_radial_progress_bar_draw,
gx_radial_progress_bar_font_set, gx_radial_progress_bar_info_set,
gx_radial_progress_bar_text_color_set, gx_radial_progress_bar_text_draw,
gx_radial_progress_bar_value_set
**gx_radial_progress_bar_font_set**

Set radial progress bar font

### Prototype

```c
UINT gx_radial_progress_bar_font_set(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_RESOURCE_ID font_id);
```

### Description

This service sets the font of a radial progress bar widget. This parameter has no effect if the widget style `GX_STYLE_PROGRESS_TEXT_DRAW` is not set.

### Parameters

- **progress bar**: Pointer to radial progress bar control block
- **font_id**: Resource ID of font

### Return Values

- **GX_SUCCESS**: (0x00) Successful radial progress bar font set
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

### Allowed From

Initialization and threads

### Example

```c
/* Set font for radial progress bar "my_progress_bar". */
status = gx_radial_progress_bar_font_set(&my_progress_bar, font);

/* If status is GX_SUCCESS the font of "my_progress_bar" has been set. */
```

### See Also

- `gx_radial_progress_bar_anchor_set`
- `gx_radial_progress_bar_background_draw`
- `gx_radial_progress_bar_create`
- `gx_radial_progress_bar_draw`
- `gx_radial_progress_bar_event_process`
- `gx_radial_progress_bar_info_set`
- `gx_radial_progress_bar_text_color_set`
- `gx_radial_progress_bar_text_draw`
- `gx_radial_progress_bar_value_set`
**gx_radial_progress_bar_info_set**

Set radial progress bar information

**Prototype**

```c
UINT gx_radial_progress_bar_info_set(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_RADIAL_PROGRESS_BAR_INFO *info);
```

**Description**

This service resets the information parameters assigned to the radial progress bar.

**Parameters**

- **progress bar**: Pointer to radial progress bar control block.
- **info**: Pointer to radial progress bar information structure. Appendix I contains definition for GX_RADIAL_PROGRESS_BAR_INFO structure.

**Return Values**

- **GX_SUCCESS**: (0x00) Successful radial progress bar info set
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

**Allowed From**

Initialization and threads
Example

GX_RAIDAL_PROGRESS_BAR_INFO info;

info.gx_radial_progress_bar_info_xcenter = 200;
info.gx_radial_progress_bar_info_ycenter = 200;
info.gx_radial_progress_bar_info_radius = 100;
info.gx_radial_progress_bar_info_current_angle = 180;
info.gx_radial_progress_bar_info_anchor_val = -180;
info.gx_radial_progress_bar_info_font_id = GX_FONT_ID_SYSTEM;
info.gx_radial_progress_bar_info_normal_text_color =
  GX_COLOR_ID_TEXT;
info.gx_radial_progress_bar_info_selected_text_color =
  GX_COLOR_ID_TEXT;
info.gx_radial_progress_bar_info_disabled_text_color =
  GX_COLOR_ID_DISABLED_TEXT;
info.gx_radial_progress_bar_info_normal_brush_width = 20;
info.gx_radial_progress_bar_info_selected_brush_width = 16;
info.gx_radial_progress_bar_info_normal_brush_color =
  GX_COLOR_ID_WIDGET_FILL;
info.gx_radial_progress_bar_info_selected_brush_color =
  GX_COLOR_ID_SELECTED_FILL;

/* Set appearance information for radial progress bar
   "my_progress_bar". */
status = gx_radial_progress_bar_info_set(&my_progress_bar, &info);

/* If status is GX_SUCCESS the appearance information of
   "my_progress_bar" has been set. */

See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw,
gx_radial_progress_bar_create, gx_radial_progress_bar_draw,
gx_radial_progress_bar_event_process, gx_radial_progress_bar_font_set,
gx_radial_progress_bar_text_color_set, gx_radial_progress_bar_text_draw,
gx_radial_progress_bar_value_set
**gx_radial_progress_bar_text_color_set**

Set radial progress bar text color

### Prototype

```c
UINT gx_radial_progress_bar_text_color_set(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_RESOURCE_ID normal_text_color,
    GX_RESOURCE_ID selected_text_color,
    GX_RESOURCE_ID disabled_text_color);
```

### Description

This service sets the text color of radial progress bar. This value is only used if the style GX_STYLE_PROGRESS_TEXT_DRAW is set.

### Parameters

- **progress bar** Pointer to radial progress bar control block
- **normal_color** Resource ID of text color in normal state. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **selected_color** Resource ID of text color when the widget gain focus. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **disabled_color** Resource ID of text color when the style GX_STYLE_ENABLED is not set. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

### Return Values

- **GX_SUCCESS** (0x00) Successful radial progress bar text color set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Invalid widget check
Allowed From

Initialization and threads

Example

/* Set text color for radial progress bar "my_progress_bar". */
status = gx_radial_progress_bar_text_color_set(&my_progress_bar,
   GX_COLOR_ID_NORMAL_TEXT,
   GX_COLOR_ID_SELECTED_TEXT,
   GX_COLOR_ID_DISABLED_TEXT);

/* If status is GX_SUCCESS the text color of "my_progress_bar" has been set. */

See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw,
gx_radial_progress_bar_create, gx_radial_progress_bar_draw,
gx_radial_progress_bar_event_process, gx_radial_progress_bar_font_set,
gx_radial_progress_bar_info_set, gx_radial_progress_bar_text_draw,
gx_radial_progress_bar_value_set
gx_radial_progress_bar_text_draw

Draw radial progress bar text

Prototype

VOID  gx_radial_progress_bar_text_draw(
       GX_RADIAL_PROGRESS_BAR  *progress_bar)

Description

This service draws the text of specified radial progress bar. This function is called internally as part of the gx_radial_progress_bar_draw(), but is exposed to the application to support those cases where the application defines a custom progress bar drawing function.

Parameters

progress bar  Pointer to radial progress bar control block

Return Values

None

Allowed From

Initialization and Threads

Example

/* Draw text for radial progress bar “my_progress_bar”. */
status = gx_radial_progress_bar_text_draw(&my_progress_bar);

/* If status is GX_SUCCESS the text of “my_progress_bar” has been drawn. */

/* Write a custom radial progress bar drawing function. */

VOID my_radial_progress_bar_draw(GX_RADIAL_PROGRESS_BAR *radial_progress)
{
   /* Add your own background draw here. */

   /* Call default radial progress bar text draw. */
gx_radial_progress_bar_text_draw(radial_progress);

   /* Draw child widgets. */
gx_widget_children_draw((GX_WIDGET *)radial_progress);
}
See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw,
gx_radial_progress_bar_create, gx_radial_progress_bar_draw,
gx_radial_progress_bar_event_process, gx_radial_progress_bar_font_set,
gx_radial_progress_bar_info_set, gx_radial_progress_bar_text_color_set,
gx_radial_progress_bar_value_set
gx_radial_progress_bar_value_set

Set radial progress bar value

Prototype

UINT gx_radial_progress_bar_value_set(
    GX_RADIAL_PROGRESS_BAR *progress_bar,
    GX_VALUE value);

Description

This service sets radial progress bar value. The assigned value is limited to the range [-360, 360], defining the possible range of angular values for the progress bar current location. The application must scale the real-world value being indicated to assign an angular value to the progress bar widget.

The progress bar is drawn such that the current value indicates the angular delta between the anchor position and the end point of the upper arc. Negative values cause the arc to be drawn in a clockwise direction starting at the anchor position. Positive current value causes the arc to be drawn in a counter-clockwise direction starting at the anchor position.

For example, to draw an arc starting at the top of the arc (12 o’clock position) and ending at the right (3 o’clock position), assign an anchor value of 90 degrees and a current value of -90 degrees.

Parameters

- **progress bar**: Pointer to radial progress bar control block
- **value**: New progress bar value

Return Values

- **GX_SUCCESS**: (0x00) Successful radial progress bar value set
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointers
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

Allowed From

Initialization and threads
Example

GX_VALUE new_value = 180;

/* Set value for radial progress bar "my_progress_bar". */
status = gx_radial_progress_bar_value_set(&my_progress_bar, 
    new_value);

/* If status is GX_SUCCESS the value of "my_progress_bar" has been set. */

See Also

gx_radial_progress_bar_anchor_set, gx_radial_progress_bar_background_draw, 
gx_radial_progress_bar_create, gx_radial_progress_bar_draw, 
gx_radial_progress_bar_event_process, gx_radial_progress_bar_font_set, 
gx_radial_progress_bar_info_set, gx_radial_progress_bar_text_color_set, 
gx_radial_progress_bar_text_draw
**gx_radio_button_create**

Create radio button

**Prototype**

```c
UINT gx_radio_button_create(GX_RADIO_BUTTON *button,
                            GX_CONST GX_CHAR *name,
                            GX_WIDGET *parent,
                            GX_RESOURCE_ID text_id, ULONG style,
                            USHORT radio_button_id,
                            GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a radio button widget. GX_RADIO_BUTTON is derived from GX_TEXT_BUTTON, and therefore all gx_text_button services are also supported by this widget type.

**Parameters**

- **button**: Pointer to radio button control block
- **name**: Logical name of radio button widget
- **parent**: Pointer to the parent widget
- **text_id**: Resource ID of radio button
- **style**: Style of radio button. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
- **radio_button_id**: Application-defined ID of radio button
- **size**: Dimensions of radio button

**Return Values**

- **GX_SUCCESS** (0x00): Successful radio button create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY_CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
- **GX_INVALID_RESOURCE_ID** (0x33): Invalid resource ID
- **GX_INVALID_WIDGET** (0x12): Invalid widget

**Allowed From**

Initialization and threads
Example

/* Create "my_radio_button". */
status = gx_radio_button_create(&my_radio_button,
    "my_radio_button", &my_parent,
    MY_RADIO_BUTTON_TEXT_RESOURCE_ID,
    GX_STYLE_BORDER_RAISED, MY_RADIO_BUTTON_ID, &size);

/* If status is GX_SUCCESS the radio button "my_radio_button" has been created. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_icon_button_create, gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw,
gx_radio_button_draw
**gx_radio_button_draw**

**Draw radio button**

**Prototype**

```c
VOID gx_radio_button_draw(GX_RADIO_BUTTON *button);
```

**Description**

This service draws a radio button widget. This service is called internally by the GUIX canvas refresh, but can also be called by overridden drawing functions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>button</code></td>
<td>Pointer to radio button widget control block</td>
</tr>
</tbody>
</table>

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom radio button drawing function. */

VOID my_radio_button_draw(GX_RADIO_BUTTON *radio_button)
{
    /* Call default radio button draw. */
    gx_radio_button_draw(radio_button);

    /* Add your own drawing here. */

    /* Draw child widgets. */
    gx_widget_children_draw((GX_WIDGET *)radio_button);
}
```

**See Also**

- `gx_button_background_draw`
- `gx_button_create`
- `gx_button_deselect`
- `gx_button_draw`
- `gx_button_event_process`
- `gx_button_select`
- `gx_icon_button_create`
- `gx_pixelmap_button_create`
- `gx_pixelmap_button_draw`
- `gx_text_button_create`
- `gx_text_button_color_set`
- `gx_text_button_draw`
- `gx_radio_button_create`
gx_radio_button_pixelmap_set

Set pixelmaps for radio button

Prototype

UINT  gx_radio_button_pixelmap_set(GX_RADIO_BUTTON *button,
GX_RESOURCE_ID off_id,
GX_RESOURCE_ID on_id,
GX_RESOURCE_ID off_disabled_id,
GX_RESOURCE_ID on_disabled_id);

Description

This service assigns the pixelmaps to be displayed by the specified radio button for each button state. The resource IDs can be duplicated.

Parameters

- **button**: Pointer to radio button widget control block
- **off_id**: Pixelmap used for radio button off state
- **on_id**: Pixelmap used for radio button on state
- **off_disabled_id**: Pixelmap used for radio button disabled and off state
- **on_disabled_id**: Pixelmap used for radio button disabled and on state

Return Values

- **GX_SUCCESS**: (0x00) Successful radio button pixelmaps set
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

Allowed From

Initialization and threads
Example

/* Set pixelmap for "my_radio_button". */
status = gx_radio_button_pixelmap_set(&my_radio_button,
    MY_OFF_PIXELMAP, MY_ON_PIXELMAP,
    MY_OFF_DISABLED_PIXELMAP,
    MY_ON_DISABLED_PIXELMAP);

/* If status is GX_SUCCESS the pixelmaps for radio button
"my_radio_button" has been set. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_icon_button_create, gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_text_button_create, gx_text_button_color_set, gx_text_button_draw,
gx_radio_button_create
gx_radial_slider_anchor_angles_set

Set radial slider anchor list

Prototype

UINT  gx_radial_slider_anchor_angles_set (GX_RADIAL_SLIDER *slider,
                                      GX_VALUE *anchor_angles, USHORT anchor_count);

Description

This service sets anchor angles for radial slider. If anchor angle list
is set, the radial slider angle will be one of defined anchor angles.

Parameters

- slider: Radial slider control block
- anchor_angles: Angle list to set
- anchor_count: Count of the anchor angles

Return Values

- GX_SUCCESS (0x00): Successful anchor angles set
- GX_PTR_ERROR (0x07): Invalid pointer
- GX_INVALID_WIDGET (0x12): Invalid widget
- GX_INVALID_VALUE (0x22): Invalid anchor list

Allowed From

Initialization and threads
Example

GX_VALUE anchor_angles[]={0, 30, 60, 90, 120, 150, 180};
USHORT anchor_count = 7;

/* Set anchor angles for radial slider. */
status = gx_radial_slider_anchor_angles_set(&my_radial_slider,
anchor_angles, anchor_count);

/* If status is GX_SUCCESS the anchor angles have been set for
"my_radial_slider". */

/* Set anchor angles for radial slider. */
status = gx_radial_slider_anchor_angles_set(&my_radial_slider,
GX_NULL, 0);

/* If status is GX_SUCCESS the anchor angles have been removed from
"my_radial_slider". */

See Also

gx_radial_slider_angle_set, gx_radial_slider_animation_set,
gx_radial_slider_animation_start, gx_radial_slider_create, gx_radial_slider_draw,
gx_radial_slider_event_process, gx_radial_slider_info_get,
gx_radial_slider_info_set, gx_radial_slider_pixelmap_set
gx_radial_slider_angle_set

Set radial slider angle

Prototype

UINT gx_radial_slider_angle_set(GX_RADIAL_SLIDER *slider,
                               GX_VALUE new_angle);

Description

This service sets new angle value for radial slider.

Parameters

- slider: Pointer to radial slider control block
- new_angle: New angle value to be set

Return Values

- GX_SUCCESS (0x00): Successful radial slider angle set
- GX_CALLER_ERROR (0x11): Invalid caller of this function
- GX_PTR_ERROR (0x07): Invalid pointer
- GX_INVALID_WIDGET (0x12): Invalid widget

Allowed From

Initialization and threads

Example

/* Set “my_radial_slider” angle to 0 degree(3 o’clock position). */
status = gx_radial_slider_angle_set(&my_radial_slider, 0);

/* If status is GX_SUCCESS the value of “my_radial_slider” has been
set to 0 degree. */

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_animation_set,
gx_radial_slider_animation_start, gx_radial_slider_create, gx_radial_slider_draw,
gx_radial_slider_event_process, gx_radial_slider_info_get,
gx_radial_slider_info_set, gx_radial_slider_pixelmap_set
gx_radial_slider_animation_set
Create radial slider animation info

Prototype

UINT  gx_radial_slider_animation_set(GX_RADIAL_SLIDER *slider,
USHORT steps, USHORT delay, USHORT animation_style,
VOID (*animation_update_callback)(GX_RADIAL_SLIDER *slider));

Description

This service sets animation steps, delay time and animation styles
for radial slider needle animation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>slider</td>
<td>Pointer to radial slider control block</td>
<td></td>
</tr>
<tr>
<td>steps</td>
<td>Total steps for one animation</td>
<td></td>
</tr>
<tr>
<td>delay</td>
<td>Delay time for each animation step</td>
<td></td>
</tr>
<tr>
<td>animation_style</td>
<td>Easing function type, includes:</td>
<td>GX_ANIMATION_BACK_EASE_IN, GX_ANIMATION_BACK_EASE_OUT, G...</td>
</tr>
</tbody>
</table>
animation_update_callback

User define callback function that will be called after each animation step

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful radial slider animation set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET (0x12)</td>
<td>Invalid widget</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
VOID animation_callback(GX_RADIAL_SLIDER *slider)
{
    /* This function will be called after each animation step,
       add custom code here. */
}

/* Set animation info for "my_radial_slider". */
status = gx_radial_slider_animation_set(&my_radial_slider, 15, 2, 
                                       GX_ANIMATION_CIRC_EASE_IN_OUT, 
                                       animation_callback);

/* If status is GX_SUCCESS the radial slider needle will move with 
   specified animation. */

/* Disable animation for "my_radial_slider". */
status = gx_radial_slider_animation_set(&my_radial_slider, 0, 0, 
                                       0, 0);

/* If status is GX_SUCCESS the radial slider needle will move 
   without animation. */
```

See Also

- `gx_radial_slider_anchor_angles_set`
- `gx_radial_slider_angle_set`
- `gx_radial_slider_animation_start`
- `gx_radial_slider_create`
- `gx_radial_slider_draw`
- `gx_radial_slider_event_process`
- `gx_radial_slider_info_get`
- `gx_radial_slider_info_set`
- `gx_radial_slider_pixelmap_set`
gx_radial_slider_animation_start

Set new radial slider value with animation

Prototype

```c
UINT gx_radial_slider_animation_start(GX_RADIAL_SLIDER *slider, 
                                       GX_VALUE target_angle);
```

Description

This service starts an animation to move the slider needle from current position to the specified position.

Parameters

- `slider` Pointer to radial slider control block
- `target_angle` Target angle value

Return Values

- `GX_SUCCESS` (0x00) Successful radial slider animation start
- `GX_CALLER_ERROR` (0x11) Invalid caller of this function
- `GX_PTR_ERROR` (0x07) Invalid pointer
- `GX_INVALID_WIDGET` (0x12) Invalid widget

Allowed From

Initialization and threads

Example

```c
/* Start an animation to move radial slider needle from current position to 90 degree position. */
status = gx_radial_slider_animation_start(&my_radial_slider, 90);

/* If status is GX_SUCCESS the radial slider needle animation has been started. */
```

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set, 
gx_radial_slider_animation_set, gx_radial_slider_create, gx_radial_slider_draw, 
gx_radial_slider_event_process, gx_radial_slider_info_get, 
gx_radial_slider_info_set, gx_radial_slider_pixelmap_set
**gx_radial_slider_create**

Create radial slider

**Prototype**

```c
UINT gx_radial_slider_create(GX_RADIAL_SLIDER *slider,
                             GX_CONST GX_CHAR *name,
                             GX_WIDGET *parent,
                             GX_RADIAL_SLIDER_INFO *info,
                             ULONG style,
                             USHORT slider_id,
                             GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a radial slider widget.

**Parameters**

- **slider**
  Pointer to radial slider control block
- **name**
  Logical name of radial slider widget
- **parent**
  Pointer to the parent widget
- **info**
  Radial slider appearance definition, Appendix I contains definition to GX_RADIAL_SLIDER_INFO.
- **style**
  Style of radio button. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **radio_button_id**
  Application-defined ID of radial slider
- **size**
  Dimensions of radial slider

**Return Values**

- **GX_SUCCESS** (0x00) Successful radial slider create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12) Invalid parent widget

**Allowed From**

Initialization and threads
**Example**

```c
GX_RADIAL_SLIDER_INFO info;
GX_RECTANGLE size;

/* Distance from left side of widget to rotating center. */
info.gx_radial_slider_info_xcenter = 100;

/* Distance from top size of widget to rotating center. */
info.gx_radial_slider_info_ycenter = 100;

/* Radius of rotating circle. */
info.gx_radial_slider_info_radius = 100;

/* Widget of rotating track. */
info.gx_radial_slider_info_track_width = 40;

/* Current angle value. */
info.gx_radial_slider_info_current_angle = 0;

/* Minimum angle value. */
info.gx_radial_slider_min_angle = -60;

/* Maximum angle value. */
info.gx_radial_slider_max_angle = 240;

/* Anchor value list. */
info.gx_radial_slider_angle_list = GX_NULL;

/* Anchor value count. */
info.gx_radial_slider_list_count = 0;

/* Resource ID of background pixelmap. */
info.gx_radial_slider_background_pixelmap = GX_PIXELMAP_ID_BKGRD;

/* Resource ID of needle pixelmap. */
info.gx_radial_slider_needle_pixelmap = GX_PIXELMAP_ID_NEEDLE;

/* Define widget size. */
gx_utility_rectangle_define(&size, 0, 0, 200, 200);

/* Create "my_radial_slider". */
status = gx_radial_slider_create(&my_radial_slider,
                               "my_radial_slider", &my_parent,
                               &info, GX_STYLE_ENABLED,
                               MY_RADIAL_SLIDER_ID, &size);

/* If status is GX_SUCCESS the radial slider "my_radial_slider" has been created. */
```

**See Also**

- `gx_radial_slider_anchor_angles_set`
- `gx_radial_slider_angle_set`
- `gx_radial_slider_animation_set`
- `gx_radial_slider_animation_start`
- `gx_radial_slider_draw`
- `gx_radial_slider_event_process`
- `gx_radial_slider_info_get`
- `gx_radial_slider_info_set`
- `gx_radial_slider_pixelmap_set`
**gx_radial_slider_draw**

**Draw radial slider**

**Prototype**

VOID  **gx_radial_slider_draw**(GX_RADIAL_SLIDER *slider);

**Description**

This service draws a radial slider. This service is called internally by the GUIX canvas refresh, but can also be called by overridden drawing functions.

**Parameters**

- **slider**
  Pointer to radial slider control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom radio button drawing function. */

VOID my_radial_slider_draw(GX_RADIAL_SLIDER *radial_slider)
{
    /* Call default radial slider draw. */
    gx_radio_slider_draw(radial_slider);

    /* Add your own drawing here. */
}
```

**See Also**

- gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set,
- gx_radial_slider_animation_set, gx_radial_slider_animation_start,
- gx_radial_slider_create, gx_radial_slider_draw, gx_radial_slider_event_process,
- gx_radial_slider_info_get, gx_radial_slider_info_set,
- gx_radial_slider_pixelmap_set
gx_radial_slider_event_process

Process radial slider event

Prototype

UINT gx_radial_slider_event_process(GX_RADIAL_SLIDER *slider,
                                 GX_EVENT *event_ptr);

Description

This service processes a radial slider event. This service should be
called as the default event handler by any custom radial slider event
processing functions.

Parameters

slider Pointer to radial slider control block
event_ptr Pointer to event to process

Return Values

GX_SUCCESS (0x00) Successful radial slider event process
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Invalid widget

Allowed From

Initialization and threads
Example

/* Write a custom event processing function. */

UINT my_event_process(GX_RADIAL_SLIDER *slider,
                      GX_EVENT *event_ptr)
{
    switch(event_ptr->gx_event_type)
    {
        case GX_EVENT_SHOW:
            /* Do default handling. */
            status = gx_radial_slider_event_process(slider, event_ptr);
            /* add my own handling here */
            break;
        default:
            status = gx_radial_slider_event_process(slider, event_ptr);
            break;
    }
    return status;
}

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set,
gx_radial_slider_animation_set, gx_radial_slider_animation_start,
gx_radial_slider_create, gx_radial_slider_draw, gx_radial_slider_info_get,
gx_radial_slider_info_set, gx_radial_slider_pixelmap_set
gx_radial_slider_info_get

Retrieve radial slider info

Prototype

UINT gx_radial_slider_info_get(GX_RADIAL_SLIDER *slider,
                                GX_RADIAL_SLIDER_INFO **info);

Description

This service retrieves radial slider information pointer.

Parameters

slider Pointer to radial slider control block
info Retrieved radial slider information pointer

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful radial slider info</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Invalid widget</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_RADIAL_SLIDER_INFO *info;

/* Retrive radial slider information structure. */
status = gx_radial_slider_info_get(&my_radial_slider,
                                   "my_radial_slider", &info);

/* If status is GX_SUCCESS the radial slider information pointer of
"my_radial_slider" has been retrieved. */

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set,
gx_radial_slider_animation_set, gx_radial_slider_animation_start,
gx_radial_slider_create, gx_radial_slider_draw, gx_radial_slider_event_process,
gx_radial_slider_info_set, gx_radial_slider_pixelmap_set
gx_radial_slider_info_set

Prototype

UINT gx_radial_slider_info_set(GX_RADIAL_SLIDER *slider,
                               GX_RADIAL_SLIDER_INFO *info);

Description

This service sets radial slider information.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slider</td>
<td>Pointer to radial slider control block</td>
</tr>
<tr>
<td>info</td>
<td>Radial slider information to set</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful radial slider info set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Invalid widget

Allowed From

Initialization and threads
Example

GX_RADIAL_SLIDER_INFO info;
/* Distance from left side of widget to rotating center. */
info.gx_radial_slider_info_xcenter = 100;

/* Distance from top size of widget to rotating center. */
info.gx_radial_slider_info_ycenter = 100;

/* Radius of rotating circle. */
info.gx_radial_slider_info_radius = 100;

/* Widget of rotating track. */
info.gx_radial_slider_info_track_width = 40;

/* Current angle value. */
info.gx_radial_slider_info_current_angle = 0;

/* Minimum angle value. */
info.gx_radial_slider_min_angle = -60;

/* Maximum angle value. */
info.gx_radial_slider_max_angle = 240;

/* Anchor value list. */
info.gx_radial_slider_angle_list = GX_NULL;

/* Anchor value count. */
info.gx_radial_slider_list_count = 0;

/* Resource ID of background pixmap. */
info.gx_radial_slider_background_pixelmap = GX_PIXELMAP_ID_BKGRD;

/* Resource ID of needle pixmap. */
info.gx_radial_slider_needle_pixelmap = GX_PIXELMAP_ID_NEEDLE;

/* Reset radial slider info for "my_radial_slider". */
status = gx_radial_slider_info_set(&my_radial_slider, &info);

/* If status is GX_SUCCESS the radial slider info of 
"my_radial_slider" has been reset. */

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set,
gx_radial_slider_animation_set, gx_radial_slider_animation_start,
gx_radial_slider_create, gx_radial_slider_draw, gx_radial_slider_event_process,
gx_radial_slider_info_get, gx_radial_slider_pixelmap_set
gx_radial_slider_pixelmap_set

Set radial slider pixelmaps

Prototype

UINT  gx_radial_slider_pixelmap_set(GX_RADIAL_SLIDER *slider,
                                      GX_RESOURCE_ID background_pixelmap,
                                      GX_REOUSRCE_ID needle_pixelmap);

Description

This service sets radial slider background and needle pixelmaps.

Parameters

slider Pointer to radial slider control block
background_pixelmap Resource ID of background pixelmap
needle_pixelmap Resource ID of needle pixelmap

Return Values

GX_SUCCESS (0x00) Successful radial slider pixelmap set
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Invalid widget

Allowed From

Initialization and threads

Example

/* Create “my_radio_button”. */
status = gx_radial_slider_pixelmap_set(&my_radial_slider,
                                       GX_PIXELMAP_ID_BG,
                                       GX_PIXELMAP_ID_NEEDLE);

/* If status is GX_SUCCESS the background and needle pixelmap of “my_radial_slider” has been reset. */

See Also

gx_radial_slider_anchor_angles_set, gx_radial_slider_angle_set,
gx_radial_slider_animation_set, gx_radial_slider_animation_start,
gx_radial_slider_create, gx_radial_slider_draw, gx_radial_slider_event_process,
gx_radial_slider_info_get, gx_radial_slider_info_set
**gx_screen_stack_create**

Initialize a screen stack

**Prototype**

```c
UINT gx_screen_stack_create(GX_SCREEN_STACK_CONTROL *control,
                              GX_WIDGET **memory_buffer,
                              INT buffer_size);
```

**Description**

This service initializes a screen stack. The application must define the memory block and buffer size used to implement the screen stack feature.

*Note: This API is obsoleted, and is replaced with* `gx_system_screen_stack_create()`. This version is provided only for backwards compatibility with previous library releases.

**Parameters**

- **control**: Screen stack control block
- **memory_buffer**: Pointer to a memory buffer that used as a screen stack
- **buffer_size**: Memory size in bytes

**Return Values**

- **GX_SUCCESS** (0x00) : Successful screen stack create
- **GX_PTR_ERROR** (0x07) : Invalid pointer
- **GX_INVALID_VALUE** (0x22) : Invalid buffer size

**Allowed From**

Initialization and threads
Example

```c
#define SCREEN_STACK_SIZE 10

GX_SCREEN_STACK_CONTROL my_stack_control;
GX_WIDGET *screen_stack[SCREEN_STACK_SIZE];

/* Initialize "my_stack_control". */
status = gx_screen_stack_create(&my_stack_control, screen_stack,
                               SCREEN_STACK_SIZE * sizeof(GX_WIDGET *)));

/* If status is GX_SUCCESS the screen control block "my_stack control" has been initialized. */
```

See Also

`gx_screen_stack_push`, `gx_screen_stack_pop`, `gx_screen_stack_reset`
**gx_screen_stack_pop**

Remove the topmost entry from the screen stack

Prototype

```c
UINT gx_screen_stack_pop(GX_SCREEN_STACK_CONTROL *control);
```

Description

This service removes the topmost entry from the screen stack, and attaches the popped screen to its previous parent. This API also detaches any existing children from the parent.

*Note: This API is obsoleted, and is replaced with gx_system_screen_stack_pop(). This version is provided only for backwards compatibility with previous library releases.*

Parameters

- **control**
  Screen stack control block

Return Values

- **GX_SUCCESS** (0x00) Successful screen stack pop
- **GX CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

- Threads

Example

```c
/* Remove the topmost entry from the screen stack. */
status = gx_screen_stack_pop(&my_stack_control);

/* If status is GX_SUCCESS the topmost entry has been removed from
the screen stack, and the popped screen has been attached to its
parent. */
```

See Also

- `gx_screen_stack_create`, `gx_screen_stack_push`, `gx_screen_stack_reset`
**gx_screen_stack_push**

Push screen and its parents to stack

**Prototype**

```
UINT  gx_screen_stack_push(GX_SCREEN_STACK_CONTROL *control,
                               GX_WIDGET *screen,
                               GX_WIDGET *new_screen);
```

**Description**

This service detaches screen from its parent, and pushes the screen pointer and the parent pointer onto the screen stack. The new screen pointer is then attached to the parent.

*Note: This API is obsoleted, and is replaced with gx_system_screen_stack_pop(). This version is provided only for backwards compatibility with previous library releases.*

**Parameters**

- **control**: Screen stack control block
- **screen**: Screen pointer to push
- **new_screen**: Pointer of the new screen

**Return Values**

- **GX_SUCCESS** (0x00) Successful screen stack push
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Threads
Example

/* Push “screen” and its parent to screen stack, Detach “screen” from its parent, and attach “new screen” to the parent. */
status = gx_screen_stack_push(&my_stack_control,
                                screen, new_screen);

/* If status is GX_SUCCESS the widget “screen” and its parent have been pushed to screen stack, “screen” has been detached from its parent, “new_screen” has been attached to the parent. */

See Also

gx_screen_stack_create, gx_screen_stack_push, gx_screen_stack_reset
**gx_screen_stack_reset**

Removes all entries from the screen stack

**Prototype**

```c
UINT  gx_screen_stack_reset(GX_SCREEN_STACK_CONTROL *control);
```

**Description**

This service removes all entries from the screen stack.

**Note:** This API is obsoleted, and is replaced with `gx_system_screen_stack_pop()`. This version is provided only for backwards compatibility with previous library releases.

**Parameters**

- `control` Screen stack control block

**Return Values**

- `GX_SUCCESS` (0x00) Successful scroll thumb create
- `GX_PTR_ERROR` (0x07) Invalid pointer

**Allowed From**

Threads

**Example**

```c
/* Remove all entries from the screen stack. */
status = gx_screen_stack_reset(&my_stack_control);

/* If status is GX_SUCCESS all entries of screen stack has been removed. */
```

**See Also**

`gx_screen_stack_create`, `gx_screen_stack_push`, `gx_screen_stack_pop`
**gx_scroll_thumb_create**

Create scroll thumb

**Prototype**

```c
UINT gx_scroll_thumb_create(GX_SCROLL_THUMB *scroll_thumb,
                           GX_SCROLLBAR *parent, ULONG style);
```

**Description**

This service creates a scroll thumbwheel. This service is normally called internally when a GX_SCROLLBAR is created, but is made public in order to allow custom scrollbar implementations.

**Parameters**

- **scroll_thumb**
  - Scroll thumb widget control block
- **parent**
  - Pointer to parent scrollbar
- **style**
  - Style of scrollbar widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.

**Return Values**

- **GX_SUCCESS** (0x00) Successful scroll thumb create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12) Parent widget not valid

**Allowed From**

Initialization and threads
Example

GX_SCROLL_THUMB my_scroll_thumb;

/* Create scroll thumb “my_scroll_thumb”. */
status = gx_scroll_thumb_create(&my_scroll_thumb, &my_scrollbar,
                                GX_STYLE_NONE);

/* If status is GX_SUCCESS the scroll thumb “my_scroll_thumb” has
been created. */

See Also

gx_scroll_thumb_draw, gx_scroll_thumb_event_process
**gx_scroll_thumb_draw**

**Draw scroll thumb**

**Prototype**

```c
VOID gx_scroll_thumb_draw(GX_SCROLL_THUMB *scroll_thumb);
```

**Description**

This service draws a scroll thumbwheel. This service is called internally by the GUIX canvas refresh, but can also be called by overridden drawing functions.

**Parameters**

- `scroll_thumb` Scroll thumb widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom scroll thumb drawing function. */

VOID my_scroll_thumb_draw(GX_SCROLL_THUMB *thumb)
{
    /* Call default scroll thumb draw. */
    gx_scroll_thumb_draw(thumb);

    /* Add your own drawing here. */
}
```

**See Also**

`gx_scroll_thumb_create`, `gx_scroll_thumb_event_process`
**gx_scroll_thumb_event_process**

Process scroll thumb event

**Prototype**

```c
UINT gx_scroll_thumb_event_process(GX_SCROLL_THUMB *scroll_thumb, 
                                   GX_EVENT *event);
```

**Description**

This service handles events sent to a scrollbar thumbwheel. This service is normally used internally by GUIX, but is made public to assist with implementing custom scrollbar behaviors.

**Parameters**

- **scroll_thumb**
  Scroll thumb widget control block
- **event**
  Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) Successful scroll thumb event process
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads
Example

/* Write a custom event processing function. */

UINT my_event_process (GX_SCROLL_THUMB *thumb, GX_EVENT *event_ptr)
{
    switch(event_ptr->gx_event_type)
    {
        case GX_EVENT_SHOW:
        /* Do default handling. */
            status = gx_scroll_thumb_event_process(thumb, event_ptr);
            /* add my own handling here */
            break;
        default:
            status = gx_scroll_thumb_event_process(thumb, event_ptr);
            break;
    }
    return status;
}

See Also

gx_scroll_thumb_create, gx_scroll_thumb_draw
gx_scroll_wheel_create

Create a base scroll wheel widget

Prototype

UINT gx_scroll_wheel_create( GX_SCROLL_WHEEL *wheel,
                          GX_CONST GX_CHAR *name,
                          GX_WIDGET *parent,
                          INT total_rows, ULONG style,
                          USHORT id,
                          GX_CONST GX_RECTANGLE *size);

Description

This service creates a generic scroll wheel widget.

A generic scroll wheel is the base widget for all scroll wheel widget
types, including the gx_text_scroll_wheel which is the base for
gx_numeric_scroll_wheel and gx_string_scroll_wheel widgets. The base
scroll wheel widget provides event handling, scrolling animation, and
selected row calculation for all scroll wheel widget types.

Applications would not normally create an instance of a generic
scroll wheel widget, since this widget type provides no drawing
function. However access to this API is provided to assist
applications which need to create a custom scroll wheel widget type.

GX_SCROLL_WHEEL is based on GX_WINDOW, and therefore all
GX_WINDOW APIs may be used with GX_SCROLL_WHEEL and
widgets derived from GX_SCROLL_WHEEL.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel</td>
<td>Pointer to generic scroll wheel control block</td>
</tr>
<tr>
<td>name</td>
<td>Application assigned widget name</td>
</tr>
<tr>
<td>parent</td>
<td>Parent widget, or GX_NULL</td>
</tr>
<tr>
<td>total_rows</td>
<td>Total available rows</td>
</tr>
<tr>
<td>style</td>
<td>Widget style flags</td>
</tr>
<tr>
<td>id</td>
<td>Application assigned widget ID</td>
</tr>
<tr>
<td>size</td>
<td>Rectangle defining initial widget size.</td>
</tr>
</tbody>
</table>

Return Values
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully created scroll wheel</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19) Invalid control block size</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget created created</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Parent widget not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
/* Call generic create function during custom widget createl. */

UINT custom_scroll_wheel_create(CUSTOM_SCROLL_WHEEL *wheel,
                                 GX_CONST GX_CHAR *name,
                                 GX_WIDGET *parent,
                                 INT total_rows, ULONG style,
                                 USHORT id,
                                 GX_CONST GX_RECTANGLE *size)
{
    /* create base widget as part of custom create */
    status = gx_scroll_wheel_create(wheel, name, parent,
                                     total_rows, style, id, size);

    /* If status is GX_SUCCESS the base scroll wheel has been
     * created. */
}
```

**See Also**

- `gx_numeric_scroll_wheel_create`, `gx_numeric_scroll_wheel_range_set`,
- `gx_scroll_wheel_event_process`, `gx_scroll_wheel_gradient_alpha_set`,
- `gx_scroll_wheel_row_height_set`, `gx_scroll_wheel_selected_background_set`,
- `gx_scroll_wheel_selected_get`, `gx_scroll_wheel_selected_set`,
- `gx_scroll_wheel_speed_set`, `gx_scroll_wheel_total_rows_set`,
- `gx_text_scroll_wheel_callback_set`, `gx_text_scroll_wheel_create`,
- `gx_text_scroll_wheel_draw`, `gx_text_scroll_wheel_font_set`,
- `gx_text_scroll_wheel_text_color_set`, `gx_string_scroll_wheel_create`,
- `gx_string_scroll_wheel_text_get`
gx_scroll_wheel_event_process

Event processing function for generic scroll wheel widget

Prototype

UINT gx_scroll_wheel_event_process(GX_SCROLL_WHEEL *wheel,
                        GX_EVENT *event);

Description

This service provides the basic input event handling for all scroll wheel widget types.

This function is exposed to the application software to assist with applications which need to create a custom scroll wheel widget type. Applications would often provide their own event processing function, but invoke the generic event processing for wheel widgets for events that they do not need to customize.

Parameters

- **wheel**: Pointer to generic scroll wheel control block
- **event**: GX_EVENT pointer

Return Values

- **GX_SUCCESS** (0x00): Successfully scroll wheel event process
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads
Example

/* Call generic scroll wheel event processing as part of custom event processing function. */

UINT custom_scroll_wheel_event_process(CUSTOM_SCROLL_WHEEL *wheel,
                                       GX_EVENT *event)
{
    switch(event->gx_event_type)
    {
    case xyz:
        /* insert custom event handling here */
        break;

    default:
        /* pass all other events to the generic scroll wheel event processing */
        gx_scroll_wheel_event_process(wheel, event);
        break;
    }
}

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_speed_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
gx_scroll_wheel_gradient_alpha_set

Assign gradient alpha values for optional overlay gradient

Prototype

UINT  gx_scroll_wheel_gradient_alpha_set(GX_SCROLL_WHEEL *wheel,
                                         GX_UBYTE start_alpha,
                                         GX_UBYTE end_alpha);

Description

This service defines the starting and ending alpha values for an optional gradient overlay of the scroll wheel widget.

All scroll wheel widgets support a “fade” effect of the scroll wheel rows as the rows near the top and bottom edge of the scroll wheel widget. This fade effect is accomplished by drawing a gradient pixelmap over the scroll wheel rows, which make the rows appear to fade out as the rows are drawn near the top and bottom of the scroll wheel widget.

This API service allows the application to modify the fading effect intensity, or disable this effect entirely by setting the start and end alpha values to 0.

The gradient pixelmap is created at runtime when the scroll wheel initially becomes visible. This requires that a runtime memory allocation service has been defined using _gx_system_memory_allocator_set(). If no memory allocator function has been defined, the gradient image will not be created and no fade effect will be available.

Parameters

wheel Pointer to generic scroll wheel control block
start_alpha The overlay gradient starting alpha value.
end_alpha The overlay gradient ending alpha value.

Return Values

GX_SUCCESS (0x00) Successfully set the scroll wheel gradient alpha
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid
Allowed From

Initialization and threads

Example

```c
status = gx_scroll_wheel_gradient_alpha_set(&wheel, 240, 0);
/* if status == GX_SUCCESS the wheel gradient alpha values were
Successfully assigned. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_speed_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
gx_scroll_wheel_row_height_set

Assign the row height for each wheel row

Prototype

UINT  gx_scroll_wheel_row_height_set(GX_SCROLL_WHEEL *wheel,  
                                         GX_VALUE row_height);

Description

This service assigns the row height for each row of the scroll wheel.  
Note that if the scroll wheel has style GX_STYLE_TEXT_SCROLL_WHEEL_ROUND, the row height 
passes through a transform which effectively reduces the row height 
as the row nears the top or bottom edge of the wheel.

Parameters

- **wheel**: Pointer to generic scroll wheel control block
- **row_height**: Row height value, in pixels.

Return Values

- **GX_SUCCESS**: (0x00) Successfully set scroll wheel height
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

- Initialization and threads
Example

```c
status = gx_scroll_wheel_row_height_set(&wheel, 40);
/* if status == GX_SUCCESS the wheel row height has been set to 40 pixels. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_gradient_alpha_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_speed_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
gx_scroll_wheel_selected_background_set
Assign background image for wheel selected row

Prototype

UINT gx_scroll_wheel_selected_background_set(
        GX_SCROLL_WHEEL* wheel, GX_RESOURCE_ID image_id);

Description

This service assigns an optional pixelmap ID that is drawn behind
the selected row of the scroll wheel. This can be used to highlight
the selected row so that the user can easily distinguish which row of
the scroll wheel is selected.

Parameters

- **wheel**: Pointer to generic scroll wheel control block
- **image_id**: Pixelmap ID to use as the selected row background image.

Return Values

- **GX_SUCCESS**: (0x00) Successfully set scroll wheel background
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

Initialization and threads

Example

```c
status = gx_scroll_wheel_selected_background_set(&wheel,
        GX_PIXELMAP_ID_SELECTED_ROW);
/* if status == GX_SUCCESS the background image has been
 assigned. */
```

See Also

- `gx_numeric_scroll_wheel_create`, `gx_numeric_scroll_wheel_range_set`,
- `gx_scroll_wheel_create`, `gx_scroll_wheel_event_process`,
- `gx_scroll_wheel_gradient_alpha_set`, `gx_scroll_wheel_row_height_set`,
- `gx_scroll_wheel_selected_get`, `gx_scroll_wheel_selected_set`,
- `gx_scroll_wheel_speed_set`, `gx_scroll_wheel_total_rows_set`,

GUIX User Guide
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
gx_scroll_wheel_selected_get

Retrieve the currently selected wheel row

Prototype

UINT    gx_scroll_wheel_selected_get(GX_SCROLL_WHEEL *wheel,
                                        INT  *row);

Description

This service will query the scroll wheel to retrieve the currently selected row. The caller must pass the location to return the selected row index as the second parameter to this function.

Parameters

    wheel  Pointer to generic scroll wheel control block
    row    Location in which selected row value will be returned.

Return Values

   GX_SUCCESS        (0x00)  Successfully retrieved selected wheel row
   GX_CALLER_ERROR   (0x11)  Invalid caller of this function
   GX_PTR_ERROR      (0x07)  Invalid pointer
   GX_INVALID_WIDGET (0x19)  Widget not valid

Allowed From

    Initialization and threads
Example

```c
INT row;
status = gx_scroll_wheel_selected_get(&wheel, &row);

/* if status == GX_SUCCESS the selected row has been returned in the row variable. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set, gx_scroll_wheel_create, gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_set, gx_scroll_wheel_speed_set, gx_scroll_wheel_total_rows_set, gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create, gx_string_scroll_wheel_text_get
gx_scroll_wheel_selected_set

Assign selected scroll wheel row

Prototype

UINT  gx_scroll_wheel_selected_set(GX_SCROLL_WHEEL *wheel,
       INT  row);

Description

This service assigns the currently selected scroll wheel row.

Parameters

wheel  Pointer to generic scroll wheel control block
row   Row of the scroll wheel to be selected.

Return Values

GX_SUCCESS  (0x00)  Successfully set the selected wheel row
GX_CALLER_ERROR  (0x11)  Invalid caller of this function
GX_PTR_ERROR  (0x07)  Invalid pointer
GX_INVALID_WIDGET  (0x12)  Widget not valid

Allowed From

Initialization and threads

Example

status = gx_scroll_wheel_selected_set(&wheel, 20);
/* if status == GX_SUCCESS the scroll wheel has been set to select row 20 */

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_speed_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
**gx_scroll_wheel_speed_set**  
Assign scrolling speed

**Prototype**

```c
UINT gx_scroll_wheel_speed_set(GX_SCROLL_WHEEL *wheel,
                               GX_FIXED_VAL start_speed_rate,
                               GX_FIXED_VAL end_speed_rate,
                               GX_VALUE max_steps,
                               GX_VALUE delay);
```

**Description**

This service assigns the scrolling speed for the scroll wheel widget.

**Parameters**

- **wheel**: Pointer to generic scroll wheel control block.
- **start_speed_rate**: The rate of scrolling start speed to flick speed.
- **end_speed_rate**: The rate of scrolling end speed to flick speed.
- **max_steps**: Max steps for scrolling.
- **delay**: Delay time of each step.

**Return Values**

- **GX_SUCCESS** (0x00): Successfully set wheel speed.
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function.
- **GX_PTR_ERROR** (0x07): Invalid pointer.
- **GX_INVALID_WIDGET** (0x12): Widget not valid.
- **GX_INVALID_VALUE** (0x22): Invalid value.

**Allowed From**

- Initialization and threads
Example

```c
status = gx_scroll_wheel_speed_set(&wheel, GX_FIXED_VAL_MAKE(2),
                                  GX_FIXED_VAL_MAKE(1) / 2, 10, 2);
/* if status == GX_SUCCESS the scroll wheel speed has been
   successfully set. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_selected_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set, gx_string_scroll_wheel_create,
gx_string_scroll_wheel_text_get
gx_scroll_wheel_total_rows_set

Assign the total scroll wheel rows available

Prototype

UINT  gx_scroll_wheel_total_rows_set(GX_SCROLL_WHEEL *wheel,
                                      INT  total_rows);

Description

This service assigns the number of rows available in the indicated scroll wheel. The scroll wheel widget usually receives the row content from the application in the form of an array of strings or user supplied string data. This API informs the scroll wheel of the total number of rows that should be presented to the user.

Parameters

- **wheel**: Pointer to generic scroll wheel control block
- **total_rows**: Total number of wheel rows to present to the user.

Return Values

- **GX_SUCCESS**(0x00) Successfully set scroll wheel total row
- **GX_CALLER_ERROR**(0x11) Invalid caller of this function
- **GX_PTR_ERROR**(0x07) Invalid pointer
- **GX_INVALID_WIDGET**(0x12) Widget not valid
- **GX_INVALID_VALUE**(0x22) Invalid value

Allowed From

Initialization and threads
Example

```c
status = gx_scroll_wheel_total_rows_set(&wheel, 100);
/* if status == GX_SUCCESS the scroll wheel has been changed to display 100 total rows */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_speed_set, gx_text_scroll_wheel_callback_set,
gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set,
gx_string_scroll_wheel_create, gx_string_scroll_wheel_text_get
**gx_scrollbar_draw**

Prototype

\texttt{VOID \textbf{gx\_scrollbar\_draw}(GX\_SCROLLBAR *scrollbar);}  

Description

This service draws a scrollbar. A common drawing function is used for both vertical and horizontal scrollbar widgets. This service is called internally by the GUIX canvas refresh, but can also be called by overridden drawing functions.

Parameters

\textbf{scrollbar} \hspace{1cm} Scrollbar widget to draw

Return Values

None

Allowed From

Threads

Example

\begin{verbatim}
/* Write a custom scrollbar drawing function. */

VOID my_scrollbar_draw(GX_SCROLLBAR *scrollbar)
{
    /* Call default scrollbar draw. */
    \textbf{gx\_scrollbar\_draw}(thumb);

    /* Add your own drawing here. */
}
\end{verbatim}

See Also

\texttt{gx\_horizontal\_scrollbar\_create, gx\_scrollbar\_event\_process,}
\texttt{gx\_scrollbar\_limit\_check, gx\_scrollbar\_reset, gx\_vertical\_scrollbar\_create}
**gx_scrollbar_event_process**

**Process scrollbar event**

**Prototype**

```c
UINT gx_scrollbar_event_process(GX_SCROLLBAR *scrollbar,
                                GX_EVENT *event);
```

**Description**

This service processes a scrollbar event. A common event handling function used for both vertical and horizontal scrollbar widgets.

**Parameters**

- `scrollbar` : Scrollbar widget control block
- `event` : Pointer to event to process

**Return Values**

- **GX_SUCCESS** (0x00) : Successful scrollbar event process
- **GX_CALLER_ERROR** (0x11) : Invalid caller of this function
- **GX_PTR_ERROR** (0x07) : Invalid pointer
- **GX_INVALID_WIDGET** (0x12) : Widget not valid

**Allowed From**

Threads
Example

/* Call generic scrollbar event processing as part of custom event
processing function. */

UINT custom_scrollbar_event_process(GX_SCROLLBAR *scrollbar,
   GX_EVENT *event)
{
    switch(event->gx_event_type)
    {
      case xyz:
        /* insert custom event handling here */
        break;

      default:
        /* pass all other events to the generic scrollbar
           event processing */
        gx_scrollbar_event_process(scrollbar, event);
        break;
    }
}

See Also

gx_horizontal_scrollbar_create, gx_scrollbar_draw, gx_scrollbar_limit_check,
gx_scrollbar_reset, gx_vertical_scrollbar_create
gx_scrollbar_limit_check

Check scrollbar limit

Prototype

UINT  gx_scrollbar_limit_check(GX_SCROLLBAR *scrollbar);

Description

This service checks the limit of the scrollbar and prevents the scrollbar thumbwheel from traveling beyond the predefined limits.

Parameters

scrollbar  Scrollbar widget control block

Return Values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful scrollbar limit check</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Check scrollbar limit of "my_scrollbar". */
status =  gx_scrollbar_limit_check(&my_scrollbar);

/* If status is GX_SUCCESS the limit of scrollbar "my_scrollbar"
has been checked. */

See Also

gx_horizontalScrollbar_create, gx_scrollbar_draw, gx_scrollbar_event_process,
gx_scrollbar_reset, gx_verticalScrollbar_create
**gx_scrollbar_reset**

Reset scrollbar

**Prototype**

```c
UINT gx_scrollbar_reset(GX_SCROLLBAR *scrollbar,
                         GX_SCROLL_INFO *info);
```

**Description**

This service resets the scrollbar.

**Parameters**

- **scrollbar**: Scrollbar widget control block
- **info**: Pointer to GX_SCROLL_INFO structure that defines the scrollbar limits, current value, and step or increment. Appendix I contains definition to GX_SCROLL_INFO structure.

**Return Values**

- **GX_SUCCESS** (0x00): Successful scrollbar reset
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_VALUE** (0x22): Scroll info not valid

**Allowed From**

Initialization and threads
Example

/* Reset scrollbar "my_scrollbar". */

GX_SCROLL_INFO my_info;

my_info.gx_scroll_value = 0;
my_info.gx_scroll_minimum = 0;
my_info.gx_scroll_maximum = 100;
my_info.gx_scroll_visible = 10;
my_info.gx_scroll_increment = 1;

status = gx_scrollbar_reset(&my_scrollbar, &my_info);

/* If status is GX_SUCCESS the scrollbar "my_scrollbar" has been reset. */

See Also

gx_horizontal_scrollbar_create, gx_scrollbar_draw, gx_scrollbar_event_process,
gx_scrollbar_limit_check, gx_verticalScrollbar_create
**gx_single_line_text_input_backspace**

Process a backspace character in text input widget

Prototype

```c
UINT gx_single_line_text_input_backspace(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

Description

This service deletes the character before text input cursor position. This service is called internally when a backspace key down event is received, but can also be invoked by the application.

Parameters

- **text_input**: Single-line text input widget control block

Return Values

- **GX_SUCCESS** (0x00): Successful single-line text input create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x23): Widget not valid

Allowed From

Initialization and threads
Example

/* Delete a character before the cursor of "my_text_input". */
status = gx_single_line_text_input_backspace(&my_text_input);

/* If status is GX_SUCCESS the character before the cursor has been
deleted. */

See Also

gx_single_line_text_input_buffer_clear, gx_single_line_text_input_buffer_get,
gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
gx_single_line_text_input_buffer_clear

Deletes all characters from the text input buffer

Prototype

UINT gx_single_line_text_input_buffer_clear(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);

Description

This service deletes all characters from the text input buffer.

Parameters

    text_input            Single-line text input widget control block

Return Values

    GX_SUCCESS            (0x00) Successfully cleared single-line text input buffer
    GX_CALLER_ERROR       (0x11) Invalid caller of this function
    GX_PTR_ERROR          (0x07) Invalid pointer
    GX_INVALID_WIDGET     (0x12) Widget not valid

Allowed From

    Initialization and threads
Example

/* clear input buffer of "my_text_input". */
status = gx_single_line_text_input_clear(&my_text_input);

/* If status is GX_SUCCESS the text input widget has emptied its input buffer. */

See Also

gx_single_line_text_input_buffer_backspace,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
gx_single_line_text_input_buffer_get

Retrieves buffer information of text input widget

Prototype

UINT gx_single_line_text_input_buffer_get(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_CHAR **buffer_address,
    UINT *content_size, UINT *buffer_size);

Description

This service retrieves buffer information of the text input widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_input</td>
<td>Single-line text input widget control block</td>
</tr>
<tr>
<td>buffer_address</td>
<td>The address of the input buffer</td>
</tr>
<tr>
<td>content_size</td>
<td>The byte count of the input data</td>
</tr>
<tr>
<td>buffer_size</td>
<td>The size of the input buffer</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully retrieved buffer information
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

Allowed From

Initialization and threads
Example

GX_CHAR *buffer_address;
UINT buffer_size;
UINT content_size;

/* Retrieve buffer information of "my_text_input" widget. */
status = gx_single_line_text_input_buffer_get(&my_text_input,
   &buffer_address, &string_size, &buffer_size);

/* If status is GX_SUCCESS the value of buffer_address, string_size
and buffer_size has been retrieved. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_end,
gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
**gx_single_line_text_input_character_delete**

Delete the character at the current cursor position

**Prototype**

```c
UINT  gx_single_line_text_input_character_delete(
     GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

**Description**

This service deletes the character after the text input cursor position. This service is called internally when a delete key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**
  Single-line text input widget control block

**Return Values**

- **GX_SUCCESS**  (0x00)  Successfully delete a character after the cursor
- **GX_CALLER_ERROR**  (0x11)  Invalid caller of this function
- **GX_PTR_ERROR**  (0x07)  Invalid pointer
- **GX_INVALID_WIDGET**  (0x12)  Widget not valid

**Allowed From**

Initialization and threads
Example

/* Delete the character after the cursor of "my_text_input". */
status = gx_single_line_text_input_character_delete(&my_text_input);

/* If status is GX_SUCCESS the character after the cursor has been deleted. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_insert,
gx_single_line_text_input_create, gx_single_line_text_input_draw,
gx_single_line_text_input_draw_position_get, gx_single_line_text_input_end,
gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_multi_line_text_input_create, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_character_insert**  
Insert a character string at current cursor position

**Prototype**

```c
UINT gx_single_line_text_input_character_insert(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_UBYTE *insert_str,
    UINT insert_size);
```

**Description**

This service inserts a character string into the text input string buffer at the current cursor position.

**Parameters**

- `text_input` Single-line text input widget control block
- `insert_str` Character string to be inserted
- `insert_size` Byte count to be inserted

**Return Values**

- **GX_SUCCESS** (0x00) Successfully inserted the character
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads
Example

/* Insert characters at current cursor position. */
GX_CHAR insert_text[10] = "insert";
status = gx_single_line_text_input_character_insert(&my_text_input,
    insert_text,
    GX_STRLEN(insert_text));

/* If status is GX_SUCCESS the text input widget has successfully
insert the string. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_create, gx_single_line_text_input_draw,
gx_single_line_text_input_draw_position_get, gx_single_line_text_input_end,
gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
**gx_single_line_text_input_create**

Create a text input widget

**Prototype**

```c
UINT gx_single_line_text_input_create(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_CONST GX_CHAR *name, GX_WIDGET *parent,
    GX_CHAR *input_buffer, UINT buffer_size,
    UINT style, USHORT text_input_id,
    GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a text input widget. The caller must provide storage for the input string and indicate the maximum length of the string.

GX_SINGLE_LINE_TEXT_INPUT is derived from GX_PROMPT and therefore all gx_prompt services may be used with GX_SINGLE_LINE_TEXT_INPUT widgets.

**Parameters**

- **text_input**: Single-line text input widget control block
- **name**: Optional widget logical name
- **parent**: Optional parent widget
- **input_buffer**: Storage for input string
- **buffer_size**: Size of input string storage area, in bytes.
- **style**: Text input style flags
- **text_input_id**: Optional ID of the input widget
- **size**: Rectangle defining initial widget size

**Return Values**

- **GX_SUCCESS**: (0x00) Successful single-line text input create
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_ALREADY_CREATED**: (0x13) Widget already created
- **GX_INVALID_SIZE**: (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET**: (0x12) Widget not valid
Allowed From

Initialization and threads

Example

GX_SINGLE_LINE_TEXT_INPUT my_text_input;
static GX_CHAR text_input_buffer[100];
GX_RECTANGLE size;

/* Define widget size. */
gx_utility_rectangle_define(&size, 10, 10, 110, 40);

/* Create single-line text input widget "my_text_input". */
status = gx_single_line_text_input_create(&my_text_input, 
                                          "text_input", GX_NULL, text_input_buffer, 100, 
                                          GX_STYLE_ENABLED, 0, &size);

/* If status is GX_SUCCESS, the text input widget has been created. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear, 
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete, 
gx_single_line_text_input_character_insert, gx_single_line_text_input_draw, 
gx_single_line_text_input_draw_position_get, gx_single_line_text_input_end, 
gx_single_line_text_input_event_process, 
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home, 
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get, 
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add, 
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set, 
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select, 
gx_single_line_text_input_text_set
**gx_single_line_text_input_draw**

Draw a text input widget

Prototype

```c
VOID gx_single_line_text_input_draw(
    GX-single_line_text_INPUT *text_input);
```

Description

This service draws a text input widget. This service is normally called internally during canvas refresh, but can also be called from custom text input drawing functions.

Parameters

- **text_input**: Single-line text input widget control block

Return Values

None

Allowed From

Threads

Example

```c
/* Write a custom single line text input draw function. */
VOID my_sl_text_input_draw(GX_SINGLE_LINE_TEXT_INPUT *input)
{
    /* Call default single line text input draw. */
    gx_single_line_text_input_draw(input);
    /* Add your own drawing here. */
}
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_draw_position_get, gx_single_line_text_input_end,
gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
gx_single_line_text_input_draw_position_get
Retrieve text draw start position

Prototype

UINT gx_single_line_text_input_draw_position_get(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_VALUE *xpos, GX_VALUE *ypos);

Description

This service retrieves the draw start position of text input text.

Parameters

- **text_input**: Single-line text input widget control block
- **xpos**: Retrieved draw start position in x coordinate
- **ypos**: Retrieved draw start position in y coordinate

Return Values

- **GX_SUCCESS**: (0x00) Successfully move text input cursor to end
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

- Initialization and threads
Example

/* Write a custom single line text input draw function. */

VOID my_sl_text_input_draw(GX_SINGLE_LINE_TEXT_INPUT *input)
{
    GX_VALUE xpos;
    GX_VALUE ypos;

    /* Draw background. */
    gx_widget_border_draw(input, border_color, upper_fill_color,
                           lower_fill_color, GX_TRUE);

    /* Retrieve text draw start position. */
    gx_single_line_text_input_draw_position_get(input, xpos, ypos);

    /* Add your text drawing here. */
}

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
**gx_single_line_text_input_end**

Move the text input cursor to the string end

Prototype

```c
UINT gx_single_line_text_input_end(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

Description

This service positions the text input widget cursor at the end of the input string. This service is called internally when an end key down event is received, but can also be invoked by the application.

Parameters

- **text_input**  
  Single-line text input widget control block

Return Values

- **GX_SUCCESS**  
  (0x00)  Successfully move text input cursor to end
- **GX_CALLER_ERROR**  
  (0x11)  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  Invalid pointer
- **GX_INVALID_WIDGET**  
  (0x12)  Widget not valid

Allowed From

Initialization and threads
Example

/* Move input cursor to end. */
status = gx_single_line_text_input_end(&my_text_input);

/* If status is GX_SUCCESS, text text input cursor has been moved to end. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
gx_single_line_text_input_event_process

Text input widget event processing function

Prototype

UINT gx_single_line_text_input_event_process(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_EVENT *event_ptr);

Description

This service processes a single line text input event. This function is internally referenced by the gx_single_line_text_input_create function, but is exposed for use by the application in those cases where the application defines a custom single line text input event processing function.

Parameters

- **text_input**: Single-line text input widget control block
- **event_ptr**: Pointer to GX_EVENT structure

Return Values

- **GX_SUCCESS** (0x00): Successfully processed text input event
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

- Initialization and threads
Example

/* Call generic single line text input event processing as part of custom event processing function. */

UINT custom_sl_text_input_event_process(
    GX_SINGLE_LINE_TEXT_INPUT *input,
    GX_EVENT *event)
{
    UINT status = GX_SUCCESS;

    switch(event->gx_event_type)
    {
        case xyz:
            /* Insert custom event handling here */
            break;

        default:
            /* Pass all other events to the default single line text input event processing */
            status =
                gx_single_line_text_input_event_process(input, event);
            break;
    }
    return status;
}

See Also
gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_fill_color_set,
gx_single_line_text_input_position_get, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_add, gx_single_line_text_input_style_remove,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_fill_color_set**

Set single line text input background color

**Prototype**

```c
UINT gx_single_line_text_input_fill_color_set(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_RESOURCE_ID normal_fill_color_id,
    GX_RESOURCE_ID selected_fill_color_id,
    GX_RESOURCE_ID disabled_fill_color_id,
    GX_RESOURCE_ID readonly_fill_color_id);
```

**Description**

This service sets the fill color of the single line text input.

**Parameters**

- **text_input**
  Pointer to single line text input control block

- **normal_fill_color_id**
  Resource ID of the widget fill color in normal state. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

- **selected_fill_color_id**
  Resource ID of the widget fill color when the widget gain focus. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

- **disabled_fill_color_id**
  Resource ID of the widget fill color when the style GX_STYLE_ENABLED is not set. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

- **readonly_fill_color_id**
  Resource ID of the widget fill color when both style GX_STYLE_ENABLED and GX_STYLE_TEXT_INPUT_READYONLY are set. **Appendix A** contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
Return Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successful single line text input fill color set</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>0x12</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Set fill colors for single line text input "my_text_input". */
status = gx_single_line_text_input_fill_color_set(&my_text_input,
        GX_COLOR_ID_NORMAL_FILL,
        GX_COLOR_ID_SELECTED_FILL,
        GX_COLOR_ID_DISABLED_FILL,
        GX_COLOR_ID_READONLY_FILL);

/* If status is GX_SUCCESS, the fill color of "my_text_input" was set. */

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_home, gx_single_line_text_input_left_arrow,
gx_single_line_text_input_position_get, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_add, gx_single_line_text_input_style_remove,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_home**

Move the text input cursor to the home position

**Prototype**

```c
UINT gx_single_line_text_input_home(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

**Description**

This service moves the text input cursor position to the start of the input string. This service is called internally when a home key down event is received, but can also be invoked by the application.

**Parameters**

- `text_input`:
  Single-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00)
  Successfully moved cursor to the home position
- **GX_CALLER_ERROR** (0x11)
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)
  Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Move cursor to the start of the input text. */
status = gx_single_line_text_input_home(&my_text_input);

/* If status is GX_SUCCESS the cursor has been moved to the home position */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_left_arrow,
gx_single_line_text_input_position_get, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_add, gx_single_line_text_input_style_remove,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_left_arrow**

**Move input cursor one character to the left**

**Prototype**

```c
UINT  gx_single_line_text_input_left_arrow(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

**Description**

This service moves the text input cursor one character to the left. This service is called internally when a left key down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**: Single-line text input widget control block

**Return Values**

- **GX_SUCCESS** (0x00): Successfully moved cursor to the left
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Move the cursor one character to the left. */
status = gx_single_line_text_input_left_arrow(&my_text_input);

/* If status is GX_SUCCESS the text input cursor has been moved one character to the left. */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_position_get, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_add, gx_single_line_text_input_style_remove,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_position_get**  
Move cursor to pixel position

**Prototype**

```c
UINT  gx_single_line_text_input_position_get(  
    GX_SINGLE_LINE_TEXT_INPUT *text_input,  
    INT pixel_position);
```

**Description**

This service positions the text input cursor based on the requested pixel position. The text input cursor index will be calculated based on the x value of the pixel position. This service is called internally when a pen down event is received, but can also be invoked by the application.

**Parameters**

- **text_input**  
  Single-line text input widget control block

- **pixel_position**  
  X value of pixel position

**Return Values**

- **GX_SUCCESS**  
  (0x00) Successfully set the cursor to requested position

- **GX_CALLER_ERROR**  
  (0x11) Invalid caller of this function

- **GX_PTR_ERROR**  
  (0x07) Invalid pointer

- **GX_INVALID_WIDGET**  
  (0x12) Invalid widget

**Allowed From**

Initialization and threads

**Example**

```c
/* Set cursor to requested position. */
status = gx_single_line_text_input_position_get(&my_text_input, 100);

/* If status is GX_SUCCESS the text input widget cursor has been positioned */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_add, gx_single_line_text_input_style_remove,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
gx_single_line_text_input_right_arrow

Move input cursor one character to the right

Prototype

UINT gx_single_line_text_input_right_arrow(
    GX_SINGLE_LINE_TEXT_INPUT *text_input);

Description

This service moves the text input cursor one character to the right. This service is called internally when a right key down event is received, but can also be invoked by the application.

Parameters

text_input

Single-line text input widget control block

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully moved cursor to the right</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Move cursor one character to the right. */
status = gx_single_line_text_input_right_arrow(&my_text_input);

/* If status is GX_SUCCESS the text input cursor has been moved one character to the right. */
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_position_get, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
**gx_single_line_text_input_style_add**

Add styles

Prototype

```c
UINT  gx_single_line_text_input_style_add(
     GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);
```

Description

This service adds the specified style(s) to the single line text input widget.

Parameters

- **text_input**: Single-line text input widget control block
- **style**: New style to add. **Appendix D** contains pre-defined general styles for all widgets

Return Values

- **GX_SUCCESS**: (0x00) Successfully added style to widget
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

Initialization and threads

Example

```c
/* Add a new style to "my_text_input". */
status = gx_single_line_text_input_style_add(&my_text_input,
                                             GX_STYLE_CURSOR_AWAYS);

/* If status is GX_SUCCESS the GX_STYLE_CURSOR_SHOR have been successfully added. */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gax_single_line_text_input_position_get, gx_single_line_text_input_right_arrow,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
**gx_single_line_text_input_style_remove**

Remove styles

**Prototype**

```c
UINT gx_single_line_text_input_style_remove(
    GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);
```

**Description**

This service removes the specified style(s) from the single line text input widget.

**Parameters**

- **text_input** Single-line text input widget control block
- **style** Style(s) to remove. Appendix D contains pre-defined general styles for all widgets

**Return Values**

- **GX_SUCCESS** (0x00) Successfully removed style(s) from widget
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Remove cursor blink style from "my_text_input". */
status = gx_single_line_text_input_style_remove(&my_text_input,
    GX_STYLE_CURSOR_BLINK);

/* If status is GX_SUCCESS the GX_STYLE_CURSOR_BLINK style has been successfully removed. */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_home, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_set, gx_single_line_text_input_text_color_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
gx_single_line_text_input_style_set

Set text input styles

Prototype

UINT    gx_single_line_text_input_style_set(
           GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);

Description

This service sets the specified style(s) to the single line text input widget.

Parameters

text_input
    Single-line text input widget control block
style
    style flags to assign

Return Values

GX_SUCCESS (0x00) Successfully set the text input style
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads

Example

/* Set style for “my text input”. */
status = gx_single_line_text_input_style_set(&my_text_input,
                                           GX_STYLE_ENABLED | GX_STYLE_CURSOR_BLINK);

/* If status is GX_SUCCESS the text input style has been successfully set to the specified styles. */
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_home, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove,
gx_single_line_text_input_text_color_set, gx_single_line_text_input_text_select,
gx_single_line_text_input_text_set
gx_single_line_text_input_text_color_set

Set single line text input text color

Prototype

UINT     gx_single_line_text_input_text_color_set(
          GX_SINGLE_LINE_TEXT_INPUT *text_input,
          GX_RESOURCE_ID normal_text_color_id,
          GX_RESOURCE_ID selected_text_color_id,
          GX_RESOURCE_ID disabled_text_color_id,
          GX_RESOURCE_ID readonly_text_color_id);

Description

This service sets the text color of the single line text input.

Parameters

text_input Pointer to single line text input control block
normal_text_color_id Resource ID of the text color in normal state. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
selected_text_color_id Resource ID of the text color when the widget gain focus. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
disabled_text_color_id Resource ID of the text color when the style GX_STYLE_ENABLED is not set. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
readonly_text_color_id Resource ID of the text color when both style GX_STYLE_ENABLED and GX_STYLE_TEXT_INPUT_READONLY are set. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

Return Values
**GX_SUCCESS** (0x00) Successful single line text input text color set

**GX_CALLER_ERROR** (0x11) Invalid caller of this function

**GX_PTR_ERROR** (0x07) Invalid pointer

**GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Set text colors for single line text input "my_text_input". */
status = gx_single_line_text_input_text_color_set(&my_text_input,
                                               GX_COLOR_ID_NORMAL_TEXT,
                                               GX_COLOR_ID_SELECTED_TEXT,
                                               GX_COLOR_ID_DISABLED_TEXT,
                                               GX_COLOR_ID_READONLY_TEXT);

/* If status is GX_SUCCESS, the text color "my_text_input" was set. */
```

**See Also**

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_input_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_select, gx_single_line_text_input_text_set
**gx_single_line_text_input_text_select**

**Prototype**

```c
UINT gx_single_line_text_input_text_color_set(  
    GX_SINGLE_LINE_TEXT_INPUT *text_input,  
    UINT start_index, UINT end_index);
```

**Description**

This service selects text with specified start mark and end mark index and highlights the selected text with the selected fill and text colors.

**Parameters**

- **text_input**: Pointer to single line text input control block
- **start_index**: Index of the first selected character
- **end_index**: Index of the last selected character

**Return Values**

- **GX_SUCCESS**: (0x00) Successful single line text input text selection
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid
- **GX_INVALID_VALUE**: (0x22) Index value not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Select text between index [0, 9]. */
status = gx_single_line_text_input_text_select(&my_text_input, 0, 9);

/* If status is GX_SUCCESS, the text between index [0, 9] "my_text_input" was selected. */
```
See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear,
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete,
gx_single_line_text_input_character_insert, gx_single_line_text_input_create,
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get,
gx_single_line_text_input_end, gx_single_line_text_input_event_process,
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home,
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get,
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add,
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set,
gx_single_line_text_input_text_set
**gx_single_line_text_input_text_set**

Set single line text input text (deprecated)

Prototype

```c
UINT gx_single_line_text_input_text_set(
    GX_SINGLE_LINE_TEXT_INPUT *text_input,
    GX_CONST GX_CHAR *text);
```

Description

This service has been deprecated in favor of
`gx_single_line_text_input_text_set_ext()`

This service sets the text of the single line text input.

Parameters

- `text_input`: Pointer to single line text input control block
- `text`: NULL-terminated text string

Return Values

- `GX_SUCCESS` (0x00): Successful single line text input text color set
- `GX_CALLER_ERROR` (0x11): Invalid caller of this function
- `GX_PTR_ERROR` (0x07): Invalid pointer
- `GX_INVALID_WIDGET` (0x12): Widget not valid
- `GX_INVALID_STRING_LENGTH` (0x34): Invalid string length

Allowed From

Initialization and threads

Example

```c
GX_CONST GX_CHAR new_text = "Set Single Line Text Input Text";

/* Set text for single line text input "my_text_input". */
status = gx_single_line_text_input_text_set(&my_text_input,
                                           new_text);

/* If status is GX_SUCCESS, the text of "my_text_input" was set. */
```

See Also
gx_single_line_text_input_text_set_ext
**gx_single_line_text_input_text_set_ext**

Set single line text input text

**Prototype**

```c
UINT gx_single_line_text_input_text_set(  
    GX_SINGLE_LINE_TEXT_INPUT *text_input,  
    GX_CONST GX_STRING *string);
```

**Description**

This service sets the text of the single line text input.

**Parameters**

- **text_input**: Pointer to single line text input control block
- **string**: GX_STRING variable

**Return Values**

- **GX_SUCCESS** (0x00): Successful single line text input text color set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_STRING_LENGTH** (0x34): Invalid string length

**Allowed From**

Initialization and threads

**Example**

```c
GX_STRING new_string;
new_string.gx_string_ptr = "Set Single Line Text Input Text";
new_string.gx_string_length = strlen(new_string.gx_string_ptr);

/* Set text for single line text input "my_text_input". */
status = gx_single_line_text_input_text_set_ext(&my_text_input,
                                              &new_string);

/* If status is GX_SUCCESS, the string has been assigned to my_text_input. */
```

**See Also**
gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear, 
gx_single_line_text_input_buffer_get, gx_single_line_text_input_character_delete, 
gx_single_line_text_input_character_insert, gx_single_line_text_input_create, 
gx_single_line_text_input_draw, gx_single_line_text_draw_position_get, 
gx_single_line_text_input_end, gx_single_line_text_input_event_process, 
gx_single_line_text_input_fill_color_set, gx_single_line_text_input_home, 
gx_single_line_text_input_left_arrow, gx_single_line_text_input_position_get, 
gx_single_line_text_input_right_arrow, gx_single_line_text_input_style_add, 
gx_single_line_text_input_style_remove, gx_single_line_text_input_style_set, 
gx_single_line_text_input_text_set_ext
gx_slider_create

Create slider

Prototype

UINT  gx_slider_create(GX_SLIDER *slider, GX_CONST GX_CHAR *name, 
                        GX_WIDGET *parent, 
                        INT tick_count, 
                        GX_SLIDER_INFO *slider_info, 
                        ULONG style, USHORT slider_id, 
                        GX_CONST GX_RECTANGLE *size);

Description

This service creates a slider widget.

GX_SLIDER is derived from GX_WIDGET, and therefore all 
gx_widget API services may be used with GX_SLIDER type 
widgets.

Parameters

slider  Slider widget control block
name   Name of slider
parent Pointer to parent widget
tick_count Number of slider ticks
slider_info Pointer to slider info which is a structure 
    used to pass the slider value limits, slider 
    needle size and position, and other slider 
    parameters. Appendix I contains 
    definition to GX_SLIDER_INFO 
    structure.
style  Style of slider. Appendix D contains pre-
    defined general styles for all widgets as 
    well as widget-specific styles.
slider_id Application-defined ID of slider
size Dimensions of slider

Return Values

GX_SUCCESS  (0x00) Successful slider create
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_ALREADY_CREATED (0x13) Widget already created
**Allowed From**

Initialization and threads

**Example**

/* Create slider “my_slider”. */

GX_SLIDER my_slider;
GX_SLIDER_INFO info;
info.gx_slider_info_min_val = 0;
info.gx_slider_info_max_val = 100;
info.gx_slider_info_current_val = 50;
info.gx_slider_info_increment = 1;
info.gx_slider_info_min_travel = 20;
info.gx_slider_info_max_travel = 20;
info.gx_slider_info_needle_width = 20;
info.gx_slider_info_needle_height = 10;
info.gx_slider_info_needle_inset = 5;
info.gx_slider_info_needle_hotspot_offset = 5;

status = gx_slider_create(&my_slider, “my_slider”,
&my_parent, 10, info, GX_STYLE_ENABLED,
ID_MY_SLIDER, &size);

/* If status is GX_SUCCESS the slider “my_slider” has been created. */

**See Also**

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_draw, gx_slider_event_process, gx_slider_needle_draw,
gx_slider_needle_position_get, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set
**gx_slider_draw**

**Draw slider**

**Prototype**

```c
VOID gx_slider_draw(GX_SLIDER *slider);
```

**Description**

This service draws a slider. This service is used internally by the `gx_slider_create` function, but is also exposed for use by the application in those instances when a custom slider drawing function is defined.

**Parameters**

- `slider`  
  Slider widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom slider draw function. */

VOID my_slider_draw(GX_SLIDER *slider)
{
    /* Call default slider draw. */
    gx_slider_draw(slider);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_pixelmap_slider_create`, `gx_pixelmap_slider_draw`,
- `gx_pixelmap_slider_event_process`, `gx_pixelmap_slider_pixelmap_set`,
- `gx_slider_create`, `gx_slider_event_process`, `gx_slider_needle_draw`,
- `gx_slider_needle_position_get`, `gx_slider_needle_position_get`,
- `gx_slider_tickmarks_draw`, `gx_slider_travel_get`, `gx_slider_value_calculate`,
- `gx_slider_value_set`
**gx_slider_event_process**

Process slider event

Prototype

```c
UINT gx_slider_event_process(GX_SLIDER *slider, GX_EVENT *event);
```

Description

This service processes a slider event. This function is internally referenced by the `gx_slider_create` function, but is exposed for use by the application in those cases where the application defines a custom slider event processing function.

Parameters

- **slider**: Slider widget control block
- **event**: Pointer to event to process

Return Values

- **GX_SUCCESS** (0x00): Successful slider event process
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

- Threads
Example

/* Call generic slider event processing as part of custom event processing function. */

UINT custom_slider_event_process(GX_SLIDER *slider,
    GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;
    default:
        /* Pass all other events to the default slider event processing */
        status = gx_slider_event_process(slider, event);
        break;
    }
    return status;
}

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_needle_draw,
gx_slider_needle_position_get, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set
gx_slider_info_set

Set slider information block

Prototype

UINT gx_slider_info_set(GX_SLIDER *slider, GX_SLIDER_INFO *info);

Description

This service assigns the specified slider information such as slider minimum, slider maximum, and slider current value to the indicated slider. The slider will update the needle position and redraw based on the new slider information.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slider</td>
<td>Slider widget control block</td>
</tr>
<tr>
<td>info</td>
<td>Pointer to the slider information structure. Appendix I contains definition to GX_SLIDER_INFO structure.</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully set slider information
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

Allowed From

Initialization and threads
Example

GX_SLIDER_INFO my_slider_info;
my_slider_info.gx_slider_info_min_val = 0;
my_slider_info.gx_slider_info_max_val = 100;
my_slider_info.gx_slider_info_current_val = 50;
my_slider_info.gx_slider_info_increment = 1;
my_slider_info.gx_slider_info_min_travel = 20;
my_slider_info.gx_slider_info_max_travel = 20;
my_slider_info.gx_slider_info_needle_width = 10;
my_slider_info.gx_slider_info_needle_height = 10;
my_slider_info.gx_slider_info_needle_inset = 5;
my_slider_info.gx_slider_info_needle_hotspot_offset = 5;

/* Set slider information for slider "my_slider". */
status = gx_slider_info_set(&my_slider, &my_slider_info);

/* If status is GX_SUCCESS the "my_slider" is configured with
my_slider_info. */

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_needle_draw,
gx_slider_needle_position_get, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set
**gx_slider_needle_draw**

Draw slider needle

**Prototype**

```c
VOID gx_slider_needle_draw(GX_SLIDER *slider);
```

**Description**

This service draws a slider needle. This service is automatically called by the `gx_slider_draw` function, but may also be invoked by the application as part of a customized slider drawing function.

**Parameters**

- **slider**: Slider widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom slider draw function. */

VOID my_slider_draw(GX_SLIDER *slider)
{
    /* Add your own background draw here. */

    /* Call default tickmarks draw. */
    gx_slider_tickmarks_draw(slider);

    /* Call default slider needle draw. */
    gx_slider_needle_draw(slider);
}
```

**See Also**

- `gx_pixelmap_slider_create`, `gx_pixelmap_slider_draw`,
- `gx_pixelmap_slider_event_process`, `gx_pixelmap_slider_pixelmap_set`,
- `gx_slider_create`, `gx_slider_draw`, `gx_slider_event_process`,
- `gx_slider_needle_position_get`, `gx_slider_needle_position_get`,
- `gx_slider_tickmarks_draw`, `gx_slider_travel_get`, `gx_slider_value_calculate`,
- `gx_slider_value_set`
**gx_slider Needle_position_get**

Get slider needle position

Prototype

```c
UINT gx_slider_needle_position_get(GX_SLIDER *slider,
                                    GX_SLIDER_INFO *slider_info,
                                    GX_RECTANGLE *return_position);
```

Description

This service computes the slider needle position based on the current slider value.

Parameters

- **slider**: Slider widget control block
- **slider_info**: Pointer to slider information structure defining the slider limits, needle size and offset, and other slider parameters. Appendix I contains definition to GX_SLIDER_INFO structure.
- **return_position**: Pointer to destination for needle position

Return Values

- **GX_SUCCESS** (0x00): Successful slider needle position get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_VALUE** (0x22): Slider info not valid

Allowed From

Initialization and threads
Example

GX_RECTANGLE needle_position;

/* Get the needle position for slider “my_slider”. */
status = gx_slider_needle_position_get(&my_slider, &slider_info,
  &needle_position);

/* If status is GX_SUCCESS the needle position for slider
  “my_slider” has been retrieved. */

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_event_process,
gx_slider_needle_draw, gx_slider_needle_position_get,
gx_slider_tickmarks_draw, gx_slider_travel_get, gx_slider_value_calculate,
gx_slider_value_set
**gx_slider_tickmarks_draw**

**Draw slider tickmarks**

**Prototype**

```c
VOID gx_slider_tickmarks_draw(GX_SLIDER *slider);
```

**Description**

This service draws the slider tickmarks. This function is called internally by the `gx_slider_draw` function, but is exposed for use by applications that might implement a custom slider drawing function.

**Parameters**

- `slider`  
  Slider widget control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom slider draw function. */

VOID my_slider_draw(GX_SLIDER *slider)
{
    /* Add your own background draw here. */

    /* Call default tickmarks draw. */
    gx_slider_tickmarks_draw(slider);

    /* Call default slider needle draw. */
    gx_slider_needle_draw(slider);
}
```

**See Also**

- `gx_pixelmap_slider_create`, `gx_pixelmap_slider_draw`,
- `gx_pixelmap_slider_event_process`, `gx_pixelmap_slider_pixelmap_set`,
- `gx_slider_create`, `gx_slider_draw`, `gx_slider_event_process`,
- `gx_slider_needle_draw`, `gx_slider_needle_position_get`,
- `gx_slider_needle_position_get`, `gx_slider_travel_get`, `gx_slider_value_calculate`,
- `gx_slider_value_set`
gx_slider_travel_get

Get slider travel

Prototype

UINT  gx_slider_travel_get(GX_SLIDER *widget,
                               GX_SLIDER_INFO *info,
                               INT *return_min_travel,
                               INT *return_max_travel);

Description

This service gets the slider travel.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slider</td>
<td>Slider widget control block</td>
</tr>
<tr>
<td>info</td>
<td>Pointer to slider info structure. Appendix I contains definition to GX_SLIDER_INFO structure.</td>
</tr>
<tr>
<td>return_min_travel</td>
<td>Pointer to destination for minimum travel value</td>
</tr>
<tr>
<td>return_max_travel</td>
<td>Pointer to destination for maximum travel value</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful slider travel get</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22) Slider info not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Get travel information for slider "my_slider". */
status = gx_slider_travel_get(&my_slider, &info,
&my_min_travel, &my_max_travel);

/* If status is GX_SUCCESS the travel max/min values for slider
"my_slider" have been retrieved. */

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_event_process,
gx_slider_needle_draw, gx_slider_needle_position_get,
gx_slider_needle_position_get, gx_slider_tickmarks_draw,
gx_slider_value_calculate, gx_slider_value_set
gx_slider_value_calculate

Prototype

UINT gx_slider_value_calculate(GX_SLIDER *slider,
                                 GX_SLIDER_INFO *info,
                                 INT new_position);

Description

This service calculates the slider value based on the slider needle position. This function is called internally by GUIX when the user moves the slider needle, but can also be invoked by the application when implementing a custom slider widget.

Parameters

- **slider**: Slider widget control block
- **info**: Pointer to slider info. [Appendix I](#) contains definition to GX_LISDER_INFO structure.
- **new_position**: New slider position

Return Values

- **GX_SUCCESS** (0x00) Successful slider value calculate
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid
- **GX_INVALID_VALUE** (0x22) Slider info not valid

Allowed From

Initialization and threads
Example

/* Calculate new value for slider “my_slider”. */
status = gx_slider_value_calculate(&my_slider,
&my_slider.gx_slider_info, new_slider_position);

/* If status is GX_SUCCESS the slider value of “my_slider” has been calculated. */

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_event_process,
gx_slider_needle_draw, gx_slider_needle_position_get,
gx_slider_needle_position_get, gx_slider_tickmarks_draw, gx_slider_travel_get,
gx_slider_value_set
gx_slider_value_set

Set slider value

Prototype

UINT gx_slider_value_set(GX_SLIDER *slider,
                        GX_SLIDER_INFO *info, INT new_value);

Description

This service sets the slider value. This API can be called by the application to move a slider needle under program control, bypassing the need for user input to drag the slider needle.

Parameters

slider Sliding widget control block
info Pointer to slider info structure. Appendix I contains definition to GX_SLIDER_INFO structure
new_value New slider value

Return Values

GX_SUCCESS (0x00) Successful slider value set
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads
Example

/* Set new value for slider "my_slider". */
status = gx_slider_value_set(&my_slider,
    &my_slider.gx_slider_info, new_value);

/* If status is GX_SUCCESS the new value has been set for slider "my_slider". */

See Also

gx_pixelmap_slider_create, gx_pixelmap_slider_draw,
gx_pixelmap_slider_event_process, gx_pixelmap_slider_pixelmap_set,
gx_slider_create, gx_slider_draw, gx_slider_event_process,
gx_slider_needle_draw, gx_slider_needle_position_get,
gx_slider_needle_position_get, gx_slider_tickmarks_draw, gx_slider_travel_get,
gx_slider_value_calculate
gx_sprite_create

Create a sprite widget

Prototype

UINT gx_sprite_create(GX_SPRITE *sprite, GX_CONST GX_CHAR *name,
                      GX_WIDGET *parent,
                      GX_SPRITE_FRAME *frame_list,
                      USHORT frame_count,
                      ULONG style, USHORT sprite_id, GX_CONST
                      GX_RECTANGLE *size);

Description

This service creates a GX_SPRITE widget. A sprite is used to
display a sequence of pixelmaps as in an animation, or can be used
as a multi-state pixelmap display widget.

GX_SPRITE is derived from GX_WIDGET and supports all
gx_widget API services.

The GX_SPRITE widget requires an array of GX_SPRITE_FRAME
structures to define the sprite animation. Appendix I contains
definition to GX_SPRITE_FRAME structure.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sprite</td>
<td>Sprite widget control block</td>
</tr>
<tr>
<td>name</td>
<td>Optional sprite name</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>frame_list</td>
<td>An array of GX_SPRITE_FRAME structures</td>
</tr>
<tr>
<td>frame_count</td>
<td>specifies the number of entries in the</td>
</tr>
<tr>
<td></td>
<td>frame list array</td>
</tr>
<tr>
<td>style</td>
<td>Style of sprite. Appendix D contains pre-</td>
</tr>
<tr>
<td></td>
<td>defined general styles for all widgets as well</td>
</tr>
<tr>
<td></td>
<td>as widget-specific styles.</td>
</tr>
<tr>
<td>sprite_id</td>
<td>Application-defined ID of sprite</td>
</tr>
<tr>
<td>size</td>
<td>Dimensions of sprite</td>
</tr>
</tbody>
</table>
Return Values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful sprite create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13) Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Parent widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

- Initialization and threads

Example

```c
/* Create sprite “my_sprite”. */
status = gx_sprite_create(&my_sprite, "my_sprite", &my_parent,
        sprite_frame_list, frame_count,
        GX_STYLE_SPRITE_AUTO|GX_STYLE_SPRITE_LOOP,
        ID_MY_SPRITE, &size);

/* If status is GX_SUCCESS the sprite “my_sprite” has been created. */
```

See Also

- gx_sprite_start
- gx_sprite_stop
- gx_sprite_current_frame_set
- gx_sprite_frame_list_set
gx_sprite_current_frame_set

Assign sprite frame

Prototype

UINT gx_sprite_current_frame_set(GX_SPRITE *sprite,
                                 USHORT frame);

Description

This service assigns the current sprite frame. If a GX_SPRITE
widget is not auto-running, it can be used as a program controlled
state light, displaying the commanded frame pixelmap.

Parameters

sprite

Sprite widget control block

frame

Sprite frame to display

Return Values

GX_SUCCESS   (0x00)   Successful
GX_CALLER_ERROR (0x11)   Invalid caller of this function
GX_PTR_ERROR   (0x07)   Invalid pointer
GX_INVALID_WIDGET (0x12)   Widget not valid

Allowed From

Initialization and threads

Example

/* Assign frame number 3 as the current sprite frame */
status = gx_sprite_current_frame_set(&my_sprite, 3);

/* If status is GX_SUCCESS the sprite “my_sprite” will display
frame index 3. */

See Also

gx_sprite_start, gx_sprite_stop, gx_sprite_create, gx_sprite_frame_list_set
gx_sprite_frame_list_set

Assign or alter a sprite frame list

Prototype

UINT  gx_sprite_frame_list_set(GX_SPRITE *sprite,
                                    GX_SPRITE_FRAME *frame_list,
                                    USHORT frame_count);

Description

This service can be used to assign or re-assign the frame list used
by a sprite widget after the sprite widget has been created. For
information about the contents of a sprite frame list, refer to the
gx_sprite_create API documentation.

Parameters

  sprite       Sprite widget control block
  frame_list   Array of GX_SPRITE_FRAME structures
               or GX_NULL if no frame list.
  frame_count  Number of frames in frame list array

Return Values

  GX_SUCCESS    (0x00)  Successful sprite frame list set
  GX_CALLER_ERROR  (0x11)  Invalid caller of this function
  GX_PTR_ERROR  (0x07)  Invalid pointer
  GX_INVALID_WIDGET  (0x12)  Widget not valid

Allowed From

  Initialization and threads
Example

/* Assign framelist_1, which has 10 frames, to my_sprite */
status = gx_sprite_frame_list_set(&my_sprite, framelist_1, 10);

/* If status is GX_SUCCESS the new frame list is now associated
with this sprite */

See Also

gx_sprite_current_frame_set, gx_sprite_stop, gx_sprite_create, gx_sprite_create
**gx_sprite_start**

Start a sprite run sequence

**Prototype**

```c
UINT  gx_sprite_start(GX_SPRITE *sprite, USHORT frame);
```

**Description**

This service starts a sprite auto-run sequence. The sprite widget will cycle through the sprite frames until the last frame is reached, or will run continuously if the GX_SPRITE_LOOP style is set.

**Parameters**

- **sprite**: Sprite widget control block
- **frame**: Initial sprite frame to display, usually frame 0

**Return Values**

- **GX_SUCCESS** (0x00): Successfully started sprite run
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Threads

**Example**

```c
/* Start the sprite “my_sprite” */
status = gx_sprite_start(&my_sprite, 0);

/* If status is GX_SUCCESS the sprite “my_sprite” will start running */
```

**See Also**

- `gx_sprite_current_frame_set`
- `gx_sprite_stop`
- `gx_sprite_create`
- `gx_sprite_frame_list_set`
**gx_sprite_stop**

**Stop a sprite run sequence**

**Prototype**

```c
UINT  gx_sprite_stop(GX_SPRITE *sprite);
```

**Description**

This service stops a sprite auto-run sequence.

**Parameters**

- **sprite**
  Sprite widget control block

**Return Values**

- **GX_SUCCESS** (0x00)
  Successfully stopped sprite run
- **GX_CALLER_ERROR** (0x11)
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)
  Widget not valid

**Allowed From**

Threads

**Example**

```c
/* Stop the sprite sequence */
status = gx_sprite_stop(&my_sprite);
/* If status is GX_SUCCESS the sprite "my_sprite" is stopped. */
```

**See Also**

- `gx_sprite_current_frame_set`
- `gx_sprite_start`
- `gx_sprite_create`
- `gx_sprite_frame_list_set`
**gx_string_scroll_wheel_create**

Create a string type scroll wheel

**Prototype**

```c
UINT gx_string_scroll_wheel_create(GX_STRING_SCROLL_WHEEL *wheel,
                                   GX_CONST GX_CHAR *name, GX_WIDGET *parent, INT total_rows,
                                   GX_CONST GX_CHAR **string_list, ULONG style, USHORT Id,
                                   GX_CONST GX_RECTANGLE *size)
```

**Description**

This service creates a string type scroll wheel. `GX_STRING_SCROLL_WHEEL` is derived from `GX_TEXT_SCROLL_WHEEL`, and therefore all `gx_text_scroll_wheel` API functions maybe be used with `GX_STRING_SCROLL_WHEEL` widgets.

The application can pass in a simple string array to the create function which defines the strings that will be displayed by the scroll wheel, or the application can pass `GX_NULL` as the `string_list` parameter and call the `gx_string_scroll_wheel_string_id_list_set()` API to provide an array of String IDs. If the latter method is used the string scroll wheel will automatically switch the displayed strings if the active application language is modified.

As an alternative, if the strings to be displayed are not statically defined or not know at the time the scroll wheel is created, the application can pass `GX_NULL` as the `string_list` parameter, and call the API function `gx_text_scroll_wheel_callback_set()` to define a callback function which will provide the strings to be displayed in a real-time as-needed basis.

**Parameters**

- **wheel**: String scroll wheel control block address
- **name**: Application defined widget name
- **parent**: Wheel parent or `GX_NULL`
- **total_rows**: Total rows to be presented to user
- **string_list**: Statically defined string array, or `GX_NULL`
- **style**: Desired style flags
- **Id**: Application defined wheel style flags
- **size**: Initial scroll wheel size

**Return Values**
GX_SUCCESS (0x00) Successfully created string scroll wheel
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_ALREADY_CREATED (0x13) Widget already created
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From
Initialization and threads

Example
GX_CONST GX_CHAR *days[] = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};
GX_STRING_SCROLL_WHEEL wheel;

/* Create the string scroll wheel. */
status = gx_string_scroll_wheel_create(&wheel, "Day Wheel",
    root, 7, days,
    GX_STYLE_ENABLED|GX_STYLE_TEXT_CENTER|GX_STYLE_TRANSPARENT|
    GX_STYLE_WRAP|GX_STYLE_TEXT_SCROLL_WHEEL_ROUND,
    ID_SCROLL_WHEEL_DAY, &size);

/* If status is GX_SUCCESS the string scroll wheel has been created. */

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_string_scroll_wheel_string_id_list_set,
gx_string_scroll_wheel_string_list_set, gx_text_scroll_wheel_callback_set,
gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set
gx_string_scroll_wheel_string_id_list_set
Assign array of string IDs

Prototype

UINT gx_string_scroll_wheel_string_id_list_set(
    GX_STRING_SCROLL_WHEEL *wheel,
    GX_CONST GX_RESOURCE_ID *string_id_list, INT id_count)

Description

This service assigns an array of string IDs to a string scroll wheel widget. This method of assigning strings to a string scroll wheel is recommended if the strings are statically defined and the widget must operate in multiple languages. If this API is to be used, the scroll wheel widget should first be created with a GX_NULL string list.

Parameters

- **wheel**: String scroll wheel control block address
- **string_id_list**: Array of String IDs
- **id_count**: Size of the ID list.

Return Values

- **GX_SUCCESS** (0x00): Successfully set string ID array
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_VALUE** (0x22): Invalid ID list size

Allowed From

- Initialization and threads
Example

```
GX_CONST RESOURCE_ID wheel_ids[] = {
    GX_STRING_ID_SUNDAY,
    GX_STRING_ID_MONDAY,
    GX_STRING_ID_TUESDAY,
    GX_STRING_ID_WEDNESDAY,
    GX_STRING_ID_THURSDAY,
    GX_STRING_ID_FRIDAY,
    GX_STRING_ID_SATURDAY
};
GX_STRING_SCROLL_WHEEL wheel;

/* Stop the sprite sequence */
status = gx_string_scroll_wheel_string_id_list_set(&wheel, wheel_ids, 7);

/* If status is GX_SUCCESS the ID list has been assigned. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_string_scroll_wheel_string_list_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set
gx_string_scroll_wheel_string_list_set

Assign array of strings

Prototype

UINT gx_string_scroll_wheel_string_list_set(
    GX_STRING_SCROLL_WHEEL *wheel,
    GX_CONST GX_CHAR *string_list, INT string_count)

Description

This assigns an array of strings to a string scroll wheel widget. This can be used to modify the strings displayed after the widget has initially been created.

Note that string_scroll_wheel does not support GX_STYLE_TEXT_COPY, and therefore the array of strings passed into this function should be statically defined by the application.

Parameters

wheel String scroll wheel control block address
string_list Array of string pointers
string_count Size of the string array.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully changed strings for scroll wheel</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22) Invalid string list size</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_CONST GX_CHAR *days[] = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};

GX_STRING_SCROLL_WHEEL wheel;

/* Set the array of strings to the scroll wheel. */
status = gx_string_scroll_wheel_string_list_set(&wheel, days, 7);

/* If status is GX_SUCCESS the string array has been assigned. */

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_string_scroll_wheel_string_list_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set,
gx_text_scroll_wheel_text_color_set
**gx_studio_widget_create**

Create widget defined in Studio generated specifications file

**Prototype**

```c
GX_WIDGET *gx_studio_widget_create(GX_BYTE *control,
                                     GX_CONST GX_STUDIO_WIDGET *definition,
                                     GX_WIDGET *parent);
```

**Description**

This service creates a widget and the widget’s children using a widget specification defined within the GUIX Studio generated specifications file. This function avoids the “by name” lookup of the similar function `gx_studio_named_widget_create()`.

The `GX_STUDIO_WIDGET` structure is defined in the application specifications header file generated by GUIX Studio.

For statically allocated widgets, the widget control block is defined in the generated specifications.c file, and given the widget name defined within GUIX Studio. For dynamically allocated widgets, the application should pass `GX_NULL` as the widget control block address and the function will attempt to dynamically allocate the widget control block using the `gx_system_memory_allocate()` function, which is also defined by and provided by the application.

For an application to directly reference the GUIX Studio widget definition within the generated specifications file, it is necessary to follow the naming convention utilized by the GUI Studio code generator. The `GX_STUDIO_WIDGET` structure generated within the specifications.c file is always named according to this convention: `<widget_name>_define`, where the `<widget_name>` field may be repeated multiple times if the widget is child of a child widget.

**Parameters**

- **control**: Pointer to widget control block, or `GX_NULL` if dynamically allocated.
- **definition**: Studio generated widget definition structure
- **parent**: Pointer to the widget parent, if any
Return Values

Pointer to the created widget control block, or GX_NULL if the creation was not successful.

Allowed From

Initialization and threads

Example

/* Create the widget "playlist_screen", which is statically allocated. */

widget = gx_studio_widget_create(&playlist_screen, 
    &playlist_screen_define, 
    root_window);

/* If widget != GX_NULL the widget was created. */

/* create the widget "songs_screen", which is dynamically allocated */

widget = gx_studio_widget_create(GX_NULL, &songs_screen_define, 
    root_window);

See Also

gx_studio_named_widget_create
**gx_studio_named_widget_create**

Create widget defined in Studio generated specifications file

**Prototype**

```c
UINT *gx_studio_named_widget_create(char *name, GX_WIDGET *parent, 
                                    GX_WIDGET **new_widget);
```

**Description**

This service creates a widget and the widget's children using a widget specification defined within the GUIX Studio generated specifications file.

This API function is used to create top-level screens using the screen name specified within the GUIX Studio application as the widget definition identifier.

**Parameters**

- **name**
  Screen name as defined within GUIX Studio application.

- **parent**
  pointer to the widget parent, if any

- **new_widget**
  location to return created widget pointer

**Return Values**

- **GX_SUCCESS**
  (0x00)  Successful
- **GX_FAILURE**
  (0x11)  Named widget could not be found

**Allowed From**

Initialization and threads
Example

    /* Create the widget named “child_popup”, which is a child of the
top-level screen “main_screen”. */
    GX_WIDGET *menu_screen;

    status = gx_studio_named_widget_create("main_menu", &root_window,
                                          &menu_screen);

    /* If status == GX_SUCCESS the screen was created and linked to the
    root window. */

See Also

    gx_studio_widget_create
**gx_studio_display_configure**

Configure display defined in GUIX Studio project

**Prototype**

```c
UINT *gx_studio_display_configure(UINT display,
                                 UINT (*driver)(GX_DISPLAY *),
                                 USHORT language, USHORT theme,
                                 GX_WINDOW_ROOT **return_root);
```

**Description**

This service initializes a GX_DISPLAY so that it is ready for use. This function consolidates the functions to initialize a GX_DISPLAY control block, create a canvas to fit the display, and create a root window for the canvas. This function also installs the language and resource theme requested after the display has been initialized.

This function consolidates the programming effort most commonly required to prepare a display for use. The function invokes the `gx_display_create()`, `gx_display_color_table_set`, `gx_display_font_table_set`, `gx_display_pixelmap_table_set`, `gx_system_language_table_set`, `gx_system_active_language_set`, `gx_system_scroll_appearance_set`, `gx_canvas_create`, and `gx_window_root_create` functions, all or some of which would otherwise be required by the application program.

**Parameters**

- **display**
  - Index into the display table, which corresponds to the display definitions in the Studio project file.

- **driver**
  - Pointer to display driver initialization function. This function is invoked to initialize the indirect function pointers of the GX_DISPLAY control block, as well as perform any required hardware setup.

- **language**
  - Initial language table index

- **language**
  - Initial theme index

- **root**
  - Pointer to variable in which to return root window address, or GX_NULL.
Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x11)</td>
<td>Display could not be initialized</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
/* Create the widget named "child_popup", which is a child of the top-level screen "main_screen" */
GX_WIDGET *menu_screen;

status = gx_studio_display_configure(MAIN_DISPLAY, my_driver_setup,
                                        LANGUAGE_ENGLISH, DEFAULT_THEME, GX_NULL);

/* If status == GX_SUCCESS the display was initialized, a canvas was created for the display, a root window was created for the canvas, and the requested language and theme have been installed. */
```

See Also

`gx_display_create`, `gx_display_color_table_set`, `gx_display_font_table_set`,
`gx_display_pixelmap_table_set`, `gx_system_language_table_set`,
`gx_system_active_language_set`, `gx_system_scroll_appearance_set`,
`gx_canvas_create`, `gx_window_root_create`
**gx_system_active_language_set**

Set active language

**Prototype**

```c
UINT  gx_system_active_language_set(GX_UBYTE language);
```

**Description**

This service set the current language. The language index must be less than the number of columns in the application string table. This function has been deprecated in favor of `gx_display_active_language_set`. All new applications should use `gx_display_active_language_set`.

**Parameters**

- **language** Language index, defined in resource header file.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set active language
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_VALUE** (0x22) Invalid language index

**Allowed From**

Initialization and threads

**Example**

```c
/* Set active language and mark widget canvas as dirty. */
status = gx_system_active_language_set(ID_LANGUAGE_ENGLISH);

/* If status is GX_SUCCESS the active language has been assigned. */
```

**See Also**

- `gx_display_language_table_set`
- `gx_display_active_language_set`
- `gx_display_string_get`
gx_system_animation_get

Obtain animation control block from system pool

Prototype

UINT  gx_system_animation_get(GX_ANIMATION **animation);

Description

This service can be used to obtain an animation control block from a pool of such control blocks maintained by the gx_system component. The animation control block pool and related API services are only provided if the constant GX_ANIMATION_POOL_SIZE is defined with a value > 0. The default setting for this value is 6, meaning that the system animation control block pool contains size GX_ANIMATION control block.

An animation control block allocated using this API is automatically returned to the free pool if the animation runs to completion. If the animation is stopped using gx_animation_stop, or fails to be started due to some returned error, the animation control block should be returned to the free pool by the application using gx_system_animation_free.

Parameters

animation  Address of pointer to receive GX_ANIMATION pointer.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully obtained animation control block</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid animation pointer</td>
</tr>
<tr>
<td>GX_OUT_OF_ANIMATIONS (0x31)</td>
<td>System animation pool exhausted</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_ANIMATION *animation;

UINT status = gx_system_animation_get(&animation);

if (status == GX_SUCCESS)
{
    gx_animation_start(animation, animation_info);
}

See Also

gx_animation_create, gx_animation_start, gx_animation_stop,
gx_system_animation_free
**gx_system_animation_free**

Return an animation control block to the system pool

**Prototype**

```c
UINT  gx_system_animation_free(GX_ANIMATION *animation);
```

**Description**

This service can be used to return an animation control block to the system pool. The animation control block pool and related API services are only provided if the constant `GX_ANIMATION_POOL_SIZE` is defined with a value > 0. The default setting for this value is 6, meaning that the system animation control block pool contains size `GX_ANIMATION` control block.

An animation control block allocated using `gx_system_animation_get()` is automatically returned to the free pool if the animation runs to completion. Attempting to return an animation control block to the free pool that has already been returned to the free pool has no effect.

If the animation is stopped using `gx_animation_stop`, or fails to be started due to some returned error, the animation control block that has been obtained using `gx_system_animation_get()` should be returned to the free pool by the application using `gx_system_animation_free()`.

An animation must be in IDLE state before it can be returned to the free pool. An animation is in the IDLE state when it has not been started, when it is stopped, or when it runs to completion.

**Parameters**

- **animation**  Pointer to the `GX_ANIMATION` control block.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td><code>GX_SUCCESS</code>  Successfully released animation block</td>
</tr>
<tr>
<td>0x11</td>
<td><code>GX_CALLER_ERROR</code>  Invalid caller of this function</td>
</tr>
<tr>
<td>0x07</td>
<td><code>GX_PTR_ERROR</code>  Invalid animation pointer</td>
</tr>
<tr>
<td>0x32</td>
<td><code>GX_INVALID_ANIMATION</code>  The animation is not IDLE, or it has not been allocated from the system pool</td>
</tr>
</tbody>
</table>
Allowed From

Initialization and threads

Example

GX_ANIMATION *animation;

UINT status = gx_system_animation_get(&animation);

if (status == GX_SUCCESS)
{
    status = gx_animation_start(animation, animation_info);

    if (status != GX_SUCCESS)
    {
        /* animation did not start, return it to the free pool */
        gx_system_animation_free(animation);
    }
}

See Also

gx_animation_create, gx_animation_start, gx_animation_stop,
gx_system_animation_get
gx_system_bidi_text_disable

Disable dynamic bi-directional text support

Prototype

UINT  gx_system_bidi_text_disable(VOID);

Description

This service disables dynamic bi-directional text support. This service requires GX_DYNAMIC_BIDI_TEXT_SUPPORT to be defined when building the GUIX library, and is only required if runtime re-ordering of BiDi string data is needed. Most applications utilize GUIX Studio to produce correctly reordered BiDi text strings.

Parameters

None

Return Values

GX_SUCCESS (0x00) Successfully disabled bidi text support

Allowed From

Initialization and threads

Example

/* GX_DYNAMIC_BIDI_TEXT_SUPPORT is defined. */
/* Diable bidi text support. */
status = gx_system_bidi_text_disable();
/* If status is GX_SUCCESS, bidi text support was disabled. */

See Also

gx_system_bidi_text_enable
gx_system_bidi_text_enable

Enable dynamic bidi text support

Prototype

UINT gx_system_bidi_text_enable(VOID);

Description

This service enables dynamic bi-directional text support. This service requires GX_DYNAMIC_BIDI_TEXT_SUPPORT to be defined when building the GUIX library, and is only required if runtime re-ordering of BiDi string data is needed. Most applications utilize GUIX Studio to produce correctly reordered BiDi text strings.

Parameters

None

Return Values

GX_SUCCESS (0x00) Successfully enabled bidi text support

Allowed From

Initialization and threads

Example

/* GX_DYNAMIC_BIDI_TEXT_SUPPORT is defined. */

/* Enable bidi text support. */
status = gx_system_bidi_text_enable();

/* If status is GX_SUCCESS, bidi text support was enabled. */

See Also

gx_system_bidi_text_disable
gx_system_canvas_refresh

Refresh all dirty canvases

Prototype

UINT  gx_system_canvas_refresh(VOID);

Description

This service forces an immediate redrawing of all dirty widgets and canvases. This service is normally invoked internally by the GUIX system component, but can be called by the application to force an immediate system redrawing operation.

Parameters

None

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS(0x00)</td>
<td>Successfully released animation block</td>
</tr>
<tr>
<td>GX_INVALID_CANVAS(0x20)</td>
<td>No canvas created</td>
</tr>
<tr>
<td>GX_CALLER_ERROR(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Force immediate redraw operation. */
status = gx_system_canvas_refresh();
/* If status is GX_SUCCESS, canvas has been redraw. */

See Also

gx_system_active_language_set, gx_system_dirty_mark,
gx_system_dirty_partial_add, gx_system_draw_context_get,
gx_system_event_fold, gx_system_event_send, gx_system_focus_claim,
gx_system_initialize, gx_system_initialize, gx_system_language_table_get,
gx_system_language_table_set, gx_system_memory_allocator_set,
gx_system_scroll_appearance_get, gx_system_scroll_appearance_get,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
gx_system_dirty_mark

Mark area dirty

Prototype

UINT gx_system_dirty_mark(GX_WIDGET *widget);

Description

This service marks the area of this widget as dirty. This effectively queues the widget for re-drawing when the system event processing has been completed.

Parameters

widget Pointer to widget control block

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully marked widget dirty</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET (0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Mark widget "my_widget" as dirty. */
status = gx_system_dirty_mark(&my_widget);

/* If status is GX_SUCCESS the area associated with "my_widget" has been marked as dirty. */
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_partial_add, gx_system_draw_context_get,
gx_system_event_fold, gx_system_event_send, gx_system_focus_claim,
gx_system_initialize, gx_system_initialize, gx_system_language_table_get,
gx_system_language_table_set, gx_system_memory_allocator_set,
gx_system_scroll_appearance_get, gx_system_scroll_appearance_get,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_dirty_partial_add**

Mark partial area dirty

**Prototype**

```c
UINT gx_system_dirty_partial_add(GX_WIDGET *widget,
GX_RECTANGLE *dirty_area);
```

**Description**

This service marks the partial area of this widget as dirty. This queues the widget for re-drawing by the GUIX canvas refresh operation when the system event processing has been completed.

**Parameters**

- **widget**
  Pointer to widget control block

- **dirty_area**
  Dirty area of widget to mark dirty

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful partial dirty area mark</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>Invalid size of dirty area</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
/* Mark widget “my_widget” partial area as dirty. */
status = gx_system_dirty_partial_add(&my_widget, &partial_area);

/* If status is GX_SUCCESS the partial area “partial_area” associated with “my_widget” has been marked as dirty. */
```
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_draw_context_get, gx_system_event_fold,
gx_system_event_send, gx_system_focus_claim, gx_system_initialize,
gx_system_initialize, gx_system_language_table_get,
gx_system_language_table_set, gx_system_memory_allocator_set,
gx_system_scroll_appearance_get, gx_system_scroll_appearance_get,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_draw_context_get**

Get drawing context

**Prototype**

```c
UINT gx_system_draw_context_get(GX_DRAW_CONTEXT **current_context);
```

**Description**

This service returns a pointer to the current drawing context.

**Parameters**

- **current_context**
  Pointer to destination for current drawing context pointer

**Return Values**

- **GX_SUCCESS** (0x00) Successful current context get
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_DRAW_CONTEXT *current_context;
/* Get current drawing context. */
status = gx_system_draw_context_get(&current_context);
/* If status is GX_SUCCESS the current drawing context is contained in “current_context”. */
```

**See Also**

- `gx_system_active_language_set`, `gx_system_canvas_refresh`
- `gx_system_dirty_mark`, `gx_system_dirty_partial_add`, `gx_system_event_fold`
- `gx_system_event_send`, `gx_system_focus_claim`, `gx_system_initialize`
- `gx_system_initialize`, `gx_system_language_table_get`
- `gx_system_language_table_set`, `gx_system_memory_allocator_set`
- `gx_system_scroll_appearance_get`, `gx_system_scroll_appearance_get`
- `gx_system_start`, `gx_system_string_get`, `gx_system_string_table_get`
- `gx_system_string_width_get`, `gx_system_timer_start`, `gx_system_timer_stop`
- `gx_system_pen_configure`, `gx_system_version_string_get`
- `gx_system_widget_find`
# gx_system_event_fold

## Send event

### Prototype

```c
UINT gx_system_event_fold(GX_EVENT *event);
```

### Description

This service searches the GUIX event queue for an event of the same type. If an event of the same type exists, the event payload is updated to match the new event. If no matching event is found, the `gx_system_event_send` function is called to add the new event to the end of the event queue.

This function is commonly used by fast touch input drivers to prevent filling the event queue with multiple PEN_DRAG events. This function can also be called by the application to prevent multiple events of the same type from being added to the GUIX event queue.

### Parameters

- **event** Pointer to event

### Return Values

- **GX_SUCCESS** (0x00) Successful event send
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

### Allowed From

- Initialization and threads

### Example

```c
GX_EVENT my_event;

memset(&my_event, 0, sizeof(GX_EVENT));

my_event.gx_event_type = GX_EVENT_PEN_DOWN;
my_event.gx_event_payload.gx_event_pointdata.gx_point_x = 100;
my_event.gx_event_payload.gx_event_pointdata.gx_point_y = 200;

/* Send “my_event” for processing. */
status = gx_system_event_fold(&my_event);

/* If status is GX_SUCCESS the event has been sent for processing. */
```
See Also

- `gx_system_active_language_set`
- `gx_system_canvas_refresh`
- `gx_system_dirty_mark`
- `gx_system_draw_context_get`
- `gx_system_event_send`
- `gx_system_focus_claim`
- `gx_system_initialize`
- `gx_system_language_table_get`
- `gx_system_language_table_set`
- `gx_system_memory_allocator_set`
- `gx_system_scroll_appearance_get`
- `gx_system_start`
- `gx_system_string_get`
- `gx_system_string_table_get`
- `gx_system_string_width_get`
- `gx_system_timer_start`
- `gx_system_timer_stop`
- `gx_system_pen_configure`
- `gx_system_version_string_get`
- `gx_system_widget_find`
gx_system_event_send

Send event

Prototype

UINT gx_system_event_send(GX_EVENT *event);

Description

This service sends the specified event into the GUIX system event queue. The new event is placed at the end of the queue.

Parameters

- **event**  
  Pointer to event

Return Values

- **GX_SUCCESS**  
  (0x00)  
  Successful event send
- **GX_CALLER_ERROR**  
  (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  
  Invalid pointer

Allowed From

Initialization and threads

Example

/* Send "new_event" for processing. */

GX_EVENT new_event;

new_event.gx_event_target = widget -> gx_widget_parent;
new_event.gx_event_type = MY_EVENT_TYPE;

/* Set optional param. */
new_event.gx_event_payload.xxxx = yyyy
new_event.gx_event_sender = widget->gx_widget_id;

/* Push the event to event pool. */
status = gx_system_event_send(&new_event);

/* If status is GX_SUCCESS the event has been sent for processing. */
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_focus_claim,
gx_system_initialize, gx_system_initialize, gx_system_language_table_get,
gx_system_language_table_set, gx_system_memory_allocator_set,
gx_system_scroll_appearance_get, gx_system_scroll_appearance_get,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_focus_claim**

**Claim focus**

**Prototype**

```c
UINT gx_system_focus_claim(GX_WIDGET *widget);
```

**Description**

This service claims the focus for the specified widget. If the widget did not previously have focus, it will receive a GX_EVENT_FOCUS_GAINED event.

**Parameters**

- `widget` Pointer to widget control block to claim focus

**Return Values**

- **GX_SUCCESS** (0x00) Successful focus claim
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_NO_CHANGE** (0x08) Widget already owns focus
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Claim focus for widget "my_widget". */
status = gx_system_focus_claim(&my_widget);

/* If status is GX_SUCCESS the focus has been claimed for "my_widget". */
```

**See Also**

- `gx_system_active_language_set`
- `gx_system_canvas_refresh`
- `gx_system_dirty_mark`
- `gx_system_dirty_partial_add`
- `gx_system_draw_context_get`
- `gx_system_event_fold`
- `gx_system_event_send`
- `gx_system_initialize`
- `gx_system_language_table_get`
- `gx_system_language_table_set`
- `gx_system_memory_allocator_set`
- `gx_system_scroll_appearance_get`
- `gx_system_scroll_appearance_get`
- `gx_system_start`
- `gx_system_string_get`
- `gx_system_string_table_get`
- `gx_system_string_width_get`
- `gx_system_timer_start`
- `gx_system_timer_stop`
- `gx_system_pen_configure`
- `gx_system_version_string_get`
- `gx_system_widget_find`
gx_system_initialize

Initialize GUIX

Prototype

UINT gx_system_initialize(VOID);

Description

This service initializes GUIX. This service must be invoked before any other GUIX API service, and should only be invoked once at system startup.

Parameters

None

Return Values

- **GX_SUCCESS** (0x00): Successful system initialize
- **GX_SYSTEM_ERROR** (0xFE): Invalid GX_EVENT control block size or event queue/mutex/thread create failed.
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function

Allowed From

Initialization and threads

Example

```c
/* Initialize GUIX. */
status = gx_system_initialize();

/* If status is GX_SUCCESS, GUIX has been initialized. */
```

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_language_table_get,
gx_system_language_table_set, gx_system_memory_allocator_set,
gx_system_scroll_appearance_get, gx_system_scroll_appearance_set,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start,
gx_system_timer_stop, gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_language_table_get**

Retrieve active language table

**Prototype**

```c
UINT gx_system_language_table_get(
    GX_CHAR ****language_table,
    GX_UBYTE *languages_count, UINT *string_count);
```

**Description**

This service retrieves the active language table. This function is deprecated in favor of `gx_display_language_table_get`. All new applications should use `gx_display_language_table_get`.

**Parameters**

- **language_table** Address of pointer to return language table.
- **languages_count** Address of variable to return table columns.
- **string_count** Address of pointer to return table rows.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully retrieved active language table
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

Initialization and threads

**Example**

```c
GX_CHAR ***language_table;
GX_UBYTE language_count;
UINT string_count;

/* Retrieve the language table */
status = gx_system_language_table_get(&language_table,
    &language_count, &string_count);
```

**See Also**

- `gx_display_language_table_get`, `gx_display_language_table_set`
gx_system_language_table_set

Assign active language table

Prototype

UINT gx_system_language_table_set(GX_CHAR ***language_table, 
GX_UBYTE languages_count, UINT string_count);

Description

This service installs the active language table. This function has 
been deprecated in favor of gx_display_language_table_set. All new 
applications should use gx_display_language_table_set.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language_table</td>
<td>Pointer to language table.</td>
</tr>
<tr>
<td>languages_count</td>
<td>Number of languages in table.</td>
</tr>
<tr>
<td>string_count</td>
<td>Number of string table rows.</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successfully set language table
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

```c
/* Retrieve the language table */
status = gx_system_language_table_set(language_table, 
language_count, string_count);
```

See Also

- gx_display_language_table_set, gx_display_language_table_get,
- gx_display_active_language_set
**gx_system_memory_allocator_set**

Assign functions for memory allocation, de-allocation

**Prototype**

```c
UINT gx_system_memory_allocator_set(VOID *(allocate)(ULONG size),
                                  VOID(*release)(VOID *));
```

**Description**

This service assigns the application supplied callback function for dynamic memory allocation and de-allocation.

If no GUIX service that uses dynamic memory allocation is needed by the application, this service does not need to be called.

If used, this service should be called after `gx_system_initialize()` which clears the GUIX service pointers, and before any GUIX service that requires use of dynamical memory allocation.

GUIX services which require a runtime memory allocation and de-allocation service include:

- Loading binary resources from external storage into the GUIX runtime environment.
- The software runtime jpeg image decoder.
- The software runtime png image decoder.
- Using text widgets with GX_STYLE_TEXT_COPY.
- Runtime pixmap resize and rotation utility functions.
- Runtime screen and widget control block allocation.

**Parameters**

- **allocator**: Memory allocator function
- **release**: Memory free function

**Return Values**

- **GX_SUCCESS**: (0x00) Successfully assigned memory allocate function
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer

**Allowed From**
Initialization and threads

Example

The following example utilizes a ThreadX byte pool to implement a thread-safe dynamic memory allocation and memory de-allocation service.

```c
TX_BYTE_POOL memory_pool;
#define SCRATCHPAD_SIZE (1024 * 4)
ULONG scratchpad[SCRATCHPAD_SIZE];

/* define memory allocation service */
VOID *memory_allocate(ULONG size)
{
    VOID *memptr;
    if (tx_byte_allocate(&memory_pool, &memptr, size, TX_NO_WAIT) == TX_SUCCESS)
    {
        return memptr;
    }
    return NULL;
}

/* define memory de-allocation service */
void memory_free(VOID *mem)
{
    tx_byte_release(mem);
}

/* create byte pool and install our dynamic memory services with GUIX */
VOID tx_application_define(void *first_unused_memory)
{
    /* create byte pool for GUIX to use */
    tx_byte_pool_create(&memory_pool, "scratchpad", scratchpad,
                        SCRATCHPAD_SIZE * sizeof(ULONG));

    guix_setup();

    /* install our memory allocator and de-allocator */
    gx_system_memory_allocator_set(memory_allocate, memory_free);
}
```

See Also

- `gx_system_active_language_set`, `gx_system_canvas_refresh`,
- `gx_system_dirty_mark`, `gx_system_dirty_partial_add`,
- `gx_system_draw_context_get`, `gx_system_event_fold`, `gx_system_event_send`,
- `gx_system_focus_claim`, `gx_system_initialize`, `gx_system_initialize`,
- `gx_system_language_table_get`, `gx_system_language_table_set`,
- `gx_system_scroll_appearance_get`, `gx_system_scroll_appearance_get`,
- `gx_system_start`, `gx_system_string_get`, `gx_system_string_table_get`,
- `gx_system_string_width_get`, `gx_system_timer_start`, `gx_system_timer_stop`,
- `gx_system_pen_configure`, `gx_system_version_string_get`,
- `gx_system_widget_find`
gx_system_pen_configure

Set pen configuration

Prototype

UINT  gx_system_pen_configure(
    GX_PEN_CONFIGURATION *pen_configuration);

Description

This service sets pen configuration to control the pen speed and
distance parameters used to trigger the generation of
GX_EVENT_FLICK event types.

The gx_pen_configuration_min_drag_dist member of
GX_PEN_CONFIGURATION is a fixed point data type, and you
should use GX_FIXED_VAL_MAKE(value) to convert from INT to
GX_FIXED_VAL. For example, if you want to set minimum drag
distance to 0.5 pixel per tick, you have to set the
gx_pen_configuration_min_drag_dist to

GX_FIXED_VAL_MAKE(1) / 2.

In GUIX releases 5.4.0 and older, the
gx_pen_configuration_min_drag_dist member of
GX_PEN_CONFIGURATION was of (INT << 8) type rather than
GX_FIXED_VAL type. If your project with 5.4.0 version GUIX library
is using this API, you will need to modify the min_drag_dist
parameter or #define GUIX_5_4_0_COMPATIBILITY when building
the GUIX library.

Parameters

pen_configuration  Pointer to pen configuration structure.

Appendix I contains definition to
GX_PEN_CONFIGURATION structure

Return Values

GX_SUCCESS  (0x00)  Successfully set pen
configuration

GX_CALLER_ERROR  (0x11)  Invalid caller of this function

Allowed From

Initialization and threads
Example

/* Define pen configuration, set minimum drag distance to 0.5 pixel per tick, maximum pen speed to 10 ticks. That means GUIX will trigger a vertical or horizontal flick event if the drag time is smaller than 10 ticks and the drag distance is bigger than 0.5 * drag_ticks. */

GX_PEN_CONFIGURATION pen_configuration;

#if defined(GUIX_5_4_0_COMPATIBILITY)
    Pen_configuration.gx_pen_configuration_min_drag_dist = (1 << 8)/ 2;
#else
    pen_configuration.gx_pen_configuration_min_drag_dist = GX_FIXED_VAL_MAKE(1) / 2;
#endif

pen_configuration.gx_pen_configuration_max_pen_speed_ticks = 10;

/* Set the pen configuration. */
status = gx_system_pen_configure(&pen_configuration);

/* If status is GX_SUCCESS the touch configure has been set. */
**gx_system_screen_stack_create**

Create and initialize the system screen stack

**Prototype**

```c
UINT gx_system_screen_stack_create(GX_WIDGET **memory, INT size);
```

**Description**

This service defines a memory pool to be used for the system screen stack. The system screen stack is an optional feature than can be used by the application to manage application screen flow. appearance.

If the application intends to utilize the screen stack services, the `gx_system_screen_stack_create` function must first be called to setup the screen stack memory region.

**Parameters**

- **memory**
  - Pointer to the reserved memory block.

- **size**
  - Size, in bytes, of the reserved memory block.

**Return Values**

- **GX_SUCCESS** (0x00)  
  - Successfull creation
- **GX_CALLER_ERROR** (0x11)  
  - Invalid caller of this function
- **GX_PTR_ERROR** (0x07)  
  - Invalid pointer

**Allowed From**

Initialization and threads
Example

#define SCREEN_STACKDEPTH 8
GX_WIDGET *screen_stack[SCREEN_STACKDEPTH * 2];
UINT status;

/* Get the scrollbar appearance. */
status = gx_system_screen_stack_create(screen_stack,
sizeof(GX_WIDGET *) * SCREEN_STACKDEPTH * 2);

/* If status is GX_SUCCESS the system screen stack is initialized
and ready for use. */

See Also

gx_system_screen_stack_get, gx_system_screen_stack_pop,
gx_system_screen_stack_push, gx_system_screen_stack_reset
**gx_system_screen_stack_get**

Pop the topmost screen stack pointers

Prototype

```c
UINT gx_system_screen_stack_get(GX_WIDGET **popped_parent, 
                              GX_WIDGET **popped_screen);
```

Description

This function pops the topmost screen stack pointers and returns those pointers to the caller. This function differs from `gx_system_screen_stack_pop()` in that the popped screen is not automatically re-attached to the previous parent. Instead, the pointers are popped from the stack and returned to the caller, allowing the caller to attach the or discard the returned screen as desired.

Parameters

- **popped_parent** Location to store the parent widget pointer.
- **popped_screen** Location to store the popped screen pointer.

Return Values

- **GX_SUCCESS** (0x00) Successful retrieval of screen stack pointers
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_FAILURE** (0x10) Invalid or empty screen stack

Allowed From

- Initialization and threads

Example

```c
UINT status;
GX_WIDGET *parent_screen;
GX_WIDGET *popped_screen;

/* Pop a screen stack entry. */
status = gx_system_screen_stack_get(&parent_screen, 
                                  &popped_screen);
```
/* If status is GX_SUCCESS, parent_screen and popped_screen hold the topmost screen stack pointers. */

See Also

gx_system_screen_stack_create, gx_system_screen_stack_pop,
gx_system_screen_stack_push, gx_system_screen_stack_reset
**gx_system_screen_stack_pop**

Pop the topmost entry from the system screen stack

**Prototype**

```c
UINT gx_system_screen_stack_pop();
```

**Description**

This function pops the topmost entry from the screen stack and automatically attaches the popped screen to the popped parent widget.

**Parameters**

none

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x10)</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
UINT status;

/* Pop a screen stack entry. */
status = gx_system_screen_stack_pop();

/* If status is GX_SUCCESS, the topmost screen stack entry has been popped from the stack and re-attached to the previous parent. */
```

**See Also**

gx_system_screen_stack_get, gx_system_screen_stack_create, gx_system_screen_stack_push, gx_system_screen_stack_reset
gx_system_screen_stack_push

Push a widget and parent pointer to the screen stack

Prototype

UINT  gx_system_screen_stack_push(GX_WIDGET *screen)

Description

This service places a pointer to the indicated widget, which is usually a top-level screen, onto the screen stack. If the widget has a parent it is detached from the parent. The parent widget pointer is also pushed to the screen stack. The parent widget may be NULL, meaning a screen that is not visible or attached to any parent may be pushed onto the screen stack. If a widget with no parent is pushed to the screen stack, the screen_stack_get() API should be used to retrieved the pushed screen pointer, rather than using the screen_stack_pop() API, which attempts to re-attached the popped widget to it's previous parent.

Parameters

screen  Pointer to the widget to be pushed to the screen stack.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfull get scrollbar appearance</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Get the scrollbar appearance. */
status = gx_system_screen_stack_push(window);

/* If status is GX_SUCCESS, the widget pointed to by “window” has
been pushed to the screen stack, along with the widget’s parent
pointer. */

See Also

gx_system_screen_stack_get, gx_system_screen_stack_pop,
gx_system_screen_stack_create, gx_system_screen_stack_reset
\textbf{gx\_system\_screen\_stack\_reset}  \hfill Reset the system screen stack

\textbf{Prototype}

\texttt{UINT \ gx\_system\_screen\_stack\_reset();}

\textbf{Description}

This function removes all entries from the system screen stack. If the screens popped from the stack have dynamically allocated control blocks allocated by GUIX Studio, the memory for those control blocks is freed.

\textbf{Parameters}

none

\textbf{Return Values}

\begin{itemize}
  \item \texttt{GX\_SUCCESS} (0x00) Successfull get scrollbar appearance
  \item \texttt{GX\_CALLER\_ERROR} (0x11) Invalid caller of this function
  \item \texttt{GX\_PTR\_ERROR} (0x07) Invalid pointer
\end{itemize}

\textbf{Allowed From}

Initialization and threads

\textbf{Example}

\begin{verbatim}
/* Get the scrollbar appearance. */
status = \ gx\_system\_screen\_stack\_reset();

/* If status is GX_SUCCESS the system screen stack has been cleared of entries. */
\end{verbatim}

\textbf{See Also}

\texttt{gx\_system\_screen\_stack\_get, gx\_system\_screen\_stack\_pop,}
\texttt{gx\_system\_screen\_stack\_push, gx\_system\_screen\_stack\_create}
**gx_system_scroll_appearance_get**

Get scroll appearance

**Prototype**

```c
UINT gx_system_scroll_appearance_get(ULONG style, 
       GX_SCROLLBAR_APPEARANCE *return_appearance);
```

**Description**

This service gets the scrollbar appearance.

**Parameters**

- **style**: Scrollbar style
  - GX_SCROLLBAR_HORIZONTAL
  - GX_SCROLLBAR_VERTICAL
- **return_appearance**: Pointer to destination for appearance. 
  *Appendix I* contains definition to 
  GX_SCROLLBAR_APPERANCE structure

**Return Values**

- **GX_SUCCESS** (0x00): Successfull get scrollbar appearance
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer

**Allowed From**

Initialization and threads
Example

GX_SCROLLBAR_APPEARANCE my_appearance;

/* Get the scrollbar appearance. */
status = gx_system_scroll_appearance_get(style, &my_appearance);

/* If status is GX_SUCCESS "my_appearance" now contains the scroll appearance. */

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialization,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_set,
gx_system_start, gx_system_string_get, gx_system_string_table_get,
gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_scroll_appearance_set**

Set scroll appearance

**Prototype**

```c
UINT gx_system_scroll_appearance_set(ULONG style,
    GX_SCROLLBAR_APPEARANCE *appearance);
```

**Description**

This service sets the default scroll appearance. When a scroll is created, this appearance structure is used unless the application provides a custom version.

**Parameters**

- **style**
  Scroll style
  - GX_SCROLLBAR_HORIZONTAL
  - GX_SCROLLBAR_VERTICAL

- **appearance**
  Pointer to appearance structure initialized with various scrollbar appearance attributes. Refer to Appendix I for the definition of the GX_SCROLLBAR_APPEARANCE structure.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set scroll appearance set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

**Allowed From**

- Initialization and threads
Example

GX_SCROLLBAR_APPEARANCE my_appearance;
memset(&my_appearance, 0, sizeof(GX_SCROLLBAR_APPEARANCE));
my_appearance.gx_scroll_width = 20;
my_appearance.gx_scroll_thumb_width = 18;
my_appearance.gx_scroll_thumb_color = GX_COLOR_ID_SCROLL_BUTTON;
my_appearance.gx_scroll_thumb_border_color = GX_COLOR_ID_SCROLL_BUTTON;
my_appearance.gx_scroll_button_color = GX_COLOR_ID_SCROLL_BUTTON;
my_appearance.gx_scroll_thumb_travel_min = 20;
my_appearance.gx_scroll_thumb_travel_max = 20;
my_appearance.gx_scroll_thumb_border_style = GX_STYLE_BORDER_THIN;

/* Set the scroll appearance. */
status = gx_system_scroll_appearance_set(GX_SCROLLBAR_VERTICAL,
                                        &my_appearance);

/* If status is GX_SUCCESS the scroll appearance has been set. */

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_start, gx_system_timer_stop, gx_system_pen_configure,
gx_system_version_string_get, gx_system_widget_find
gx_system_start

Prototype

UINT  gx_system_start(VOID);

Description

This service starts GUIX processing. Under normal circumstances this function never returns, but instead begins processing the GUIX event queue. When the GUIX event queue is empty, this service suspends the calling thread until new events arrive in the GUIX event queue.

Parameters

None

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful system start (0x00)</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function (0x11)</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```
/* Start GUIX. */
status =  gx_system_start();
/* If status is GX_SUCCESS . GUIX has been started. */
```

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_start, gx_system_timer_stop, gx_system_pen_configure,
gx_system_version_string_get, gx_system_widget_find
gx_system_string_get

Get string

Prototype

UINT gx_system_string_get(GX_RESOURCE_ID string_id,
GX_CHAR **return_string);

Description

This service gets the string for the specified resource ID, using the first defined display and the currently active language. This function has been deprecated in favor of gx_display_string_get. All new applications should use gx_display_string_get.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_id</td>
<td>String resource ID</td>
</tr>
<tr>
<td>return_string</td>
<td>Pointer to string destination pointer</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successful string get</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_RESOURCE_ID</td>
<td>0x33</td>
<td>Invalid resource ID</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Get the string associated with MY_STRING_ID. */
status = gx_system_string_get(MY_STRING_RESOURCE_ID, &my_string);

/* If status is GX_SUCCESS the string is contained in “my_string”. */

See Also

gx_display_string_get, gx_display_string_table_get,
gx_display_language_table_set
**gx_system_string_table_get**

Retrieves the string table

Prototype

```c
UINT gx_system_string_table_get(GX_UBYTE language,
                                GX_CHAR **string_table,
                                UINT *get_size);
```

Description

This service retrieves the string table for the requested language from the first display. This function has been deprecated in favor of `gx_display_string_table_get`. All new applications should use `gx_display_string_table_get`.

Parameters

- **language**
  Language index

- **string_table**
  Pointer to storage space of the string table pointer, or NULL if the caller does not need to get the pointer to the string table.

- **get_size**
  Pointer to the storage for the number of strings in string table, or NULL if the caller does not need to get the number of strings in the string table.

Return Values

- **GX_SUCCESS** (0x00) Successful string table get
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function

Allowed From

Initialization and threads
Example

/* Get the string table. */
CHAR **my_string_table;
UINT table_size;
status = gx_system_string_table_get(LANGUAGE_ID_ENGLISH,
    &my_string_table, &table_size);

/* If status is GX_SUCCESS . the pointer to the string table has
been obtained. */

See Also

gx_display_string_table_get, gx_display_string_get,
gx_display_active_language_set, gx_display_language_table_set
gx_system_string_width_get

Get string width (deprecated)

Prototype

UINT gx_system_string_width_get(GX_FONT *font, GX_CHAR *string, INT string_length, GX_VALUE *return_width);

Description

This service is deprecated in favor of gx_system_string_width_get_ext().

This service computes the display width of the supplied string in pixels using the specified font. If the string_length parameter is >= 0, only the request count of characters are included in the calculation. If the string_length parameter is -1, the entire string up to the NULL terminator is used in the calculation.

Parameters

- font: Pointer to string’s font
- string: Pointer to string
- string_length: Length of string
- return_width: Destination for width of string

Return Values

- GX_SUCCESS: (0x00) Successful string width get
- GX_PTR_ERROR: (0x07) Invalid pointer
- GX_CALLER_ERROR: (0x11) Invalid caller of this function
- GX_INVALID_FONT: (0x16) Invalid font
- GX_INVALID_STRING_LENGTH: (0x34) Invalid string length

Allowed From

Initialization and threads

Example

/* Get the string width of "my_string". */
status = gx_system_string_width_get(&my_font, &my_string, strlen(my_string), &my_width);

/* If status is GX_SUCCESS . "my_width" contains the string width. */
See Also

gx_system_string_width_get_ext
gx_system_string_width_get_ext

Get string width

Prototype

UINT  gx_system_string_width_get_ext(GX_FONT *font,
   GX_STRING *string,
   GX_VALUE *return_width);

Description

This service computes the display width of the supplied string in pixels using the specified font.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font</td>
<td>Pointer to string’s font</td>
</tr>
<tr>
<td>string</td>
<td>Pointer to string</td>
</tr>
<tr>
<td>return_width</td>
<td>Destination for width of string</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful string width get</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_INVALID_FONT</td>
<td>(0x16)</td>
<td>Invalid font</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>(0x34)</td>
<td>Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_STRING my_string;
my_string.gx_string_ptr = "Monday";
my_string.gx_string_length = strlen(my_string.gx_string_ptr);

/* Get the string width of "my_string". */
status =  gx_system_string_width_get_ext(&my_font, &my_string, &my_width);

/* If status is GX_SUCCESS . "my_width" contains the string width. */
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_timer_start, gx_system_timer_stop,
gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
**gx_system_timer_start**

Start timer

Prototype

```c
UINT  gx_system_timer_start(GX_WIDGET *owner, UINT timer_id,
                             UINT initial_ticks,
                             UINT reschedule_ticks);
```

Description

This service starts a timer for the specified widget. The constant `GX_MAX_ACTIVE_TIMERS` defines the maximum active timers supported. The default setting for this value is 32.

Parameters

- **owner**: Pointer to widget control block
- **timer_id**: ID of timer
- **initial_ticks**: Initial expiration ticks
- **reschedule_ticks**: Periodic expiration ticks

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful timer start</td>
</tr>
<tr>
<td>GX_OUT_OF_TIMERS</td>
<td>No more timers</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>Timer value(s) not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
/* Start a periodic timer for the widget "my_widget". */
status = gx_system_timer_start(&my_widget, MY_TIMER_ID, 10, 20);

/* If status is GX_SUCCESS, the timer for "my_widget" has been started. */
```
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_stop, gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
gx_system_timer_stop

Stop timer

Prototype

UINT gx_system_timer_stop(GW_WIDGET *owner, UINT timer_id);

Description

This service stops the timer with the specified timer_id associated with the calling widget. To stop all timers linked to a particular widget, the application can pass the timer_id value of 0.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Pointer to widget control block</td>
</tr>
<tr>
<td>timer_id</td>
<td>ID of timer, or 0 for all timers</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful timer stop</td>
</tr>
<tr>
<td>GX_NOT_FOUND</td>
<td>(0x09)</td>
<td>Timer ID not found</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Stop the periodic timer for the widget "my_widget". */
status = gx_system_timer_stop(&my_widget, MY_TIMER_ID);

/* If status is GX_SUCCESS . the timer for "my_widget" has been stopped. */
See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_start, gx_system_pen_configure, gx_system_version_string_get,
gx_system_widget_find
gx_system_version_string_get
Retrieve GUIX library version string (deprecated)

Prototype

UINT  gx_system_version_string_get(GX_CHAR **version);

Description

This service is deprecated in favor of
gx_system_version_string_get_ext().

This service retrieves the GUIX library version string.

Parameters

version  Pointer to return string value.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)   Successfully retrieved version string</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)   Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_CHAR *version;

/* get the library version string. */
status = gx_system_version_string_get(&version);

See Also

gx_system_version_string_get_ext()
gx_system_version_string_get_ext
Retrieves GUIX library version string

Prototype

UINT gx_system_version_string_get(GX_STRING *version);

Description

This service retrieves the GUIX library version string.

Parameters

version Pointer to return string value.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successfully retrieved version string</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>0x34</td>
<td>Invalid string length</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_STRING version;

/* get the library version string. */
status = gx_system_version_string_get_ext(&version);

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_start, gx_system_timer_stop, gx_system_pen_configure,
gx_system_widget_find
gx_system_widget_find

Find widget

Prototype

UINT  gx_system_widget_find(USHORT widget_id,
                             INT search_level,
                             GX_WIDGET **return_search_result);

Description

This service searches for the specified widget ID. Unlike gx_widget_find(), this function searches the children of all root windows defined in the system, meaning this is an exhaustive search of all visible widgets. If you know the parent of the widget you are searching for, use gx_widget_find() instead.

Parameters

widget_id

Widget ID to search for

search_level

Defines the recursive nesting level into which child widgets are searched. If this value is 0, only immediate children of each root window are searched. If this value is GX_SEARCH_DEPTH_INFINITE, the function nests down into all children searching for the requested widget ID. For any other value > 0, the search level defines how deeply nested this function will go searching for the requested widget ID.

return_search_result

Pointer to destination for widget found

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful widget search</td>
</tr>
<tr>
<td>GX_NOT_FOUND (0x09)</td>
<td>Widget ID not found</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

/* Search recursively from the top level widget for the widget with ID of MY_WIDGET_ID. */
status = gx_system_widget_find(MY_WIDGET_ID,
                              GX_SEARCH_DEPTH_INFINITY,
                              &my_widget);

/* If status is GX_SUCCESS, the search was successful and "my_widget" contains the pointer to the widget. */

See Also

gx_system_active_language_set, gx_system_canvas_refresh,
gx_system_dirty_mark, gx_system_dirty_partial_add,
gx_system_draw_context_get, gx_system_event_fold, gx_system_event_send,
gx_system_focus_claim, gx_system_initialize, gx_system_initialize,
gx_system_language_table_get, gx_system_language_table_set,
gx_system_memory_allocator_set, gx_system_scroll_appearance_get,
gx_system_scroll_appearance_get, gx_system_start, gx_system_string_get,
gx_system_string_table_get, gx_system_string_width_get,
gx_system_timer_start, gx_system_timer_stop, gx_system_pen_configure,
gx_system_version_string_get
**gx_text_button_create**

Create text button

Prototype

```c
UINT gx_text_button_create(GX_TEXT_BUTTON *text_button,
                         GX_CONST GX_CHAR *name,
                         GX_WIDGET *parent,
                         GX_RESOURCE_ID text_id,
                         ULONG style,
                         USHORT text_button_id,
                         GX_CONST GX_RECTANGLE *size);
```

Description

This service creates a text button widget.

GX_TEXT_BUTTON is derived from GX_BUTTON and supports all gx_button API services.

Parameters

- **text_button**: Pointer to text button control block
- **name**: Logical name of text button
- **parent**: Pointer to parent widget of the button
- **text_id**: Resource ID of text
- **style**: Text button style. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **text_button_id**: Application-defined ID of the text button
- **size**: Size of the button

Return Values

- **GX_SUCCESS** (0x00): Successful text button create
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_ALREADY_CREATED** (0x13): Widget already created
- **GX_INVALID_SIZE** (0x19): Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12): Parent widget not valid

Allowed From

- Initialization and threads
Example

GX_TEXT_BUTTON my_text_button;
GX_RECTANGLE size;

/* Define widget size. */
gx_utility_rectangle_define(&size, 0, 0, 100, 100);

/* Create text button "my_text_button". */
status = gx_text_button_create(&my_text_button, "my text button",
                            &my_parent_window, MY_TEXT_RESOURCE_ID,
                            GX_STYLE_BUTTON_TOGGLE, MY_TEXT_BUTTON_ID, &size);

/* If status is GX_SUCCESS, the text button "my_text_button" was created. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_icon_button_create, gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_text_button_color_set, gx_text_button_draw, gx_text_button_font_set,
gx_text_button_text_get, gx_text_button_text_set, gx_text_button_text_id_set
**gx_text_button_draw**

---

**Draw text button**

**Prototype**

```c
VOID  gx_text_button_draw(GX_TEXT_BUTTON *button);
```

**Description**

This service draws the text button. This service is normally called internally during canvas refresh, but can also be called from custom text button drawing functions.

**Parameters**

- **button**
  Pointer to text button control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom text button draw function. */

VOID my_text_button_draw(GX_TEXT_BUTTON *text_button)
{
    /* Call default text button draw. */
    gx_text_button_draw(text_button);

    /* Add your own drawing here. */
}
```

**See Also**

- `gx_button_background_draw`
- `gx_button_create`
- `gx_button_deselect`
- `gx_button_draw`
- `gx_button_event_process`
- `gx_button_select`
- `gx_icon_button_create`
- `gx_pixelmap_button_create`
- `gx_pixelmap_button_draw`
- `gx_text_button_create`
- `gx_text_button_color_set`
- `gx_text_button_font_set`
- `gx_text_button_text_get`
- `gx_text_button_text_set`
- `gx_text_button_text_id_set`
gx_text_button_font_set

Set the font to text button

Prototype

UINT gx_text_button_font_set(GX_TEXT_BUTTON *button,
                             GX_RESOURCE_ID font_id);

Description

This service assigns a font to the specified button.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>button</td>
<td>Pointer to text button control block</td>
</tr>
<tr>
<td>font_id</td>
<td>Resource ID fo the font</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successfully set the font</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>0x12</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Set the text button with the font ID MY_FONT. */
status = gx_text_button_font_set(&my_text_button, MY_FONT);

/* If status is GX_SUCCESS, the font of the text button
"my_text_button" was set to MY_FONT. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_icon_button_create, gx_pixelmap_button_create, gx_pixelmap_button_draw,
gx_text_button_create, gx_text_button_draw, gx_text_button_color_set,
gx_text_button_text_get, gx_text_button_text_set, gx_text_button_text_id_set
gx_text_button_text_color_set

Set text button color

Prototype

UINT  gx_text_button_text_color_set(GX_TEXT_BUTTON *text_button,
                                   GX_RESOURCE_ID normal_text_color_id,
                                   GX_RESOURCE_ID selected_text_color_id,
                                   GX_RESOURCE_ID disabled_text_color_id);

Description

This service sets the color of the text button.

Parameters

text_button       Pointer to text button control block
normal_text_color_id  Resource ID of normal text. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

selected_text_color_id  Resource ID of selected text. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

disabled_text_color_id  Resource ID of color for disabled text. Appendix A contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

Return Values

GX_SUCCESS  (0x00)  Successful text button color set
GX_CALLER_ERROR  (0x11)  Invalid caller of this function
GX_PTR_ERROR  (0x07)  Invalid pointer
GX_INVALID_WIDGET  (0x12)  Widget not valid

Allowed From

Initialization and threads
Example

/* Set the color of the text button “my_text_button”. */
status = gx_text_button_text_color_set(&my_text_button,
    GX_COLOR_ID_NORMAL_TEXT,
    GX_COLOR_ID_SELECTED_TEXT,
    GX_COLOR_ID_DISABLED_TEXT);

/* If status is GX_SUCCESS, the text color of “my_text_button” was set. */

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect,
gx_button_draw, gx_button_event_process, gx_button_select,
gx_icon_button_create, x_pixelmap_button_create, gx_pixelmap_button_draw,
gx_text_button_create, gx_text_button_draw, gx_text_button_font_set,
gx_text_button_text_get, gx_text_button_text_set, gx_text_button_text_id_set
gx_text_button_text_draw

Support function to draw button text

Prototype

VOID gx_text_button_text_draw(GX_TEXT_BUTTON *text_button)

Description

This support function draws the text portion of a text button. This function is called internally by gx_text_button_draw, and is provided as a separate API as a convenience for applications that define a custom button drawing function. Applications that want to customize the button background drawing can provide their custom drawing function, and invoke the gx_text_button_text_draw service as part of their custom drawing to draw the button text over the background.

Parameters

text_button Pointer to text button control block

Return Values

None

Allowed From

Threads
Example

/* Define a custom drawing function */

VOID my_button_draw(GX_TEXT_BUTTON *button)
{
    /* Insert code here to draw button background */

    /* Call support function to do text drawing */
    gx_text_button_text_draw(button);

    /* Draw child widgets */
    gx_widget_children_draw((GX_WIDGET *) button);
}

See Also

gx_button_background_draw, gx_button_create, gx_button_deselect, 
gx_button_draw, gx_button_event_process, gx_button_select, 
gx_icon_button_create, x_pixelmap_button_create, gx_pixelmap_button_draw, 
gx_text_button_create, gx_text_button_draw, gx_text_button_font_set, 
gx_text_button_text_color_set, gx_text_button_text_set, 
gx_text_button_text_id_set
**gx_text_button_text_get**

Get text from the text button (deprecated)

**Prototype**

```c
UINT gx_text_button_text_get(GX_TEXT_BUTTON *text_button,
                              GX_CHAR **return_text)
```

**Description**

This service is deprecated in favor of `gx_text_button_text_get_ext()`.

This service retrieves the specified string from the text button.

**Parameters**

- `text_button`: Pointer to text button control block
- `return_text`: Pointer to the string retrieved from the text button

**Return Values**

- **GX_SUCCESS** (0x00): Successfully get the text from the button
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
GX_CHAR *string;

/* Get the string from the text button “my_text_button”. */
status = gx_text_button_text_get(&my_text_button, &string);

/* If status is GX_SUCCESS, the string pointer from “my_text_button” is retrieved and stored in string. */
```

**See Also**

`gx_text_button_text_get_ext`
### gx_text_button_text_get_ext

Get text from the text button

#### Prototype

```c
UINT gx_text_button_text_get_ext(GX_TEXT_BUTTON *text_button,
GX_STRING *return_string)
```

#### Description

This service retrieves the specified string from the text button.

#### Parameters

- **text_button**: Pointer to text button control block
- **return_string**: Pointer to the string retrieved from the text button

#### Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully get the text from the button</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET (0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

#### Allowed From

Initialization and threads

#### Example

```c
GX_STRING string;

/* Get the string from the text button "my_text_button". */
status = gx_text_button_text_get_ext(&my_text_button, &string);

/* If status is GX_SUCCESS, the string pointer and length from "my_text_button" is retrieved and stored in string. */
```

#### See Also

- `gx_button_background_draw`
- `gx_button_create`
- `gx_button_deselect`
- `gx_button_draw`
- `gx_button_event_process`
- `gx_button_select`
- `gx_icon_button_create`
- `x_pixelmap_button_create`
- `gx_pixelmap_button_draw`
- `gx_text_button_create`
- `gx_text_button_draw`
- `gx_text_button_font_set`
- `gx_text_button_text_color_set`
- `gx_text_button_text_set`
- `gx_text_button_text_id_set`
**gx_text_button_text_id_set**

Set text resource ID to the text button

**Prototype**

```c
UINT gx_text_button_text_id_set(GX_TEXT_BUTTON *text_button,
                                RESOURCE_ID string_id)
```

**Description**

This service sets the specified string resource ID to the text button.

**Parameters**

- `text_button` Pointer to text button control block
- `string_id` Resource ID of the string

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully set the string resource ID to the text button</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_RESOURCE_ID</td>
<td>(0x33) String ID not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the string ID "MY_STRING_ID" to the text button "my_text_button". */
status = gx_text_button_text_id_set(&my_text_button, MY_STRING_ID);

/* If status is GX_SUCCESS, the string ID MY_STRING_ID was set to "my_text_button". */
```

**See Also**

- `gx_button_background_draw`, `gx_button_create`, `gx_button_deselect`
- `gx_button_draw`, `gx_button_event_process`, `gx_button_select`
- `gx_icon_button_create`, `x_pixelmap_button_create`, `gx_pixelmap_button_draw`
- `gx_text_button_create`, `gx_text_button_draw`, `gx_text_button_font_set`
- `gx_text_button_text_color_set`, `gx_text_button_text_get`
**gx_text_button_text_set**

Assign text to the text button (deprecated)

**Prototype**

```c
UINT  gx_text_button_text_set(GX_TEXT_BUTTON *text_button,
                             GX_CHAR *text)
```

**Description**

This service is deprecated in favor of `gx_text_button_text_set_ext()`.

This service assigns the specified string to the text button. If the text_button widget was created with style `GX_STYLE_TEXT_COPY`, the widget creates a private copy of the text string assigned. If `GX_STYLE_TEXT_COPY` is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text_button</td>
<td>Pointer to text button control block</td>
</tr>
<tr>
<td>text</td>
<td>pointer to the NULL-terminated string</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully set the text to the button</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
<tr>
<td>GX_SYSTEM_MEMORY_ERROR</td>
<td>(0x30) Memory allocator not defined or memory allocation failed</td>
</tr>
<tr>
<td>GX_INVALID_STRING_LENGTH</td>
<td>(0x34) Invalid string length</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads
Example

/* Set the string “my string” to the text button “my_text_button”. */
status = gx_text_button_text_set(&my_text_button, “my string”);

/* If status is GX_SUCCESS, the string “my_text_button” was set. */

See Also

gx_text_button_text_set_ext, gx_text_button_text_id_set
**gx_text_button_text_set_ext**

Assign text to the text button

**Prototype**

```
UINT gx_text_button_text_set_ext(GX_TEXT_BUTTON *text_button, GX_STRING *string)
```

**Description**

This service assigns the specified string to the text button. If the `text_button` widget was created with style `GX_STYLE_TEXT_COPY`, the widget creates a private copy of the text string assigned. If `GX_STYLE_TEXT_COPY` is not active, the widget does not make a private copy of the incoming string, and therefore the string must be statically or globally allocated, i.e. it may not be an automatic or temporary variable.

**Parameters**

- **text_button**: Pointer to text button control block
- **string**: pointer to the GX_STRING variable

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set the text to the button
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid
- **GX_SYSTEM_MEMORY_ERROR** (0x30) Memory allocator not defined or memory allocation failed
- **GX_INVALID_STRING_LENGTH** (0x34) Invalid string length

**Allowed From**

Initialization and threads
Example

GX_STRING new_string;
new_string.gx_string_ptr = "Monday";
new_string.gx_string_length = strlen(new_string.gx_string_ptr);

/* Assign the string "new_string" to the text button "my_text_button". */
status = gx_text_button_text_set_ext(&my_text_button, &new_string);

/* If status is GX_SUCCESS, the string "my_text_button" was set. */

See Also

  gx_button_background_draw, gx_button_create, gx_button_deselect,
  gx_button_draw, gx_button_event_process, gx_button_select,
  gx_icon_button_create, x_pixelmap_button_create, gx_pixelmap_button_draw,
  gx_text_button_create, gx_text_button_draw, gx_text_button_font_set,
  gx_text_button_text_color_set, gx_text_button_text_get,
  gx_text_button_text_id_set
**gx_text_input_cursor_blink_interval_set**

*Set cursor blink interval*

**Prototype**

```c
UINT gx_text_input_cursor_blink_interval_set(
    GX_TEXT_INPUT_CURSOR *cursor_input,
    GX_UBYTE blink_interval)
```

**Description**

This service sets blink interval value of the cursor.

**Parameters**

- `cursor_input`: Cursor control block
- `blink_interval`: Value to be set

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully set the cursor blink interval</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22) Blink interval value not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_TEXT_INPUT_CURSOR *input_cursor;

/* Pointer the input cursor to the cursor instance of single/multi line text input widget. */
input_cursor = &sl_input.gx_single_line_text_input_cursor_instance;

/* Set the blink interval value of ”input_cursor” to 2. */
status = gx_text_input_cursor_blink_interval_set(input_cursor, 2);

/* If status is GX_SUCCESS, the blink interval value of ”input_cursor” has been successfully set to 2. */
```

**See Also**

gx_text_input_cursor_height_set, gx_text_input_cursor_width_set
**gx_text_input_cursor_height_set**

Set cursor height

**Prototype**

UINT gx_text_input_cursor_height_set(GX_TEXT_INPUT_CURSOR *cursor_input, GX_UBYTE height)

**Description**

This service sets height of the cursor.

**Parameters**

<table>
<thead>
<tr>
<th>cursor_input</th>
<th>Cursor control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Value to be set</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>GX_SUCCESS</th>
<th>(0x00)</th>
<th>Successfully set cursor height</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>(0x22)</td>
<td>Height value not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_TEXT_INPUT_CURSOR *input_cursor;

/* Pointer the input cursor to the cursor instance of single/multi
 line text input widget. */
input_cursor = &sl_input.gx_single_line_text_input_cursor_instance;

/* Set height value of “input_cursor”. */
status = gx_text_input_cursor_height_set(&input_cursor, 15);

/* If status is GX_SUCCESS, the height value of “input_cursor” has
 been successfully set to 15. */
```

**See Also**

gx_text_input_cursor_blink_interval_set, gx_text_input_cursor_width_set
**gx_text_input_cursor_width_set**

Set cursor width

**Prototype**

```
UINT gx_text_input_cursor_blink_width_set(
    GX_TEXT_INPUT_CURSOR *cursor_input, GX_UBYTE *width)
```

**Description**

This service sets width of the cursor.

**Parameters**

- `cursor_input` : Cursor control block
- `width` : Value to be set

**Return Values**

- **GX_SUCCESS** (0x00) : Successfully set the cursor width
- **GX_PTR_ERROR** (0x07) : Invalid pointer
- **GX_INVALID_VALUE** (0x22) : Width value not valid

**Allowed From**

Initialization and threads

**Example**

```
GX_TEXT_INPUT_CURSOR *input_cursor;

/* Pointer the input cursor to the cursor instance of single/multi line text input widget. */
input_cursor = &sl_input.gx_single_line_text_input_cursor_instance;

/* Set width of “input_cursor” to 2. */
status = gx_text_input_cursor_blink_width_set(&input_cursor, 2);

/* If status is GX_SUCCESS, the width of “input_cursor” has been successfully set to 2. */
```

**See Also**

- `gx_text_input_cursor_blink_interval_set`
- `gx_text_input_cursor_height_set`
**gx_text_scroll_wheel_callback_set**

Assign the callback function of text type scroll wheel (deprecated)

**Prototype**

```c
UINT gx_text_scroll_wheel_callback_set(GX_TEXT_SCROLL_WHEEL *wheel,
                                      GX_CONST GX_CHAR *(callback)(GX_TEXT_SCROLL_WHEEL *, int))
```

**Description**

This service is deprecated in favor of
`gx_text_scroll_wheel_callback_set_ext()`.

This service assigns the callback function which a text type scroll wheel will invoke to determine the text string to be displayed at each row of the scroll wheel.

For `GX_NUMERIC_SCROLL_WHEEL` and `GX_STRING_SCROLL_WHEEL`, default callback functions are provided and the application does not need to make any changes to use these default implementations.

This API is provided to allow the application to customize the formatting or other parameters of the string that is displayed on each row of the scroll wheel widget.

The callback function will receive as input a pointer to the scroll wheel control block and the row number that is being displayed. The function should return a pointer to a text string.

**Parameters**

- **wheel**: String scroll wheel control block address
- **callback**: Pointer to callback function

**Return Values**

- `GX_SUCCESS` (0x00)  Successfully set callback
- `GX_PTR_ERROR` (0x07)  Invalid pointer
- `GX_INVALID_WIDGET` (0x12)  Widget not valid

**Allowed From**

Initialization and threads
Example

```c
GX_TEXT_SCROLL_WHEEL wheel;
GX_CHAR string_buffer[20];

GX_CHAR *my_wheel_callback(GX_TEXT_SCROLL_WHEEL *wheel, int row)
{
    /* Just for an example, return row number as string for rows 
       >= 0, and return text “Invalid” otherwise */
    if (row >= 0)
    {
        gx_utility_ltoa(row, string_buffer, 20);
    }
    else
    {
        return("Invalid");
    }
}

gx_text_scroll_wheel_create(&wheel, "my wheel", root, 10,
    GX_STYLE_ENABLED|GX_STYLE_TEXT_CENTER|GX_STYLE_TRANSPARENT|GX_STYLE_WRAP|ID_MY_WHEEL, &size);

status = gx_text_scroll_wheel_callback_set(&wheel,
    my_wheel_callback);

/* If status is GX_SUCCESS, the scroll wheel callback function has 
been set. */
```

See Also

- `gx_numeric_scroll_wheel_create`, `gx_numeric_scroll_wheel_range_set`,
- `gx_scroll_wheel_create`, `gx_scroll_wheel_event_process`,
- `gx_scroll_wheel_gradient_alpha_set`, `gx_scroll_wheel_row_height_set`,
- `gx_scroll_wheel_selected_background_set`, `gx_scroll_wheel_selected_get`,
- `gx_scroll_wheel_selected_set`, `gx_scroll_wheel_total_rows_set`,
- `gx_text_scroll_wheel_create`, `gx_text_scroll_wheel_draw`,
- `gx_text_scroll_wheel_font_set`, `gx_text_scroll_wheel_text_color_set`
gx_text_scroll_wheel_callback_set_ext

Assign the callback function of text type scroll wheel

Prototype

UINT gx_text_scroll_wheel_callback_set_ext(
    GX_TEXT_SCROLL_WHEEL *wheel,
    UINT *(*callback)(GX_TEXT_SCROLL_WHEEL *, int, GX_STRING *)
)

Description

This service assigns the callback function which a text type scroll wheel will invoke to determine the text string to be displayed at each row of the scroll wheel.

For GX_NUMERIC_SCROLL_WHEEL and GX_STRING_SCROLL_WHEEL, default callback functions are provided and the application does not need to make any changes to use these default implementations.

This API is provided to allow the application to customize the formatting or other parameters of the string that is displayed on each row of the scroll wheel widget.

The callback function will receive as input a pointer to the scroll wheel control block and the row number that is being displayed. The function should return a pointer to a text string.

Parameters

- **wheel**: String scroll wheel control block address
- **callback**: Pointer to callback function

Return Values

- **GX_SUCCESS** (0x00) Successfully set callback
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

Allowed From

Initialization and threads
Example

```c
GX_TEXT_SCROLL_WHEEL wheel;
GX_CHAR string_buffer[20];

UINT *my_wheel_callback(GX_TEXT_SCROLL_WHEEL *wheel,
                        int row,
                        GX_STRING *return_string)
{
    /* Just for an example, return row number as string for rows 
       >= 0, and return text “Invalid” otherwise */
    if (row >= 0)
    {
        gx_utility_ltoa(row, string_buffer, 20);
        return_string->gx_string_ptr = string_buffer;
        return_string->gx_string_length = strlen(string_buffer);
    }
    else
    {
        return_string->gx_string_ptr = "Invalid";
        return_string->gx_string_length = strlen("Invalid");
    }
    return GX_SUCCESS;
}

gx_text_scroll_wheel_create(&wheel, "my wheel", root, 10,
                          GX_STYLE_ENABLED|GX_STYLE_TEXT_CENTER|GX_STYLE_TRANSPARENT|
                          GX_STYLE_WRAP|ID_MY_WHEEL, &size);

status = gx_text_scroll_wheel_callback_set_ext(&wheel,
                        my_wheel_callback);

/* If status is GX_SUCCESS, the scroll wheel callback function has 
   been set. */
```

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_selected_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set
gx_text_scroll_wheel_create

Create a text scroll wheel

Prototype

UINT gx_text_scroll_wheel_create(GX_TEXT_SCROLL_WHEEL *wheel,
                                 GX_CONST GX_CHAR *name, GX_WIDGET *parent, INT total_rows,
                                 ULONG style, USHORT Id, GX_CONST GX_RECTANGLE *size)

Description

This service creates a text scroll wheel. The text scroll wheel is a base widget for the GX_STRING_SCROLL_WHEEL and GX_NUMERIC_SCROLL_WHEEL type widgets. This function is called internally by gx_string_scroll_wheel_create and gx_numeric_scroll_wheel_create, and is provided as a separate API as a convenience for applications that define a custom scroll wheel widget.

Parameters

wheel Text scroll wheel control block address
name Application defined widget name
parent Wheel parent or GX_NULL
total_rows Total rows to be presented to user
style Desired style flags
Id Application defined wheel style flags
size Initial scroll wheel size

Return Values

GX_SUCCESS (0x00) Successfully created text scroll wheel
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_ALREADY_CREATED (0x13) Widget already created
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads

Example

/* Define a custom scroll wheel widget. */

typedef MY_SCROLL_WHEEL_STRUCT{
  GX_TEXT_SCROLL_WHEEL text_scroll_wheel;
}
/* Add custom members here. */
MY_SCROLL_WHEEL;
MY_SCROLL_WHEEL my_scroll_wheel;

UINT my_scroll_wheel_create(MY_SCROLL_WHEEL *wheel,
    GX_CONST GX_CHAR *name, GX_WIDGET *parent,
    INT total_rows, ULONG style, USHORT Id,
    GX_CONST GX_RECTANGLE *size)
{
    /* Call base creation. */
    status = gx_text_scroll_wheel_create(
        &wheel.text_scroll_wheel,
        "my_text_scroll_wheel", GX_NULL, 7,
        GX_STYLE_ENABLED, ID_MY_SCROLL_WHEEL, &size);

    if (status == GX_SUCCESS)
    {
        /* Add custom initialization here. */
        If(parent)
        {
            gx_widget_link(parent, (GX_WIDGET *)wheel);
        }
    }
}

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_event_process, gx_scroll_wheel_gradient_alpha_set,
gx_scroll_wheel_row_height_set, gx_scroll_wheel_selected_background_set,
gx_scroll_wheel_selected_get, gx_scroll_wheel_selected_set,
gx_scroll_wheel_total_rows_set, gx_string_scroll_wheel_string_id_list_set,
gx_string_scroll_wheel_string_list_set, gx_text_scroll_wheel_callback_set,
gx_text_scroll_wheel_create, gx_text_scroll_wheel_draw,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set
gx_text_scroll_wheel_draw

Draw a text scroll wheel

Prototype

VOID gx_text_scroll_wheel_draw(GX_TEXT_SCROLL_WHEEL *wheel)

Description

This is the default drawing function for all wheel types based on GX_TEXT_SCROLL_WHEEL. This function can be overridden by applications that require customization of the text scroll wheel drawing appearance.

GX_STRING_SCROLL_WHEEL and GX_NUMERIC.Scroll_WHEEL are both based on or derived from GX_TEXT_SCROLL_WHEEL.

Parameters

wheel String scroll wheel control block address

Return Values

None

Allowed From

Initialization and threads
Example

/* Write a custom wheel draw function. */
UINT my_wheel_draw(GX_TEXT_SCROLL_WHEEL *wheel)
{
    /* Perform default drawing */
    gx_text_scroll_wheel_draw(wheel);

    /* Add custom drawing here */
}

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_selected_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_font_set, gx_text_scroll_wheel_text_color_set
gx_text_scroll_wheel_font_set
Assign fonts used to draw scroll wheel rows

Prototype

UINT gx_text_scroll_font_set(GX_TEXT_SCROLL_WHEEL *wheel, 
                GX_RESOURCE_ID normal_font, GX_RESOURCE_ID selected_font)

Description

Assign the fonts use to draw the text of a text scroll wheel based widget.

Parameters

wheel            String scroll wheel control block address

Return Values

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successfully assigned wheel font</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_NUMERIC_SCROLL_WHEEL wheel;

status = gx_text_scroll_wheel_font_set(&wheel,
        GX_FONT_ID_WHEEL_NORMAL,
        GX_FONT_ID_WHEEL_SELECTED);

/* If status is GX_SUCCESS, the scroll wheel fonts have been
assigned. */

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_selected_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_text_color_set
gx_text_scroll_wheel_text_color_set

Assign colors used to draw scroll wheel rows

Prototype

UINT gx_text_scroll_wheel_text_color_set(GX_TEXT_SCROLL_WHEEL *wheel,
GX_RESOURCE_ID normal_text_color,
GX_RESOURCE_ID selected_text_color,
GX_RESOURCE_ID disabled_text_color)

Description

This function assigns the text colors used to draw a text based scroll wheel rows.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wheel</td>
<td>String scroll wheel control block address</td>
</tr>
<tr>
<td>normal_text_color</td>
<td>Color used to draw non-selected rows</td>
</tr>
<tr>
<td>selected_text_color</td>
<td>Color used to draw selected row.</td>
</tr>
<tr>
<td>disabled_text_color</td>
<td>Color used to draw text for disabled widget.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully assigned scroll wheel text color</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_STRING_SCROLL_WHEEL wheel;

UINT status = gx_text_scroll_wheel_text_color_set(&wheel,
            GX_COLOR_ID_NORMAL_TEXT,
            GX_COLOR_ID_SELECTED_TEXT,
            GX_COLOR_ID_DISABLED_TEXT);

/* If status is GX_SUCCESS, the colors used to draw the wheel text
have been assigned. */

See Also

gx_numeric_scroll_wheel_create, gx_numeric_scroll_wheel_range_set,
gx_scroll_wheel_create, gx_scroll_wheel_event_process,
gx_scroll_wheel_gradient_alpha_set, gx_scroll_wheel_row_height_set,
gx_scroll_wheel_selected_background_set, gx_scroll_wheel_selected_get,
gx_scroll_wheel_selected_set, gx_scroll_wheel_total_rows_set,
gx_text_scroll_wheel_callback_set, gx_text_scroll_wheel_create,
gx_text_scroll_wheel_draw, gx_text_scroll_wheel_font_set
**gx_tree_view_create**

Create a tree view

**Prototype**

```c
UINT gx_tree_view_create(GX_TREE_VIEW *tree,
                          GX_CONST GX_CHAR *name, GX_WIDGET *parent,
                          ULONG style, USHORT tree_view_id,
                          GX_CONST GX_RECTANGLE *size);
```

**Description**

This service creates a tree view as specified and associates the tree view with the supplied parent widget. It accepts all types of widget as child menu item. It’s recommended to use GX_MENU type widget as its child menu item.

GX_TREE_VIEW is derived from GX_WINDOW and supports all gx_window API services.

**Parameters**

- **tree**
  Pointer to tree view control block
- **name**
  Name of the tree view
- **parent**
  Pointer to parent widget
- **style**
  Style of the widget. Appendix D contains pre-defined general styles for all widgets as well as widget specific styles.
- **menu_id**
  Application-defined ID of the tree view
- **size**
  Size of the tree view

**Return Values**

- **GX_SUCCESS** (0x00) Successful tree view creation
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12) Widget not valid
Allowed From

Initialization and threads

Example

```c
status = gx_tree_view_create(&my_tree_view, "my_tree_view", parent,
    GX_STYLE_ENABLED, MY_TREE_VIEW_ID,
    &size);

/* If status is GX_SUCCESS the tree view was successfully created. */
```

See Also

gx_menu_draw, gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set, gx_tree_view_draw, gx_tree_view_event_process,
gx_tree_view_indentation_set, gx_tree_view_position,
gx_tree_view_root_line_color_set, gx_tree_view_root_pixelmap_set,
gx_tree_view_selected_get, gx_tree_view_selected_set
**gx_tree_view_draw**

**Prototype**

VOID  
gx_tree_view_draw(GX_TREE_VIEW *tree);

**Description**

This service draws the specified tree view. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions for custom tree view widgets.

**Parameters**

- **tree**
  Pointer to tree view control block

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom tree view draw function. */
UINT my_tree_view_draw(GX_TREE_VIEW *tree_view)
{
    /* Perform default drawing */
    gx_tree_view_draw(tree_view);

    /* Add custom drawing here */
}
```

**See Also**

- gx_menu_draw, gx_menu_insert, gx_menu_remove, gx_menu_text_draw, gx_menu_text_offset_set, gx_tree_view_create, gx_tree_view_event_process, gx_tree_view_indentation_set, gx_tree_view_position, gx_tree_view_root_line_color_set, gx_tree_view_root_pixelmap_set, gx_tree_view_selected_get, gx_tree_view_selected_set
**gx_tree_view_event_process**  
Process tree view event

### Prototype

```c
UINT gx_tree_view_event_process(GX_TREE_VIEW *tree, GX_EVENT event_ptr);
```

### Description

This service processes an event for the specified tree view. This service should be called as the default event handler by any custom tree view event processing functions.

### Parameters

- **tree**  
  Pointer to tree view control block

- **event_ptr**  
  Pointer to the event to process

### Return Values

- **GX_SUCCESS**  
  (0x00) Successful process tree view event
- **GX_CALLER_ERROR**  
  (0x11) Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07) Invalid pointer
- **GX_INVALID_WIDGET**  
  (0x12) Widget not valid

### Allowed From

- Threads
Example

/* Call generic tree view event processing as part of custom event processing function. */

UINT custom_tree_view_event_process(GX_TREE_VIEW *tree_view,
                                   GX_EVENT *event)
{
    UINT status = GX_SUCCESS;
    switch(event->gx_event_type)
    {
        case xyz:
            /* Insert custom event handling here */
            break;
        default:
            /* Pass all other events to the default tree view event processing */
            status = gx_tree_view_event_process(tree_view, event);
            break;
    }
    return status;
}

See Also

gx_menu_draw, gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set, gx_tree_view_create, gx_tree_view_draw,
gx_tree_view_indentation_set, gx_tree_view_position,
gx_tree_view_root_line_color_set, gx_tree_view_root_pixelmap_set,
gx_tree_view_selected_get, gx_tree_view_selected_set
gx_tree_view_indentation_set

Set tree view indentation

Prototype

UINT  gx_tree_view_indentation_set(GX_TREE_VIEW *tree,
                                   GX_VALUE indentation);

Description

This service sets indentation for the tree view.

Parameters

tree Pointer to tree view control block
indentation Indentation to set

Return Values

GX_SUCCESS (0x00) Successfully set tree view indentation
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads

Example

/* Set tree view "my_tree" indentation to 10. */
status = gx_tree_view_indentation_set(&my_tree, 10);

/* If status is GX_SUCCESS the indentation of tree view "my_tree"
has been set to 10. */

See Also

gx_menu_draw, gx_menu_insert, gx_menu_remove, gx_menu_text_draw,
gx_menu_text_offset_set, gx_tree_view_create, gx_tree_view_draw,
gx_tree_view_event_process, tree_view_position,
gx_tree_view_root_line_color_set, gx_tree_view_root_pixmap_set,
gx_tree_view_selected_get, gx_tree_view_selected_set
**gx_tree_view_position**

Position tree view items

**Prototype**

```c
UINT  gx_tree_view_position(GX_TREE_VIEW *tree);
```

**Description**

This service positions tree view items.

**Parameters**

- `tree` Pointer to tree view control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully positioned tree view items
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Position tree view "my_tree" items. */
status = gx_tree_view_position(&my_tree);

/* If status is GX_SUCCESS the items of tree view "my_tree" has been positioned. */
```

**See Also**

- `gx_menu_draw`, `gx_menu_insert`, `gx_menu_remove`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`, `gx_tree_view_create`, `gx_tree_view_draw`,
- `gx_tree_view_event_process`, `gx_tree_view_indentation_set`,
- `gx_tree_view_root_line_color_set`, `gx_tree_view_rootPixmap_set`,
- `gx_tree_view_selected_get`, `gx_tree_view_selected_set`
**gx_tree_view_root_line_color_set**

Set tree view root line color

**Prototype**

```c
UINT gx_tree_view_root_line_color_set(GX_TREE_VIEW *tree, 
                                       GXRESOURCE_ID color_id);
```

**Description**

This service assigns root line color for the tree view.

**Parameters**

- **tree**
  Pointer to tree view control block
- **color_id**
  Resource id of root line color

**Return Values**

- **GX_SUCCESS** (0x00)
  Successful set root line color
- **GX_CALLER_ERROR** (0x11)
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)
  Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Set root line color for tree view "my_tree". */
status = gx_tree_view_root_line_color_set(&my_tree, 
                                         MY_ROOT_LINE_COLOR_ID);

/* If status is GX_SUCCESS the root line color of the tree view 
   "my_tree" has been set. */
```

**See Also**

- `gx_menu_draw`, `gx_menu_insert`, `gx_menu_remove`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`, `gx_tree_view_create`, `gx_tree_view_draw`,
- `gx_tree_view_event_process`, `gx_tree_view_indentation_set`,
- `gx_tree_view_position`, `gx_tree_view_root_pixelmap_set`,
- `gx_tree_view_selected_get`, `gx_tree_view_selected_set`
Set tree view root pixelmap

Prototype

```c
UINT gx_tree_view_root_pixelmap_set(GX_TREE_VIEW *tree,
    GX_RESOURCE_ID expand_map_id,
    GX_RESOURCE_ID collapse_map_id);
```

Description

This service assigns expand and collapse pixelmap for the tree view.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>Pointer to tree view control block</td>
</tr>
<tr>
<td>expand_map_id</td>
<td>Resource id of expand pixelmap</td>
</tr>
<tr>
<td>collapse_map_id</td>
<td>Resource id of collapse pixelmap</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00): Successfully set root pixelmap
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads

Example

```c
/* Set root pixelmaps for tree view "my_tree". */
status = gx_tree_view_root_pixelmap_set(&my_tree,
    MY_EXPAND_MAP_ID,
    MY_COLLAPSE_MAP_ID);

/* If status is GX_SUCCESS the root pixelmaps of tree view "my_tree" has been set. */
```

See Also

- `gx_menu_draw`, `gx_menu_insert`, `gx_menu_remove`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`, `gx_tree_view_create`, `gx_tree_view_draw`,
- `gx_tree_view_event_process`, `gx_tree_view_indentation_set`,
- `gx_tree_view_position`, `gx_tree_view_selected_get`, `gx_tree_view_selected_set`
**gx_tree_view_selected_get**

Get selected item

**Prototype**

```c
UINT gx_tree_view_selected_get(GX_TREE_VIEW *tree,
                               GX_WIDGET **selected);
```

**Description**

This service retrieves current selected item of the tree view.

**Parameters**

- `tree` Pointer to tree view control block
- `selected` Pointer to selected widget pointer

**Return Values**

- **GX_SUCCESS** (0x00) Successfully retrieved selected item
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Retrieve selected item of tree view “my_tree”. */
GX_WIDGET *selected;
status = gx_tree_view_selected_get(&my_tree, &selected);

/* If status is GX_SUCCESS the selected item of tree view “my_tree” has been retrieved. */
```

**See Also**

- `gx_menu_draw`, `gx_menu_insert`, `gx_menu_remove`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`, `gx_tree_view_create`, `gx_tree_view_draw`,
- `gx_tree_view_event_process`, `gx_tree_view_indentation_set`,
- `gx_tree_view_position`, `gx_tree_view_root_line_color_set`,
- `gx_tree_view_root_pixmap_set`, `gx_tree_view_selected_set`
**gx_tree_view_selected_set**

Set selected item

**Prototype**

```
UINT  gx_tree_view_selected_set(GX_TREE_VIEW *tree, 
                                GX_WIDGET *selected);
```

**Description**

This service sets selected item for the tree view.

**Parameters**

- **tree**
  Pointer to tree view control block
- **selected**
  Pointer to the new selected item

**Return Values**

- **GX_SUCCESS** (0x00) Successful draw menu
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

- Initialization and threads

**Example**

```
/* Set selected item of tree view “my_tree” to “tree_view_item”. */
status = gx_tree_view_selected_set(&my_tree, &tree_view_item);

/* If status is GX_SUCCESS selected item of tree view “my_menu” has
 been set to “tree_view_item”. */
```

**See Also**

- `gx_menu_draw`, `gx_menu_insert`, `gx_menu_remove`, `gx_menu_text_draw`,
- `gx_menu_text_offset_set`, `gx_tree_view_create`, `gx_tree_view_draw`,
- `gx_tree_view_event_process`, `gx_tree_view_indentation_set`,
- `gx_tree_view_position`, `gx_tree_view_root_line_color_set`,
- `gx_tree_view_root_pixelmap_set`, `gx_tree_view_selected_get`


**gx_utility_canvas_to_bmp**

Convert canvas memory to bitmap

**Prototype**

```c
UINT gx_utility_canvas_to_bmp(GX_CANVAS *canvas, GX_RECTANGLE *rect, UINT (*write_data)(GX_UBYTE *byte_data, UINT data_count));
```

**Description**

This service converts canvas memory to bitmap file.

**Parameters**

- **canvas**: Canvas control block pointer
- **rect**: Rectangle to convert
- **write_data**: Callback function pointer to write data to

**Return Values**

- **GX_SUCCESS** (0x00): Successfully converted integer value to string
- **GX_PTR_ERROR** (0x07): Invalid return buffer pointer
- **GX_INVALID_SIZE** (0x19): Invalid return buffer size

**Allowed From**

- Initialization and threads
Example

FILE *fp = GX_NULL;
/* define call back function of how to write the data read from
canvas memory. */
UINT write_data_callback(GX_UBYTE *byte_data, UINT data_count)
{
    if (fp)
    {
        fwrite(byte_data, 1, data_count, fp);
    }
    return GX_SUCCESS;
}

VOID scroll_wheel_screen_draw(GX_WINDOW *window)
{
    UINT status;
    GX_RECTANGLE size = {31, 31, 610, 450};
    gx_window_draw(window);
    if (screenshot)
    {
        fp = fopen("../screenshot.bmp", "wb");
        /* Convert canvas memory to bitmap format.
         Status GX_SUCCESS means operation succeed. */
        status = gx_utility_canvas_to_bmp(
            root->gx_window_root_canvas, &size, write_data_callback);
        fclose(fp);
    }
}

See Also

gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangle_grow,
gx_utility_rectangle_overlap_detect, gx_utility_rectangle_point_detect,
gx_utility_rectangle_shift
gx_utility_gradient_create

Create a gradient pixelmap

Prototype

INT gx_utility_gradient_create(GX_GRADIENT *gradient,
                                    GX_VALUE width, GX_VALUE height,
                                    UCHAR type, GX_UBYTE start_alpha,
                                    GX_UBYTE end_alpha);

Description

This service creates a gradient pixelmap at runtime. A gradient image can be used to accomplish fade effects and other interesting visual changes.

The width and height of the requested gradient can be no less than 2x2 pixels.

GUIX internally maintains a list of created gradients, and this function will first search the gradient list to find a matching gradient pixelmap before creating a new pixelmap. In other words, if the same gradient pixelmap is needed multiple times, only one pixelmap is actually created, and each gradient that requires this pixelmap shares the created pixelmap.

This API requires the gx_system_memory_allocator function be defined to allow runtime memory allocation.

The gradient type flags include GX_GRADIENT_TYPE_ALPHA and GX_GRADIENT_TYPE_MIRROR. Only GX_GRADIENT_TYPE_ALPHA type gradients are currently supported (i.e. this type flag must be set). The GX_GRADIENT_TYPE_MIRROR flag is optional, and when set instructs the gradient creation logic to create a gradient that changes from start_alpha to end_alpha and back to start_alpha. Otherwise a linear gradient is created.

Parameters

- **gradient**
  - Pointer to gradient control block structure
- **width**
  - Requested pixelmap width
- **height**
  - Requested pixelmap height
- **type**
  - Requested gradient type
- **start_alpha**
  - Starting alpha value
- **end_alpha**
  - End alpha value
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Gradient was created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE (0x19)</td>
<td>Gradient is not at least 2x2 pixels</td>
</tr>
<tr>
<td>GX_NOT_SUPPORTED (0x28)</td>
<td>Gradient is not type GX_GRADIENT_TYPE_ALPHA</td>
</tr>
<tr>
<td>GX_FAILURE (0x10)</td>
<td>Memory allocator is not defined or memory allocation is failed</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Gradient pointer not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE (0x22)</td>
<td>Width and height value not valid</td>
</tr>
<tr>
<td>GX_INVALID_TYPE (0x1B)</td>
<td>Gradient type not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

```c
GX_GRADIENT gradient;
UINT status;

status = gx_utility_gradient_create(&gradient, 3, 40,
                                     GX_GRADIENT_TYPE_ALPHA, 240, 0);

/* If status == GX_SUCCESS the gradient pixelmap has been created */
```

See Also

gx_utility_ltoa, gx_utility_math_asin, gx_utility_math_cos,
gx_utility_math_sin, gx_utility_math_sqrt, gx_utility_pixelmap_rotate,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift


**gx_utility_gradient_delete**

Delete a previously created gradient

**Prototype**

```c
INT gx_utility_gradient_delete(GX_GRADIENT *gradient);
```

**Description**

This service deletes a previously created gradient. If the pixelmap associated with this gradient is not in use by any other gradients, the pixelmap data will also be deleted.

**Parameters**

- **gradient**: Pointer to gradient control block

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Gradient was deleted</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Gradient pointer is not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_GRADIENT gradient;
UINT status;

/* Delete previously created gradient. */
status = gx_utility_gradient_delete(&gradient);

/* If status == GX_SUCCESS, the gradient has been deleted. */
```

**See Also**

- `gx_utility_ltoa`
- `gx_utility_math_asin`
- `gx_utility_math_cos`
- `gx_utility_math_sin`
- `gx_utility_math_sqrt`
- `gx_utility_pixelmap_rotate`
- `gx_utility_pixelmap_simple_rotate`
- `gx_utility_rectangle_center`
- `gx_utility_rectangle_center_find`
- `gx_utility_rectangle_combine`
- `gx_utility_rectangle_compare`
- `gx_utility_rectangle_define`
- `gx_utility_rectangle_grow`
- `gx_utility_rectangle_overlap_detect`
- `gx_utility_rectangle_point_detect`
- `gx_utility_rectangle_shift`
gx_utility_ltoa

Convert long integer to ASCII

Prototype

UINT  gx_utility_itoa(LONG value, GX_CHAR *return_buffer,  
                       UINT  return_buffer_size);

Description

This service converts a long integer value into an ASCII string.

Parameters

<table>
<thead>
<tr>
<th>value</th>
<th>Long integer value to convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>return_buffer</td>
<td>Destination buffer for ASCII string</td>
</tr>
<tr>
<td>return_buffer_size</td>
<td>Size of destination buffer</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>GX_SUCCESS</th>
<th>(0x00)</th>
<th>Successfully converted integer value to string</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid return buffer pointer</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19)</td>
<td>Invalid return buffer size</td>
</tr>
</tbody>
</table>

Allowed From

All
Example

INT my_value = 200;
GX_CHAR string_buffer[10];
UINT status;

/* Convert "my_value" into an ASCII string. */
status = gx_utility_ltoa(my_value, string_buffer, 10);

/* If status is GX_SUCCESS, "string_buffer" contains the ASCII representation of "my_value". */

See Also

gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt, 
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate, 
gx_utility_rectangle_center, gx_utility_rectangle_center_find, 
gx_utility_rectangle_combine, gx_utility_rectangle_compare, 
gx_utility_rectangle_define, gx_utility_rectangle_grow, 
gx_utility_rectangle_overlap_detect, gx_utility_rectangle_point_detect, 
gx_utility_rectangle_shift
**gx_utility_math_acos**

**Compute arc cosine**

**Prototype**

```
INT gx_utility_math_acos(GX_FIXED_VAL x);
```

**Description**

This service computes the angle value of the arc cosine `x`.

The input value is a fixed point data type, call `GX_FIXED_VAL_MAKE` to convert from INT to `GX_FIXED_VAL` type. For example, if you want to calculate the arc cosine of 0.5, make the input as `GX_FIXED_VAL_MAKE(1) / 2`.

In 5.4.0 or lesser version GUIX, the input value type of this function is INT, and the value is limited to the range [-256, 256]. The application must scale the value from range [-1, 1] to range [-256, 256] before invoke this service. If your project with GUIX version equal or lesser than 5.4.0 has reference to this API, and you want to upgrade your project with the latest guix library. You have two options.

1) Fix the input value of this API call to use `GX_FIXED_VAL` data type value.
2) Define `GUIX_5_4_0_COMPATIBILITY`.

**Parameters**

- `x` Value whose arc cosine is computed

**Return Values**

- `angle` Angle value of arc cosine `x`

**Allowed From**

All
Example

/* Compute the angle value of arc cosine of “0.5”. */

#if defined(GUIX_5_4_0_COMPATIBILITY)
x = 256 / 2;
#else
x = GX_FIXED_VAL_MAKE(1) / 2;
#endif

angle = gx_utility_math_acos(x);

/* “angle” contains the angle value of arc cosine “x”. */

See Also

gx_utility_ltoa, gx_utility_math_asin, gx_utility_math_cos, gx_utility_math_sin,
gx_utility_math_sqrt, gx_utility_pixelmap_rotate,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
**gx_utility_math_asin**

Compute arc sine

**Prototype**

```c
INT  gx_utility_math_asin(GX_FIXED_VAL x);
```

**Description**

This service computes the angle value of the arc sine `x`.

The input value is a fixed point data type, call `GX_FIXED_VAL_MAKE` to convert from INT to `GX_FIXED_VAL` type. For example, if you want to calculate the arc sin of 0.5, make the input as `GX_FIXED_VAL_MAKE(1) / 2`.

In 5.4.0 or lesser version GUIX, the input value type of this function is INT, and the value is limited to the range [-256, 256]. The application must scale the value from range [-1, 1] to range [-256, 256] before invoke this service. If your project with GUIX version equal or lesser than 5.4.0, and you want to upgrade your project with the latest guix library. You have two options.

1) Fix the input value of this API call to use `GX_FIXED_VAL` data type value.
2) Define `GUIX_5_4_0_COMPATIBILITY`.

**Parameters**

- **x**
  Value whose arc sine is computed

**Return Values**

- **angle**
  Angle value of arc sine `x`

**Allowed From**

- All
Example

/* Compute the angle value of arc sine of “x”. */
#if defined GUIX_5_4_0_COMPATIBILITY
x = 256 / 2;
#else
X = GX_FIXED_VAL_MAKE(1) / 2;
#endif

angle = gx_utility_math_asin(x);

/* “angle” contains the angle value of arc sine “x”. */

See Also

gx_utility_ltoa, gx_utility_math_acos, gx_utility_math_cos, gx_utility_math_sin,
gx_utility_math_sqrt, gx_utility_pixelmap_rotate,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
gx_utility_math_cos

Compute cosine

Prototype

GX_FIXED_VAL  gx_utility_math_cos(GX_FIXED_VAL angle);

Description

This service computes the cosine of the supplied angle.

The input value is a fixed point data type, call GX_FIXED_VAL_MAKE to convert from INT to GX_FIXED_VAL. For example, if you want to calculate the cosine of 90 degree, make input as GX_FIXED_VAL_MAKE(90).

The return value is a fixed point data type, call GX_FIXED_VAL_TO_INT to convert from GX_FIXED_VAL to INT.

In 5.4.0 or lesser version GUIX version, the input value and return value type of this service is INT, the input value and return value are enlarged by 256. And therefore, the application must scale the angle value by 256 before invoke this service. If your project with GUIX version equal or lesser than 5.4.0, and you want to upgrade your project with the latest guix library, you have two options.

1) Fix the input vaue and the handling to the return value of this API call to use GX_FIXED_VAL date type value.
2) Define GUIX_5_4_0_COMPATIBILITY.

Parameters

angle  Angle to compute cosine of

Return Values

cosine  Cosine of supplied angle

Allowed From

All
Example

/* Compute cosine of 90 degree. */
INT angle = 90;

#if defined (GUIX_5_4_0_COMPATIBILITY)
INT scaled_angle = angle << 8;
#else
GX_FIXED_VAL scaled_angle = GX_FIXED_VAL_MAKE(angle);
#endif

my_angle_cosine = gx_utility_math_cos(scaled_angle);

/* “my_angle_cosine” contains the cosine of “my_angle”. */

See Also

gx_utility_ltoa, gx_utility_math_acos, gx_utility_math_asin, gx_utility_math_sin,
gx_utility_math_sqrt, gx_utility_pixelmap_rotate,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
**gx_utility_math_sin**

**Compute sine**

**Prototype**

\[
GX\_FIXED\_VAL \quad gx\_utility\_math\_sin(GX\_FIXED\_VAL \ angle);
\]

**Description**

This service computes the sine of the supplied angle.

The input value is a fixed point data type, call GX\_FIXED\_VAL\_MAKE to convert from INT to GX\_FIXED\_VAL. For example, if you want to calculate the sine of 90 degree, make input as GX\_FIXED\_VAL\_MAKE(90).

The return value is a fixed point data type, call GX\_FIXED\_VAL\_TO\_INT to convert from GX\_FIXED\_VAL to INT.

In 5.4.0 or lesser version GUIX, the input value and return value type is INT, the input value and return value are enlarged by 256. And therefore, the application must scale the angle value by 256 before invoke this service. If your project with GUIX version equal or lesser than 5.4.0, and you want to upgrade your project with the latest guix library, you have two options.

3) Fix the input vaue and the handing to the return value of this API call to use GX\_FIXED\_VAL data type value.
4) Define  GUIX\_5\_4\_0\_COMPATIBILITY.

**Parameters**

- **angle**
  Angle to compute sine of

**Return Values**

- **sine**
  Sine of supplied angle

**Allowed From**

All
Example

INT my_angle = 80;
/* Compute sine of “my_angle”. */
#if defined(GUIX_5_4_0_COMPATIBILITY)
INT scaled_angle = my_angle << 8;
#else
GX_FIXED_VAL = GX_FIXED_VAL_MAKE(my_angle);
#endif

my_angle_sine = gx_utility_math_sin(scaled_angle);
/* “my_angle_sine” contains the sine of “my_angle”. */

See Also

gx_utility_ltoa, gx_utility_math_acos, gx_utility_asin, gx_utility_math_cos,
gx_utility_math_sqrt, gx_utility_pixelmap_rotate,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
gx_utility_math_sqrt

Prototype

UINT gx_utility_math_sqrt(UINT value);

Description

This service computes the square root of the supplied value.

Parameters

value Value to compute square root of

Return Values

square root Square root of supplied value

Allowed From

All

Example

/* Compute square root of "my_value". */
my_square_root = gx_utility_math_sqrt(my_value);

/* "my_square_root" contains the square root of "my_value". */

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangleGrow,
gx_utility_rectangle_overlap_detect, gx_utility_rectangle_point_detect,
gx_utility_rectangle_shift
gx_utility_pixelmap_resize

Resize pixelmap

Prototype

UINT gx_utility_pixelmap_resize(GX_PIXELMAP *src,
                                GX_PIXELMAP *destination,
                                INT width, INT height);

Description

This service resizes a pixelmap and returns a pointer to a new
pixelmap, which is the result of the pixelmap resize.

This service requires the prior use of
gx_system_memory_allocator_set, to allow allocation of memory to
hold the resized pixelmap data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Pointer to the pixelmap to resize</td>
</tr>
<tr>
<td>destination</td>
<td>Destination buffer for the resulting</td>
</tr>
<tr>
<td></td>
<td>pixelmap</td>
</tr>
<tr>
<td>width</td>
<td>Width of the resulting pixelmap, in pixels</td>
</tr>
<tr>
<td>height</td>
<td>Height of the resulting pixelmap, in pixels</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful pixelmap resize</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid source or destination pixelmap pointer</td>
</tr>
<tr>
<td></td>
<td>Width or height value not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>Source pixelmap is compressed format</td>
</tr>
<tr>
<td>GX_NOT_SUPPORTED</td>
<td>Memory allocator is not defined or memory allocation is failed</td>
</tr>
</tbody>
</table>
Allowed From

All

Example

GX_PIXLEMAP *des_pixelmap;

/* Resize "src_pixelmap" with specificy width and height. */
status = gx_utility_pixelmap_resize(src_pixelmap, &des_pixelmap, 100, 200);

/* If status is GX_SUCCESS. "des_pixelmap" successfully load the resulting pixelmap of resize. */

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift,
gx_canvas_pixelmap_rotate
gx Utility Pixelmap Rotate

Prototype

UINT gx_utility_pixelmap_rotate(GX_PIXELMAP *src, INT angle, GX_PIXELMAP *destination, UINT *rot_cx, UINT *rot_cy);

Description

This service rotates a pixelmap and returns a pointer to a new pixelmap, which is the result of the pixelmap rotation. To rotate a pixelmap directly to the canvas, use gx_canvas_pixelmap_rotate().

This service requires the prior use of gx_system_memory_allocator_set, to allow allocation of memory to hold the rotated pixelmap data.

Parameters

src The pixelmap to rotate
angle Angle of rotation in degrees
destination Destination buffer for the resulting pixelmap
rot_cx Retrieved x coordinate of rotation center with respect to destination pixelmap. Should be initiated with the x coordinate of rotation center with respect to source pixelmap. If rot_cx is GX_NULL, value will not be retrieved.
rot_cy Retrieved y coordinate of rotation center with respect to destination pixelmap. Should be initiated with the y coordinate of rotation center with respect to source pixelmap. If rot_cy is GX_NULL, value will not be retrieved.

Return Values

GX_SUCCESS (0x00) Successful pixelmap rotate
GX_PTR_ERROR (0x07) Invalid source or destination pixelmap pointer
GX_INVALID_VALUE (0x22) Angle value is 0
**GX_INVALID_FORMAT** (0x28)  
Source pixmap is compressed format, which is not supported

**GX_SYSTEM_MEMORY_ERROR** (0x30)  
Memory allocator is not defined or memory allocation is failed

**Allowed From**

All

**Example**

```c
rot_cx = source_rotate_center_x;
rot_cy = source_rotate_center_y;

/* rotate ”src_pixelmap” by 30 degree in clockwise direction. */
status = gx_utility_pixelmap_rotate(src_pixelmap, 30, &des_pixelmap,
                                   &rot_cx, &rot_cy);

/* If status is GX_SUCCESS. ”des_pixelmap” successfully load the resulting pixelmap of rotation. */
```

**See Also**

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,  
gx_utility_pixelmap_simple_rotate, gx_utility_rectangle_center,  
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,  
gx_utility_rectangle_compare, gx_utility_rectangle_define,  
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,  
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift,  
gx_canvas_pixelmap_rotate
**gx.utility.pixelmap_simple_rotate**

**Rotate pixelmap**

**Prototype**

```c
UINT gx_utility_pixelmap_simple_rotate(GX_PIXELMAP *src,
                                       INT angle,
                                       GX_PIXELMAP *destination,
                                       UINT *rot_cx,
                                       UINT *rot_cy);
```

**Description**

This service rotates a pixelmap by 90, 180 or 270 degree.

**Parameters**

- **src**: The pixelmap to rotate
- **angle**: Angle of rotation in degrees
- **destination**: Destination buffer for the resulting pixelmap
- **rot_cx**: Retrieved x coordinate of rotation center with respect to destination pixelmap. Should be initiated with the x coordinate of rotation center with respect to source pixelmap. If rot_cx is GX_NULL, value will not be retrieved.
- **rot_cy**: Retrieved y coordinate of rotation center with respect to destination pixelmap. Should be initiated with the y coordinate of rotation center with respect to source pixelmap. If rot_cy is GX_NULL, value will not be retrieved.

**Return Values**

- **GX_SUCCESS** (0x00): Successful pixelmap rotate
- **GX_PTR_ERROR** (0x07): Invalid source or destination pixelmap pointer
- **GX_INVALID_VALUE** (0x22): Angle value is 0 or not a simple angle like 90, 180, 270
- **GX_INVALID_FORMAT** (0x28): Source pixelmap is compressed format, which is not supported
**GX_SYSTEM_MEMORY_ERROR**

(0x30) Memory allocator is not defined or memory allocation is failed

**Allowed From**

All

**Example**

```c
rot_cx = source_rotate_center_x;
rot_cy = source_rotate_center_y;

/* rotate "src_pixelmap" by 90 degree in clockwise direction. */
status = gx_utility_pixelmap_simple_rotate(src_pixelmap, 90,
                                          &des_pixelmap,
                                          &rot_cx, &rot_cy);

/* If status is GX_SUCCESS. "des_pixelmap" successfully load the resulting pixelmap of rotation. */
```

**See Also**

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_rectangle_center,
gx_utility_rectangle_center_find, gx_utility_rectangle_combine,
gx_utility_rectangle_compare, gx_utility_rectangle_define,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
**gx_utility_rectangle_center**

Center rectangle within another rectangle

**Prototype**

```c
UINT gx_utility_rectangle_center(GX_RECTANGLE *rectangle,
                                 GX_RECTANGLE *within_rectangle);
```

**Description**

This service centers the rectangle within another rectangle.

**Parameters**

- `rectangle` Rectangle to center
- `within_rectangle` Rectangle to center within

**Return Values**

- **GX_SUCCESS** (0x00) Successfully centered the rectangle
- **GX_PTR_ERROR** (0x07) Invalid input rectangle pointer
- **GX_INVALID_SIZE** (0x19) Invalid rectangle size

**Allowed From**

All

**Example**

```c
UINT status;

/* Center “my_inner_rectangle” inside of “my_outer_rectangle”. */
status = gx_utility_rectangle_center(&my_inner_rectangle,
                                      &my_outer_rectangle);

/* Is status is GX_SUCCESS, “my_inner_rectangle” is centered within “my_other_rectangle”. */
```

**See Also**

- `gx_utility_ltoa`, `gx_utility_math_cos`, `gx_utility_math_sin`, `gx_utility_math_sqrt`,
- `gx_utility_pixelmap_rotate`, `gx_utility_pixelmap_simple_rotate`,
- `gx_utility_rectangle_center_find`, `gx_utility_rectangle_combine`,
- `gx_utility_rectangle_compare`, `gx_utility_rectangle_define`,
- `gx_utility_rectangle_grow`, `gx_utility_rectangle_overlap_detect`,
- `gx_utility_rectangle_point_detect`, `gx_utility_rectangle_shift`
**gx_utility_rectangle_center_find**

Find center of rectangle

Prototype

```c
UINT gx_utility_rectangle_center_find(GX_RECTANGLE *rectangle,
                                     GX_POINT *return_center);
```

Description

This service finds the center of the rectangle.

Parameters

- **rectangle**: Rectangle
- **return_center**: Pointer to center point

Return Values

- **GX_SUCCESS** (0x00): Successfully found the center of the rectangle
- **GX_PTR_ERROR** (0x07): Invalid input pointer
- **GX_INVALID_SIZE** (0x19): Invalid rectangle size

Allowed From

Initialization and threads

Example

```c
UINT status;
gx_utility_define(&my_rectangle, 0, 0, 100, 100);

/* Find center of “my_rectangle”. */
status = gx_utility_rectangle_center_find(&my_rectangle,
                                          &my_center_point);

/* If status is GX_SUCCESS, “my_center_point” is the center point of “my_rectangle” (50, 50). */
```

See Also

- `gx_utility_ltoa`, `gx_utility_math_cos`, `gx_utility_math_sin`, `gx_utility_math_sqrt`,
- `gx_utility_pixelmap_rotate`, `gx_utility_pixelmap_simple_rotate`,
- `gx_utility_rectangle_center`, `gx_utility_rectangle_combine`,
- `gx_utility_rectangle_compare`, `gx_utility_rectangle_define`,
- `gx_utility_rectangle_grow`, `gx_utility_rectangle_overlap_detect`,
- `gx_utility_rectangle_point_detect`, `gx_utility_rectangle_shift`
**gx_utility_rectangle_combine**

Combine two rectangles into first

Prototype

```c
UINT gx_utility_rectangle_combine(GX_RECTANGLE *first_rectangle,
                                   GX_RECTANGLE *second_rectangle);
```

Description

This service combines the first and second rectangle into the first rectangle. The first rectangle is expanded to include the second.

Parameters

- **first_rectangle**
  First rectangle and combined rectangle
- **second_rectangle**
  Second rectangle

Return Values

- **GX_SUCCESS**
  (0x00) Successfully combined two rectangles
- **GX_PTR_ERROR**
  (0x07) Invalid input pointer

Allowed From

Initialization and threads

Example

```c
UINT status;
GX_RECTANGLE rect_a;
GX_RECTANGLE rect_b;

gx_utility_rectangle_define(&rect_a, 0, 0, 100, 100);
gx_utility_rectangle_define(&rect_b, 50, 50, 200, 200);

/* Combine “my_rectangle_a” to “my_rectangle_b”. */
status = gx_utility_rectangle_combine(&rect_a, &rect_b);

/* If status is GX_SUCCESS, “rect_a” is (0, 0, 200, 200) the merger of the original “rect_a” and “rect_b”. */
```

See Also

- `gx_utility_ltoa`, `gx_utility_math_cos`, `gx_utility_math_sin`, `gx_utility_math_sqrt`,
- `gx_utility_pixelmap_rotate`, `gx_utility_pixelmap_simple_rotate`,
- `gx_utility_rectangle_center`, `gx_utility_rectangle_center_find`,
- `gx_utility_rectangle_compare`, `gx_utility_rectangle_define`,
- `gx_utility_rectangle_grow`, `gx_utility_rectangle_overlap_detect`,
- `gx_utility_rectangle_point_detect`, `gx_utility_rectangle_shift`
**gx_utility_rectangle_compare**

Compare two rectangles

**Prototype**

```c
GX_BOOL gx_utility_rectangle_compare(
    GX_RECTANGLE *first_rectangle,
    GX_RECTANGLE *second_rectangle);
```

**Description**

This service compares the first and second rectangle. If they are equal, a value of GX_TRUE is returned.

**Parameters**

- `first_rectangle`  
  First rectangle
- `second_rectangle`  
  Second rectangle

**Return Values**

- `result`  
  GX_TRUE if rectangles are equal, otherwise GX_FALSE is returned.

**Allowed From**

Initialization and threads

**Example**

```c
/* Compare "my_rectangle_a" to "my_rectangle_b". */
result = gx_utility_rectangle_compare(&my_rectangle_a,
                                         &my_rectangle_b);
/* If result is GX_TRUE, the two rectangles are equal. */
```

**See Also**

- `gx_utility_ltoa`
- `gx_utility_math_cos`
- `gx_utility_math_sin`
- `gx_utility_math_sqrt`
- `gx_utility_pixelmap_rotate`
- `gx_utility_pixelmap_simple_rotate`
- `gx_utility_rectangle_center`
- `gx_utility_rectangle_center_find`
- `gx_utility_rectangle_combine`
- `gx_utility_rectangle_define`
- `gx_utility_rectangle_grow`
- `gx_utility_rectangle_overlap_detect`
- `gx_utility_rectangle_point_detect`
- `gx_utility_rectangle_shift`
**gx_utility_rectangle_define**

Define a rectangle

**Prototype**

```c
UINT gx_utility_rectangle_define(GX_RECTANGLE *rectangle,
                                GX_VALUE left,
                                GX_VALUE top, GX_VALUE right,
                                GX_VALUE bottom);
```

**Description**

This service defines a rectangle as specified.

**Parameters**

- **rectangle**: Rectangle control block
- **left**: Left most coordinate
- **top**: Top most coordinate
- **right**: Right most coordinate
- **bottom**: Bottom most coordinate

**Return Values**

- **GX_SUCCESS**: (0x00) Successfully defined a rectangle
- **GX_PTR_ERROR**: (0x07) Invalid rectangle pointer

**Allowed From**

All
Example

```c
UINT status;
GX_RECTANGLE my_rect;

/* Define "my_rect". */
status = gx_utility_rectangle_define(&my_rect, 10, 5, 200, 100);

/* If status is GX_SUCCESS, "my_rect" is defined. */
```

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_grow, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
gx_utility_rectangle_overlap_detect

Detect overlap of rectangles

Prototype

GX_BOOL  gx_utility_rectangle_overlap_detect(GX_RECTANGLE *first_rectangle,
                                           GX_RECTANGLE *second_rectangle,
                                           GX_RECTANGLE *return_overlap_area);

Description

This service detects any overlap of the supplied rectangles. If overlap is found, the service returns GX_TRUE and the overlapping rectangle.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first_rectangle</td>
<td>First rectangle</td>
</tr>
<tr>
<td>second_rectangle</td>
<td>Second rectangle</td>
</tr>
<tr>
<td>return_overlap_area</td>
<td>Overlapping rectangle area</td>
</tr>
</tbody>
</table>

Return Values

result GX_TRUE if rectangles overlap, otherwise GX_FALSE.

Allowed From

All

Example

/* Detect overlap of "my_rectangle_a" and "my_rectangle_b". */
result = gx_utility_rectangle_overlap_detect(&my_rectangle_a,
                                             &my_rectangle_b,
                                             &my_overlap_area);

/* If result is GX_TRUE, "my_overlap_area" specifies the area the rectangles overlap. */

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangle_grow,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
gx_utility_rectangle_point_detect

Detect if point resides in rectangle

Prototype

GX_BOOL gx_utility_rectangle_point_detect(GX_RECTANGLE *rectangle,
                                          GX_POINT point);

Description

This service detects if the specified point resides in the rectangle. If
the point does reside in the rectangle, the service returns
GX_TRUE.

Parameters

rectangle  Rectangle
point      Point

Return Values

result     GX_TRUE if point resides in rectangle,
            otherwise GX_FALSE

Allowed From

All
Example

GX_RECTANGLE my_rectangle;
GX_POINT my_point;

gx_utility_rectangle_define(&my_rectangle, 0, 0, 100, 100);

my_point.gx_point_x = 20;
my_point.gx_point_y = 20;

/* Detect if point “my_point” is within “my_rectangle”. */
result = gx_utility_rectangle_point_detect(&my_rectangle,
                                          &my_point);

/* If result is GX_TRUE, “my_point” resides in the rectangle. */

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangle_grow,
gx_utility_rectangle_overlap_detect, gx_utility_rectangle_shift
gx_utility_rectangle_resize

Grow rectangle

Prototype

UINT gx_utility_rectangle_resize(GX_RECTANGLE *rectangle,
                                 GX_VALUE adjust);

Description

This service increases the size of the rectangle as specified.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectangle</td>
<td>Pointer to rectangle</td>
</tr>
<tr>
<td>adjust</td>
<td>Amount to adjust the rectangle</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successfully resized the rectangle</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid input rectangle pointer</td>
</tr>
</tbody>
</table>

Allowed From

All

Example

UINT status;

/* Adjust “my_rectangle” by increasing 20 pixels on four sides */
status = gx_utility_rectangle_resize(&my_rectangle, 20);

/* If status is GX_SUCCESS, “my_rectangle” is 20 pixels larger. */

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangle_overlap_detect,
gx_utility_rectangle_point_detect, gx_utility_rectangle_shift
The function `gx_utility_rectangle_shift` shifts a rectangle by the specified values.

**Prototype**

```c
UINT gx_utility_rectangle_shift(GX_RECTANGLE *rectangle,
    GX_VALUE x_shift,
    GX_VALUE y_shift);
```

**Description**

This service shifts the rectangle by the specified values.

**Parameters**

- `rectangle` : Rectangle to shift
- `x_shift` : Number of pixels to shift on the x-axis
- `y_shift` : Number of pixels to shift on the y-axis

**Return Values**

- **GX_SUCCESS** (0x00) : Successfully shifted the rectangle
- **GX_PTR_ERROR** (0x07) : Invalid input rectangle pointer

**Allowed From**

All

**Example**

```c
UINT status;

/* Shift "my_rectangle". */
status = gx_utility_rectangle_shift(&my_rectangle, 10, 20);

/* If status is GX_SUCCESS, "my_rectangle" has been shifted. */
```

**See Also**

- `gx_utility_ltoa`, `gx_utility_math_cos`, `gx_utility_math_sin`, `gx_utility_math_sqrt`, `gx_utility_pixelmap_rotate`, `gx_utility_pixelmap_simple_rotate`,
- `gx_utility_rectangle_center`, `gx_utility_rectangle_center_find`,
- `gx_utility_rectangle_combine`, `gx_utility_rectangle_compare`,
- `gx_utility_rectangle_define`, `gx_utility_rectangle_grow`,
- `gx_utility_rectangle_overlap_detect`, `gx_utility_rectangle_point_detect`
**gx_utility_string_to_alphamap**

Render string to an 8bpp alphamap type pixelmap (deprecated)

**Prototype**

```c
UINT gx_utility_string_to_alphamap(const GX_CHAR *text,
const GX_FONT *font, GX_PIXELMAP *return_map);
```

**Description**

This service has been deprecated in favor of gx_utility_string_to_alphamap_ext().

This service renders a text string to an alphamap, which is a special form of 8bpp pixelmap containing only alpha values. This service is typically used along with gx_utility_pixelmap_rotate and gx_canvas_pixelmap_draw to draw rotated text to the canvas.

This service calculates the memory size needed for the resulting alphamap, and invokes the gx_system_memory_allocator() function defined by the application to dynamically allocate memory. The application must call gx_system_memory_allocator_set() at some point, usually during program startup, prior to using this service.

If a text string is to be rotated and drawn to the canvas just once, the service gx_canvas_rotated_text_draw() is provided as an alternate. gx_canvas_rotated_text_draw() will call gx_utility_string_to_alphamap(), gx_utility_pixelmap_rotate(), and gx_canvas_pixelmap_draw() to render the rotated text in one operation. However if the same text will be drawn multiple times rotated at various angles, it is more efficient to create the alphamap once using the gx_utility_string_to_alphamap API, then rotate the resulting alphamap multiple times as needed.

**Parameters**

- **text**
  - Text string to render to alphamap
- **font**
  - The font to be to render the text
- **return_map**
  - Pointer to the GX_PIXELMAP to be returned to the caller.

**Return Values**

- **GX_SUCCESS** (0x00) Successfully rendered a text string to an alphamap
- **GX_PTR_ERROR** (0x07) Invalid input pointer
- **GX_SYSTEM_MEMORY_ERROR**
(0x30) Memory allocation/free function is not defined

GX_INVALID_STRING_LENGTH (0x34) Invalid string length

Allowed From

Initialization and threads

Example

GX_PIXELMAP alphamap;
GX_PIXELMAP rotated_text;
INT xpos;
INT ypos;

gx_widget_font_get(widget, GX_FONT_ID_SCREEN_LABEL, &font);

/* render string to alphamap once */
gx_utility_string_to_alphamap("Hello World", font, &alphamap);

/* rotate and render the alphmap at multiple angles */
gx_utility_pixelmap_rotate(&alphamap, 45, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(10, 10, &rotated_text);

gx_utility_pixelmap_rotate(&alphamap, 135, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(100, 100, &rotated_text);

gx_utility_pixelmap_rotate(&alphamap, 300, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(200, 200, &rotated_text);

See Also

gx_utility_string_to_alphamap_ext


**gx_utility_string_to_alphamap_ext**

Render string to an 8bpp alphamap type pixelmap

Prototype

```c
UINT gx_utility_string_to_alphamap_ext(
    GX_CONST GX_STRING *string,
    GX_CONST GX_FONT *font, GX_PIXELMAP *return_map);
```

Description

This service renders a text string to an alphamap, which is a special form of 8bpp pixelmap containing only alpha values. This service is typically used along with `gx_utility_pixelmap_rotate` and `gx_canvas_pixelmap_draw` to draw rotated text to the canvas.

This service calculates the memory size needed for the resulting alphamap, and invokes the `gx_system_memory_allocator()` function defined by the application to dynamically allocate memory. The application must call `gx_system_memory_allocator_set()` at some point, usually during program startup, prior to using this service.

If a text string is to be rotated and drawn to the canvas just once, the service `gx_canvas_rotated_text_draw()` is provided as an alternate. `gx_canvas_rotated_text_draw()` will call `gx_utility_string_to_alphamap()`, `gx_utility_pixelmap_rotate()`, and `gx_canvas_pixelmap_draw()` to render the rotated text in one operation. However if the same text will be drawn multiple times rotated at various angles, it is more efficient to create the alphamap once using the `gx_utility_string_to_alphamap` API, then rotate the resulting alphamap multiple times as needed.

Parameters

- **string**: Text string to render to alphamap
- **font**: The font to be to render the text
- **return_map**: Pointer to the GX_PIXELMAP to be returned to the caller.

Return Values

- **GX_SUCCESS** (0x00): Successfully rendered a text string to an alphamap
- **GX_PTR_ERROR** (0x07): Invalid input pointer
- **GX_SYSTEM_MEMORY_ERROR**
(0x30) Memory allocation/free function is not defined

GX_INVALID_STRING_LENGTH (0x34) Invalid string length

Allowed From

Initialization and threads

Example

GX_STRING string;
GX_PIXELMAP alphamap;
GX_PIXELMAP rotated_text;
INT xpos;
INT ypos;

gx_widget_font_get(widget, GX_FONT_ID_SCREEN_LABEL, &font);

string.gx_string_ptr = "Hello World";
string.gx_string_length = strlen(string.gx_string_ptr);

/* render string to alphamap once */
gx_utility_string_to_alphamap_ext(&string, font, &alphamap);

/* rotate and render the alphmap at multiple angles */
gx_utility_pixelmap_rotate(&alphamap, 45, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(10, 10, &rotated_text);

gx_utility_pixelmap_rotate(&alphamap, 135, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(100, 100, &rotated_text);

gx_utility_pixelmap_rotate(&alphamap, 300, &rotated_text, &xpos, &ypos);
gx_canvas_pixelmap_draw(200, 200, &rotated_text);

See Also

gx_utility_ltoa, gx_utility_math_cos, gx_utility_math_sin, gx_utility_math_sqrt,
gx_utility_pixelmap_rotate, gx_utility_pixelmap_simple_rotate,
gx_utility_rectangle_center, gx_utility_rectangle_center_find,
gx_utility_rectangle_combine, gx_utility_rectangle_compare,
gx_utility_rectangle_define, gx_utility_rectangle_grow,
gx_utility_rectangle_overlap_detect, gx_utility_rectangle_point_detect
**gx_vertical_list_children_position**

Position children for the vertical list

**Prototype**

```c
UINT gx_vertical_list_children_position(
    GX_VERTICAL_LIST *vertical_list)
```

**Description**

This function positions the children for the vertical list.

**Parameters**

- `vertical_list` Pointer to the vertical list control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully positioned the children for the vertical list
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

- Initialization and threads

**Example**

```c
/* Position children in the vertical list */
status = gx_vertical_list_children_position (&vertical_list);

/* If status is GX_SUCCESS the children in the vertical list are positioned.. */
```

**See Also**

- `gx_vertical_list_create`, `gx_vertical_list_event_process`,
- `gx_vertical_list_page_index_set`, `gx_vertical_list_selected_index_get`,
- `gx_vertical_list_selected_widget_get`, `gx_vertical_list_selected_widget_get`,
- `gx_vertical_list_selected_set`, `gx_vertical_list_total_rows_set`
gx_vertical_list_create

Create vertical list

Prototype

UINT gx_vertical_list_create(GX_VERTICAL_LIST *vertical_list, 
   GX_CONST GX_CHAR *name, GX_WIDGET *parent, INT total_rows, 
   VOID (*callback)(GX_VERTICAL_LIST *, GX_WIDGET *, INT), 
   ULONG style, USHORT vertical_list_id, 
   GX_CONST GX_RECTANGLE *size);

Description

This service creates a vertical list.

GX_VERTICAL_LIST is derived from GX_WINDOW and supports all gx_window API services.

Parameters

vertical_list: Vertical list widget control block
name: Name of vertical list
parent: Pointer to parent widget
total_rows: Total number of rows in vertical list
callback: A function that will be called by the vertical list when the list is scrolled. The caller should initially create enough GX_WIDGET based children to fill the visible list rows. As the list is scrolled, this function is called to re-create the list children corresponding to the supplied list index
style: Style of scrollbar widget. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
vertical_list_id: Application-defined ID of vertical list
size: Dimensions of vertical list
Return Values

- **GX_SUCCESS** (0x00) Successfully created the vertical list
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size
- **GX_INVALID_VALUE** (0x22) Number of rows not valid
- **GX_INVALID_WIDGET** (0x12) Widget not valid

Allowed From

Initialization and threads

Example

```c
/* Create vertical list "my_list" with 20 rows. */
status = gx_vertical_list_create(&my_list, "my_list", &my_parent,
    20, callback, GX_STYLE_WRAP, MY_LIST_ID,
    &size);

/* If status is GX_SUCCESS the vertical list "my_list" has been created. */
```

See Also

- `gx_vertical_list_children_position`, `gx_vertical_list_event_process`,
- `gx_vertical_list_page_index_set`, `gx_vertical_list_selected_index_get`,
- `gx_vertical_list_selected_widget_get`, `gx_vertical_list_selected_set`,
- `gx_vertical_list_total_rows_set`
**gx_vertical_list_event_process**

Process vertical list event

Prototype

```c
UINT gx_vertical_list_event_process(GX_VERTICAL_LIST *list,
                                   GX_EVENT *event);
```

Description

This service processes an event for the vertical list.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>Vertical list widget control block</td>
</tr>
<tr>
<td>event</td>
<td>Pointer to event to process</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully processed the vertical list event</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Threads

Example

```c
/* Process "my event" for vertical list "my_list". */
status = gx_vertical_list_event_process(&my_list, &my_event);

/* If status is GX_SUCCESS the event for vertical list "my_list"
has been processed. */
```

See Also

- `gx_vertical_list_children_position`
- `gx_vertical_list_create`
- `gx_vertical_list_page_index_set`
- `gx_vertical_list_selected_index_get`
- `gx_vertical_list_selected_widget_get`
- `gx_vertical_list_selected_set`
**gx_vertical_list_page_index_set**

Set starting page index

**Prototype**

```c
UINT gx_vertical_list_page_index_set(GX_VERTICAL_LIST *list, INT index);
```

**Description**

This service sets the starting index for the vertical list.

**Parameters**

- `list` Vertical list widget control block
- `index` The new top index

**Return Values**

- **GX_SUCCESS** (0x00) Successfully set starting page index for the vertical list
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid widget pointer
- **GX_INVALID_VALUE** (0x22) Invalid index value
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Set the starting page index of vertical list "my_list" to 4. */
status = gx_vertical_list_page_index_set(&my_list, 4);

/* If status is GX_SUCCESS the starting page index of "my_list" has been set to 4. */
```

**See Also**

- `gx_vertical_list_children_position`
- `gx_vertical_list_create`
- `gx_vertical_list_event_process`
- `gx_vertical_list_selected_index_get`
- `gx_vertical_list_selected_widget_get`
- `gx_vertical_list_selected_set`
- `gx_vertical_list_total_rows_set`
**gx_vertical_list_selected_index_get**

Get selected index from vertical list

**Prototype**

```c
UINT gx_vertical_list_selected_index_get(
    GX_VERTICAL_LIST *vertical_list,
    INT *return_index);
```

**Description**

This service returns the selected index of the vertical list

**Parameters**

- `vertical_list`: Vertical list widget control block
- `return_index`: Destination for return of selected index

**Return Values**

- **GX_SUCCESS** (0x00): Successfully get the vertical list entry
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
INT current_selected_index;

/* Get the list entry at the current index of vertical list "my_list". */
status = gx_vertical_list_selected_index_get(&my_list, &current_selected_index);

/* If status is GX_SUCCESS, “current_list_index” contains the index of the selected list item. */
```

**See Also**

- `gx_vertical_list_children_position`
- `gx_vertical_list_create`
- `gx_vertical_list_event_process`
- `gx_vertical_list_page_index_set`
- `gx_vertical_list_selected_widget_get`
- `gx_vertical_list_selected_set`
- `gx_vertical_list_total_rows_set`
**gx_vertical_list_selected_set**

Assign the selected entry in a vertical list

Prototype

```c
UINT gx_vertical_list_selected_set(
    GX_VERTICAL_LIST *vertical_list,
    INT index);
```

Description

This service assigns the selected entry in a vertical list. If necessary the vertical list will scroll to make the selected entry visible.

Parameters

- `vertical_list`: Vertical list widget control block
- `index`: Index based position of new list entry

Return Values

- **GX_SUCCESS** (0x00): Successfully set the vertical list entry
- **GX_FAILURE** (0x10): Input index not found in list
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Vertical list or list entry widget not valid

Allowed From

Initialization and threads

Example

```c
/* Set the list entry of “my_list” to the child in line 12. */
status = gx_vertical_list_selected_set(&my_list, 12);

/* If status is GX_SUCCESS, the list entry of “my_list” has been successfully set to 12. */
```

See Also

- `gx_vertical_list_children_position`, `gx_vertical_list_create`
- `gx_vertical_list_event_process`, `gx_vertical_list_page_index_get`
- `gx_vertical_list_selected_index_get`, `gx_vertical_list_selected_widget_get`
- `gx_vertical_list_total_rows_set`
**gx_vertical_list_selected_widget_get**

Get selected widget from vertical list

**Prototype**

```c
UINT gx_vertical_list_selected_widget_get(
    GX_VERTICAL_LIST *vertical_list,
    GX_WIDGET **return_list_entry);
```

**Description**

This service returns the selected widget of the vertical list. Note that if the list contains more rows than child widgets, and the selected child widget has been scrolled from view, this function will return GX_NULL as the GX_WIDGET pointer, since the widget has been re-used to display a new list entry.

**Parameters**

- `vertical_list` Vertical list widget control block
- `return_list_entry` Destination for return list entry widget

**Return Values**

- **GX_SUCCESS** (0x00) Successfully get the vertical list entry
- **GX_FAILURE** (0x10) The selected widget has been scrolled from view.
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads
Example

GX_WIDGET *current_selected_widget;

/* Get the list entry at the current index of vertical list
"my_list". */
status = gx_vertical_list_selected_widget_get(&my_list,
&current_selected_widget);

/* If status is GX_SUCCESS, “current_list_entry” contains a pointer
to the currently selected widget. */

See Also

gx_vertical_list_children_position, gx_vertical_list_create,
gx_vertical_list_event_process, gx_vertical_list_page_index_set,
gx_vertical_list_selected_index_get, gx_vertical_list_selected_set,
gx_vertical_list_total_rows_set
gx_vertical_list_total_rows_set

Set total number of vertical list rows

Prototype

UINT gx_vertical_list_total_rows_set(
    GX_VERTICAL_LIST *vertical_list,
    INT count);

Description

This service assigns or changes the total number of list rows.

Parameters

| vertical_list | Vertical list widget control block |
| count | New list row count |

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>0x00</td>
<td>Successfully set the vertical list row count</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>0x11</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>0x07</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>0x12</td>
<td>Widget not valid</td>
</tr>
<tr>
<td>GX_INVALID_VALUE</td>
<td>0x22</td>
<td>Row count value not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Set the list row count to 20 items. */
status = gx_vertical_list_total_rows_set(&my_list, 20);

/* If status is GX_SUCCESS, the total rows of “my_list” has been set to 20. */

See Also

gx_vertical_list_children_position, gx_vertical_list_create,
gx_vertical_list_event_process, gx_vertical_list_page_index_set,
gx_vertical_list_selected_index_get, gx_vertical_list_selected_widget_get,
gx_vertical_list_selected_set
**gx_vertical_scrollbar_create**

Create vertical scrollbar

**Prototype**

```c
UINT gx_verticalScrollbar_create(GX_SCROLLBAR *scrollbar,
GX_CONST GX_CHAR *name,
GX_WINDOW *parent,
GX_SCROLLBAR_APPEARANCE *appearance,
ULONG style);
```

**Description**

This service creates a vertical scrollbar.

**Parameters**

- **scrollbar**
  - Scrollbar widget control block
- **name**
  - Name of scrollbar
- **parent**
  - Pointer to parent widget
- **appearance**
  - Appearance of vertical scrollbar widget.
- **style**
  - Style of the scrollbar.

**Return Values**

- **GX_SUCCESS** (0x00)
  - Successful vertical scrollbar create
- **GX_CALLER_ERROR** (0x11)
  - Invalid caller of this function
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer
- **GX_ALREADY_CREATED** (0x13)
  - Widget already created
- **GX_INVALID_SIZE** (0x19)
  - Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12)
  - Parent widget not valid

**Allowed From**

Initialization and threads
Example

/* Create vertical scrollbar ”my_scrollbar”. */
status = gx_vertical_scrollbar_create(&my_scrollbar,
    "my_vertical_scrollbar",
    &my_parent, &scrollbar_appearance,
    GX_STYLE_ENABLED);

/* If status is GX_SUCCESS the vertical scrollbar ”my_scrollbar”
has been created. */

See Also

gx_horizontal_scrollbar_create, gx_scrollbar_draw, gx_scrollbar_event_process,
gx_scrollbar_limit_check, gx_scrollbar_reset
**gx_widget_allocate**

Allocate a widget control block

**Prototype**

```c
UINT gx_widget_allocate(GX_WIDGET **control_block, ULONG memsize);
```

**Description**

This service dynamically allocates a widget control block, by calling the application defined memory allocation function. This service is primarily used by the functions generated by GUIX Studio to dynamically allocate control block when the “Dynamic Allocation” property is selected in the GUIX Studio properties view.

**Parameters**

- **control_block**: Pointer to returned control block pointer
- **memsize**: Control block size, in bytes

**Return Values**

- **GX_SUCCESS** (0x00): Successful widget allocate
- **GX_SYSTEM_MEMORY_ERROR** (0x30): Memory allocator is not defined or memory allocation failed
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_MEMORY_SIZE** (0x29): Memory size not valid

**Allowed From**

- Initialization and threads
Example

GX_TEXT_BUTTON *button;

/* Attach “my_widget” to “my_parent”. */
status = gx_widget_allocate(&button, sizeof(GX_TEXT_BUTTON));

/* If status is GX_SUCCESS the button widget control block is
allocated. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_back_style_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_child_show, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_attach

Attach widget to its parent

Prototype

UINT  gx_widgetAttach(GX_WIDGET *parent, GX_WIDGET *widget);

Description

This service attaches the widget to the specified parent. If the widget is already attached to another parent, it is first detached. If the widget is already attached to the same parent, the function does nothing.

The widget becomes the front-most child of its parent in terms of z-ordering. If sibling widgets overlap, this widget is drawn on top of siblings. To put the new widget in the back of the z-order, use gx_widget_backAttach or gx_widget_back_move.

Parameters

parent  Pointer to parent widget
widget  Pointer to child widget

Return Values

GX_SUCCESS  (0x00)  Successful widget attach
GX_CALLER_ERROR  (0x11)  Invalid caller of this function
GX_PTR_ERROR  (0x07)  Invalid pointer
GX_INVALID_WIDGET  (0x12)  Parent or widget not valid

Allowed From

Initialization and threads
Example

/* Attach "my_widget" to "my_parent". */
status = gx_widget_attach(&my_parent, &my_widget);

/* If status is GX_SUCCESS the widget "my_widget" is attached to "my_parent". */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_background_draw**

Draw a widget background

**Prototype**

```c
VOID gx_widget_background_draw(GX_WIDGET *widget);
```

**Description**

This service performs a solid color fill of a widget background. This service is automatically called by the `gx_widget_draw` function, but may also be invoked by the application as part of a customized widget drawing function.

**Parameters**

- **widget**: Pointer to widget to be drawn

**Return Values**

- None

**Allowed From**

- Threads

**Example**

```c
/* Write a custom widget draw function. */

VOID my_widget_draw(GX_WIDGET * widget)
{
    /* Call default widget background draw. */
    gx_widget_background_draw(widget);

    /* Add your own drawing here. */

    /* Draw child widgets. */
    gx_widget_children_draw(widget);
}
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_back_attach**

Attach widget to its parent

**Prototype**

```c
UINT gx_widget_back_attach(GX_WIDGET *parent, GX_WIDGET *widget);
```

**Description**

This service attaches the widget to the specified parent. If the widget is already attached to another parent, it is first detached. If the widget is already attached to the same parent, the function does nothing.

The widget becomes the back-most child of its parent in terms of z-ordering. If sibling widgets overlap, this widget is drawn behind those siblings. To put the new widget in the front of the z-order, use `gx_widget_attach` or `gx_widget_front_move`.

**Parameters**

- **parent** Pointer to parent widget
- **widget** Pointer to child widget

**Return Values**

- **GX_SUCCESS** (0x00) Successful widget attach
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Parent or widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Attach “my_widget” to “my_parent”. */
status = gx_widget_back_attach(&my_parent, &my_widget);

/* If status is GX_SUCCESS the widget “my_widget” is attached to “my_parent”. */
```
See Also

gx_widget_back_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_back_move**

**Move widget to back**

**Prototype**

```c
UINT  gx_widget_back_move(GX_WIDGET *widget,
                            GX_BOOL *return_widget_moved);
```

**Description**

This service moves the widget to the back in the parent's Z-order of child widgets.

**Parameters**

- **parent**
  - Pointer to parent widget
- **return_widget_moved**
  - Pointer to destination for flag indicating the widget was moved

**Return Values**

- **GX_SUCCESS** (0x00)
  - Successful widget move to the back
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer
- **GX_CALLER_ERROR** (0x11)
  - Invalid caller of this function
- **GX_NO_CHANGE** (0x08)
  - No changes are applied
- **GX_INVALID_WIDGET** (0x12)
  - Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Move “my_widget” to the back. */
status = gx_widget_back_move(&my_widget, &moved_flag);

/* If status is GX_SUCCESS and “moved_flag” is GX_TRUE, the widget “my_widget” was moved to the back. */
```
See Also

gx_widget_attach, gx_widget_background_set, gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_block_move

Move a rectangular block of pixels

Prototype

UINT gx_widget_block_move(GX_WIDGET *widget,
                           GX_RECTANGLE *block,
                           INT xshift, INT yshift);

Description

This service moves a rectangular block of pixels. This service is
most often used to implement fast scrolling.

Parameters

- **widget**: Pointer to widget requesting block move
- **block**: Rectangle bounding block to move
- **xshift**: The x shift amount in pixels
- **yshift**: The y shift amount in pixels

Return Values

- **GX_SUCCESS** (0x00): Successful widget move to
  the back
- **GX_INVALID_CANVAS** (0x20): Widget canvas not found
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

- Initialization and threads

Example

```c
/* Move a block of pixels 20 pixels to the right. */
status = gx_widget_block_move(&my_widget, &size, 20, 0);

/* If status is GX_SUCCESS the block of pixels was moved. */
```
See Also

gx_widget_attach, gx_widget_background_set, gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_border_draw**

Draw widget border

**Prototype**

```c
VOID gx_widget_border_draw(GX_WIDGET *widget,
                           GX_COLOR border_color,
                           GX_COLOR upper_fill, GX_COLOR lower_fill,
                           GX_BOOL fill);
```

**Description**

This service draws the widget border. This service is normally invoked as part of a widget drawing function. This service interprets the widget border style flags to draw no border, a thin border, a raised border, a recessed border, or a thick border.

**Parameters**

- **widget**  
  Pointer to widget

- **border_color**  
  Color of border. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.

- **upper_fill**  
  Color of upper fill. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.

- **lower_fill**  
  Color of lower fill. Appendix A contains pre-defined colors. Note that the application may add custom colors as well.

- **fill**  
  This boolean flag indicates whether or not the widget area should be filled with the supplied fill colors. If this value is GX_FALSE, only the widget border is drawn.

**Return Values**

None

**Allowed From**

Threads
Example

/* Write a custom widget draw function. */

VOID my_widget_draw(GX_WIDGET * widget)
{
    /* Call widget border draw. */
    gx_widget_border_draw(widget, GX_COLOR_BLACK,
                          GX_COLOR_GREEN, GX_COLOR_BLUE,
                          GX_TRUE);

    /* Add your own drawing here. */

    /* Draw child widgets. */
    Gx_widget_children_draw(widget);
}

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_border_style_set

Set widget border style

Prototype

UINT gx_widget_border_style_set(GX_WIDGET *widget, ULONG style);

Description

This service sets the widget border style.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>style</td>
<td>Style of border. <strong>Appendix D</strong> contains pre-defined general styles for all widgets as well as widget-specific styles.</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful widget border style set
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

Allowed From

Initialization and threads
Example

/* Set border style of "my_widget". */
status = gx_widget_border_style_set(&my_widget,
        GX_STYLE_BORDER_RAISED);

/* If status is GX_SUCCESS the widget "my_widget" border style has
been set. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_backgr
ground_set, gx_widget_child_detect, gx_widget_children
draw, gx_widget_client_get, gx_widget_create, gx_widget_c
reated_test, gx_widget_delete, gx_widget_detach, gx_wi
dget_draw, gx_widget_draw_set, gx_widget_event_generate,
gx_widget_event_process, gx_widget_event_process_set,
gx_widget_event_to_parent, gx_widget_find, gx_widget_fron
move, gx_widget_height_get, gx_widget_hide, gx_widget_res
ze, gx_widget_shift, gx_widget_show, gx_widget_status_add,
gx_widget_status_get, gx_widget_status_remove, gx_widget
status_test, gx_widget_style_add, gx_widget_style_get,
gx_widget_style_remove, gx_widget_style_set, gx_widget
width_get
**gx_widget_border_width_get**

Get widget border width

**Prototype**

```c
UINT gx_widget_border_width_get(GX_WIDGET *widget,
                                 GX_VALUE *return_width);
```

**Description**

This service gets the widget border width.

**Parameters**

- **widget**: Pointer to widget
- **return_width**: Pointer to destination for widget border width

**Return Values**

- **GX_SUCCESS** (0x00): Successfully retrieved border width
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads
Example

GX_VALUE my_width;

/* Get border width of “my_widget”. */
status = gx_widget_border_width_get(&my_widget, &my_width);

/* If status is GX_SUCCESS, “my_width” contains the border width of
the widget “my_widget”. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set, gx_widget_canvas_get,
gx_widget_child_detect, gx_widget_children_draw, gx_widget_client_get,
gx_widget_create, gx_widget_created_test, gx_widget_delete, gx_widget_detach,
gx_widget_draw, gx_widget_draw_set, gx_widget_event_generate,
gx_widget_event_process, gx_widget_event_process_set,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_canvas_get**

Get widget canvas

Prototype

```c
UINT gx_widget_canvas_get(GX_WIDGET *widget,
                           GX_CANVAS **return_canvas)
```

Description

This service returns a pointer to the canvas onto which this widget is rendered.

Parameters

- `widget`: Pointer to widget
- `return_canvas`: Pointer to destination for widget’s canvas

Return Values

- **GX_SUCCESS** (0x00): Successful widget canvas get
- **GX_FAILURE** (0x10): Widget canvas not found
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads

Example

```c
/* Get canvas associated with ”my_widget”. */
status = gx_widget_canvas_get(&my_widget, &my_canvas);

/* If status is GX_SUCCESS, ”my_canvas” contains the canvas of the widget ”my_widget”. */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_child_detect

Detect widget child

Prototype

UINT gx_widget_child_detect(GX_WIDGET *parent, GX_WIDGET *child, 
GX_BOOL *return_detect);

Description

This service detects if the widget is a child of the parent widget. This
service nests to search children of children, and returns TRUE if the
parent widget is at any level an ancestor of the child widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>child</td>
<td>Pointer to child widget</td>
</tr>
<tr>
<td>return_detect</td>
<td>Pointer to destination for detection</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful widget child detection
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Parent or child widget not valid

Allowed From

Initialization and threads
Example

GX_BOOL detected;

/* Determine if "my_child" is a child of "my_widget". */
status = gx_widget_child_detect(&my_widget, &my_child, &detected);

/* If status is GX_SUCCESS and "detected" is GX_TRUE, "my_child" is
a child of widget "my_widget". */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_children_draw

Prototype

VOID gx_widget_children_draw(GX_WIDGET *widget);

Description

This service draws all children of the parent widget. This service is normally invoked by all standard widget drawing functions to draw any existing child widgets, and should be invoked by any custom drawing functions to allow child widgets to be attached to your custom parent widget type.

Parameters

widget Pointer to widget

Return Values

None

Allowed From

Threads

Example

/* Write a custom widget draw function. */

VOID my_widget_draw(GX_WIDGET * widget)
{  
   /* Call default widget background draw. */
   gx_widget_background_draw(widget);

   /* Add your own drawing here. */

   /* Draw child widgets. */
   gx_widget_children_draw(widget);
}
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_client_get**

Get widget client area

**Prototype**

```c
UINT gx_widget_client_get(GX_WIDGET *widget,
                           GX_VALUE border_width,
                           GX_RECTANGLE *return_client_area);
```

**Description**

This service computes the client area of widget by subtracting the widget border width from the overall widget size.

**Parameters**

- **widget**
  Pointer to widget
- **border_width**
  Width of widget border
- **return_client_area**
  Destination for returning client area

**Return Values**

- **GX_SUCCESS** (0x00)  
  Successful widget client area get
- **GX_PTR_ERROR** (0x07)  
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)  
  Widget not valid
- **GX_INVALID_VALUE** (0x22)  
  Widget border not valid

**Allowed From**

Initialization and threads
**Example**

```c
GX_RECTANGLE client_area
/* Get client area of widget "my_widget". */
status = gx_widget_client_get(&my_widget, my_widget_width,
    &client_area);

/* If status is GX_SUCCESS, the "client_area" is the client area of
"my_widget". */
```

**See Also**

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_color_get

Get color

Prototype

UINT gx_widget_color_get(GX_WIDGET *widget,
                        GX_RESOURCE_ID resource_id,
                        GX_COLOR *return_color);

Description

This service gets the color associated with the supplied resource ID.
This service should only be called by visible widgets.

Parameters

- **widget**: Pointer to widget control block
- **resource_id**: Resource ID of color. [Appendix B](#) contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **return_color**: Pointer to destination for color. [Appendix A](#) contains pre-defined colors. Note that the application may add custom colors as well.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful color get</td>
</tr>
<tr>
<td>GX_INVALID_RESOURCE_ID</td>
<td>(0x33) Invalid resource ID</td>
</tr>
<tr>
<td>GX_INVALID_CANVAS</td>
<td>(0x20) Widget canvas not valid or widget is invisible</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_COLOR actual_color;

/* Get color for resource ID MY_FIRST_COLOR_ID. */
status = gx_widget_color_get(my_widget, MY_FIRST_COLOR_RESOURCE_ID,
                             &actual_color);

/* If status is GX_SUCCESS the actual color is contained in 
   "actual_color". */

See Also

gx_widget_font_get, gx_widget_pixelmap_get
**gx_widget_create**

Create widget

Prototype

```c
UINT gx_widget_create(GX_WIDGET *widget, GX_CONST GX_CHAR *name,
GX_WIDGET *parent, ULONG style, USHORT widget_id,
GX_CONST GX_RECTANGLE *size);
```

Description

This service creates a widget.

Parameters

- **widget** Pointer to widget
- **name** Logical name of widget
- **parent** Pointer to parent widget
- **style** Style. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.
- **widget_id** Application-defined ID of the widget
- **size** Size of the widget

Return Values

- **GX_SUCCESS** (0x00) Successful widget create
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_ALREADY_CREATED** (0x13) Widget already created
- **GX_INVALID_SIZE** (0x19) Invalid widget control block size
- **GX_INVALID_WIDGET** (0x12) Parent widget not valid

Allowed From

Initialization and threads
Example

```c
GX_WIDGET my_widget;
GX_RECTANGLE size;

gx_utility_rectangle_define(&size, 0, 0, 100, 100);

/* Get client area of widget "my_widget". */
status = gx_widget_create(&my_widget, "my widget",
                           &my_parent_window,
                           GX_STYLE_BORDER_RAISED, MY_WIDGET_ID,
                           &size);

/* If status is GX_SUCCESS, the widget "my_widget" has been
   created. */
```

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_created_test

Test if widget created

Prototype

UINT gx_widget_created_test(GX_WIDGET *widget, 
                           GX_BOOL *return_test);

Description

This service tests to determine if the widget has previously been created. If no errors are encountered, this function return GX_SUCCESS, regardless if the widget is created yet or not. The result of the test is in the return_test pointer.

Parameters

widget Pointer to widget
return_test Destination for test result

Return Values

GX_SUCCESS (0x00) Successful test completion
GX_PTR_ERROR (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

GX_BOOL was_created;

/* Test to see if widget “my_widget” is created. */
status = gx_widget_created_test(&my_widget, &was_created);

/* If status is GX_SUCCESS, no error occurred. If “was_created” is 
GX_TRUE, the widget “my_widget” has been created. */
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_delete**

Delete widget

**Prototype**

```c
UINT  gx_widget_delete(GX_WIDGET *widget);
```

**Description**

This service deletes the widget. If the widget control block is dynamically allocated, the `gx_system_memory_free` service is invoked to free dynamically allocated storage.

**Parameters**

- **widget**: Pointer to widget

**Return Values**

- **GX_SUCCESS** (0x00): Successful widget delete
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_SYSTEM_MEMORY_ERROR** (0x30): Memory free function is not defined

**Allowed From**

Initialization and threads

**Example**

```c
/* Delete widget "my_widget". */
status = gx_widget_delete(&my_widget);

/* If status is GX_SUCCESS the widget "my_widget" has been deleted. */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_detach**

Detach widget from parent

**Prototype**

```c
UINT gx_widget_detach(GX_WIDGET *widget);
```

**Description**

This service detaches the widget from its parent.

**Parameters**

- `widget` Pointer to widget

**Return Values**

- `GX_SUCCESS` (0x00) Successful widget detach
- `GX_CALLER_ERROR` (0x11) Invalid caller of this function
- `GX_PTR_ERROR` (0x07) Invalid pointer
- `GX_INVALID_WIDGET` (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Detach widget ”my_widget” from its parent. */
status = gx_widget_detach(&my_widget);

/* If status is GX_SUCCESS the widget ”my_widget” has been detached. */
```

**See Also**

- `gx_widget_attach`, `gx_widget_back_move`, `gx_widget_background_set`,
- `gx_widget_border_draw`, `gx_widget_border_style_set`,
- `gx_widget_border_width_get`, `gx_widget_canvas_get`, `gx_widget_child_detect`,
- `gx_widget_children_draw`, `gx_widget_client_get`, `gx_widget_created`,
- `gx_widget_created_test`, `gx_widget_delete`, `gx_widget_draw`,
- `gx_widget_draw_set`, `gx_widget_event_generate`, `gx_widget_event_process`,
- `gx_widget_event_process_set`, `gx_widget_event_to_parent`, `gx_widget_find`,
- `gx_widget_front_move`, `gx_widget_height_get`, `gx_widget_hide`, `gx_widget_resize`,
- `gx_widget_shift`, `gx_widget_show`, `gx_widget_status_add`, `gx_widget_status_get`,
- `gx_widget_status_remove`, `gx_widget_status_test`, `gx_widget_style_add`,
- `gx_widget_style_get`, `gx_widget_style_remove`, `gx_widget_style_set`,
- `gx_widget_width_get`
**gx_widget_draw**

**Prototype**

```c
VOID gx_widget_draw(GX_WIDGET *widget);
```

**Description**

This service draws the widget. This function is normally called internally by the GUIX canvas refresh mechanism, but is exposed to the application to assist with implementing custom drawing functions.

**Parameters**

- `widget`  
  Pointer to widget

**Return Values**

- None

**Allowed From**

- Threads
Example

/* Write a custom widget draw function. */

VOID my_custom_widget_draw(GX_WIDGET *widget)
{
    /* Call default widget draw. */
    gx_widget_draw(widget);
    /* Add your own drawing here. */
}

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_draw_set**

Assign the widget drawing function

**Prototype**

```c
UINT gx_widget_draw_set(GX_WIDGET *widget,
                        VOID (*drawing_function) (GX_WIDGET *));
```

**Description**

This service overrides the default drawing function of the widget.

**Parameters**

- `widget` Pointer to widget
- `drawing_function` Pointer to drawing function

**Return Values**

- **GX_SUCCESS** (0x00) Successful widget drawing function override
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Define a custom drawing function. */
VOID my_drawing_function(GX_WIDGET *widget)
{
    /* Add your own drawing here. */
}

/* Set the drawing function of widget "my_widget" to
"my_drawing_function". */
status = gx_widget_draw_set(&my_widget, my_drawing_function);

/* If status is GX_SUCCESS the widget "my_widget" has the drawing
function "my_drawing_function". */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_event_generate**

Generate widget event

**Prototype**

```c
UINT gx_widget_event_generate(GX_WIDGET *widget, USHORT event_type, LONG value);
```

**Description**

This service generates a GX_SIGNAL type of event, which is a particular type or class of GX_EVENT. `gx_widget_event_generate()` encodes the 16 bit widget ID, along with the passed in event_type, into a single 32 bit GX_EVENT.gx_event_type value. The value parameter is encoded into the generated `gx_event.gx_event_payload.gx_event_longdata` field.

The generated `gx_event.target` field is always loaded with the calling widget's parent, meaning the generated event is always sent first to the parent of the generating widget.

Note that `gx_widget_event_generate` should only be used to send GX_SIGNAL range event types. For all other event types, including user defined event types, use the `gx_system_event_send()` API, which grants full control over every field of the event pushed in the GUIX event queue.

**Parameters**

- **widget**: Pointer to widget
- **event_type**: Type of event. **Appendix E** contains pre-defined GUIX events. Additional events may be added by the application.
- **value**: Additional event information

**Return Values**

- **GX_SUCCESS** (0x00): Successful widget event generation
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Threads
Example

/* Generate a redraw event for widget “my_widget”. */
status = gx_widget_event_generate(&my_widget, GX_EVENT_REDRAW, 0);
/* If status is GX_SUCCESS the redraw event for widget “my_widget”
has been generated. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_process, gx_widget_event_process_set,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get, gx_system_event_send
gx_widget_event_process

Process widget event

Prototype

UINT gx_widget_event_process(GX_WIDGET *widget, GX_EVENT *event);

Description

This is the default event processing function for all widgets. When a custom event processing function is written, the default action for any event type should always be to pass the event to the widget type upon which a widget is based. Widgets that are based on the most basic GX_WIDGET type pass use gx_widget_event_process as their default event processing function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>event</td>
<td>Pointer to event to process</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful widget event processing</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Threads

Example

/* Process event “my_event” for widget “my_widget”. */
status = gx_widget_event_process(&my_widget, &my_event);

/* If status is GX_SUCCESS the event “my_event” for widget “my_widget” has been processed. */
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process_set,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_event_process_set**

Set event processing function of widget

### Prototype

```c
UINT  gx_widget_event_process_set(GX_WIDGET *widget,
                                     UINT (*event_processing) (GX_WIDGET *, GX_EVENT *));
```

### Description

This service overrides the event processing function of the widget.

### Parameters

- **widget**
  Pointer to widget

- **event_processing**
  Pointer to new event processing function

### Return Values

- **GX_SUCCESS** (0x00)  
  Successful widget event processing override

- **GX_PTR_ERROR** (0x07)  
  Invalid pointer

- **GX_INVALID_WIDGET** (0x12)  
  Widget not valid

### Allowed From

Initialization and threads
Example

```c
UINT my_event_process(GX_TREE_VIEW *tree_view,
GX_EVENT  *event)
{
    UINT status = GX_SUCCESS;

    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;

    default:
        /* Pass all other events to the default tree view
        event processing */
        status = gx_tree_view_event_process(tree_view, event);
        break;
    }
    return status;
}
/* Use “my_event_process” to process events for widget
“my_tree_view”. */
status = gx_widget_event_process_set((GX_WIDGET *)my_tree_view,
    (VOID (*)(GX_WIDGET *))my_event_process);
/* If status is GX_SUCCESS all event processing for widget
“my_tree_view” is handled by “my_event_process”. */
```

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_event_to_parent

Send event to widget's parent

Prototype

UINT  gx_widget_event_to_parent(GX_WIDGET *widget,
                                        GX_EVENT *event);

Description

This service sends an event to the widget's parent.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>event</td>
<td>Pointer to the event</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successfully sent event to widget's parent</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
</tbody>
</table>

Allowed From

Threads

Example

/* Send my_event to the widget's parent */
status = gx_widget_event_to_parent(&my_widget, my_event);

/* If status is GX_SUCCESS the event has been delivered to the
parent of my_widget. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_fill_color_set**

Set widget background color

**Prototype**

```c
UINT gx_widget_fill_color_set(GX_WIDGET *widget,
                                  GXRESOURCE_ID normal_color_id,
                                  GXRESOURCE_ID selected_color_id,
                                  GXRESOURCE_ID disabled_color_id);
```

**Description**

This service sets the widget background colors.

**Parameters**

- **widget**: Pointer to widget
- **normal_color_id**: Resource ID of the fill color in normal state. [Appendix A](#) contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **selected_color_id**: Resource ID of the fill color when the widget gain focus. [Appendix A](#) contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.
- **disabled_color_id**: Resource ID of the fill color when the style GX_STYLE_ENABLED is not set. [Appendix A](#) contains pre-defined color Resource IDs. Note that the application may add custom color Resource IDs as well.

**Return Values**

- **GX_SUCCESS** (0x00): Successfully set widget fill color
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads
Example

/* Set background of "my_widget". */
status = gx_widget_fill_color_set(&my_widget,
   GX_COLOR_ID_NORMAL_FILL,
   GX_COLOR_ID_SELECTED_FILL,
   GX_COLOR_ID_DISABLED_FILL);

/* If status is GX_SUCCESS the widget "my_widget" background has been set. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_border_draw,
gx_widget_border_style_set, gx_widget_border_width_get,
gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw,
gx_widget_client_get, gx_widget_create, gx_widget_created_test,
gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set,
gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_find**

Find child widget of parent widget

Prototype

```c
UINT gx_widget_find(GX_WIDGET *parent, USHORT widget_id, INT search_depth, GX_WIDGET **return_widget);
```

Description

This service searches through the children of the specified parent looking for a widget with the requested ID value.

Parameters

- **parent**: Pointer to parent widget from which search is started
- **widget_id**: Widget ID to search for
- **search_depth**: Defines the recursive nesting level into which the function will search child widgets. If this value is <= 0, only immediate children of the parent widget are searched. If this value is GX_SEARCH_DEPTH_INFINITE, all children of all child widgets are exhaustively searched. For any other value > 0, this value limits how deeply nested this function will search through child widgets looked for the requested widget ID.
- **return_widget**: Pointer to destination for found widget

Return Values

- **GX_SUCCESS** (0x00): Successful widget find
- **GX_NOT_FOUND** (0x09): Widget not found
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads
Example

GX_WIDGET *widget_found;

/* Find widget "my widget". */
status = gx_widget_find(&my_widget, GX_SEARCH_DEPTH_INFINITE
                      MY_WIDGET_ID, &widget_found);

/* If status is GX_SUCCESS, the pointer "widget_found" contains the
pointer to the widget found. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_first_child_get

Return pointer to first child widget

Prototype

UINT gx_widget_first_child_get(GX_WIDGET *parent,
                              GX_WIDGET **widget_return);

Description

GUIX maintains a tree structured list of parent and child widgets.
This service returns a pointer to the first child widget of the parent.

Parameters

- **parent**: Pointer to parent widget
- **widget_return**: Pointer to return widget pointer

Return Values

- **GX_SUCCESS** (0x00): pointer returned
- **GX_PTR_ERROR** (0x07): Invalid widget pointer
- **GX_INVALID_WIDGET** (0x12): Invalid widget

Allowed From

Threads

Example

```c
/* Retrieve child widget pointer. */

GX_WIDGET *get_child_widget(GX_WIDGET *parent)
{
    GX_WIDGET *child;
    UINT status;

    status = gx_widget_first_child_get(parent, &child);
    if (status == GX_SUCCESS)
    {
        return child;
    }
    return GX_NULL;
}
```

See Also

gx_widget_last_child_get, gx_widget_next_sibling_get, gx_widget_parent_get,
gx_widget_previous_sibling_get, gx_widget_top_visible_child_find
### gx_widget_focus_next

**Move focus to next widget in navigation order**

**Prototype**

```c
UINT gx_widget_focus_next(GX_WIDGET *widget);
```

**Description**

This service moves focus to the next sibling widget in the linked list of widgets that accept focus.

**Parameters**

- `widget` Pointer to widget control block

**Return Values**

- **GX_SUCCESS** (0x00) focus was moved
- **GX_FAILURE** (0x00) focus was not moved
- **GX_PTR_ERROR** (0x07) Invalid widget pointer
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_INVALID_WIDGET** (0x12) Invalid widget

**Allowed From**

Initialization and threads

**Example**

```c
/* Move focus to next widget in navigation order. */
status = gx_widget_focus_next(&my_widget);

/* If status is GX_SUCCESS the focus has been moved to the next
widget in the navigation order */
```

**See Also**

- `gx_widget_focus_previous`
**gx_widget_focus_previous**

Move focus to previous widget in navigation order

Prototype

```c
UINT gx_widget_focus_previous(GX_WIDGET *widget);
```

Description

This service moves focus to the previous widget in the navigation order.

Parameters

- `widget` Pointer to widget that current has input focus.

Return Values

- `GX_SUCCESS` (0x00) focus was moved
- `GX_FAILURE` (0x00) focus was not moved

Allowed From

Initialization and threads

Example

```c
/* Move focus to previous widget in navigation order. */
status = gx_widget_focus_previous(&my_widget);

/* If status is GX_SUCCESS the input focus has been moved to the previous widget. */
```

See Also

- `gx_widget_focus_next`
**gx_widget_font_get**

Get font for specified resource ID

Prototype

```c
UINT gx_widget_font_get(GX_WIDGETG *widget,
                         GX_RESOURCE_ID resource_id,
                         GX_FONT **return_font);
```

Description

This service retrieves the font associated with the specified resource ID from the font table of the display on which this widget is visible. This function should only be called by a visible widget.

Parameters

- **widget**: Pointer to widget control block
- **resource_id**: Resource ID of font
- **return_font**: Pointer to destination for font pointer

Return Values

- **GX_SUCCESS** (0x00): Successfully retrieved font
- **GX_INVALID_RESOURCE_ID** (0x33): Invalid resource ID
- **GX_INVALID_CANVAS** (0x20): Widget canvas not valid or widget is invisible
- **GX_PTR_ERROR** (0x07): Invalid widget pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads

Example

```c
GX_FONT *my_font;

/* Get font for MY_FONT_ID. */
status = gx_widget_font_get(widget, MY_FONT_RESOURCE_ID, &my_font);

/* If status is GX_SUCCESS the font pointer has been retrieved in "my_font". */
```

See Also

- `gx_widget_color_get`, `gx_widget_pixelmap_get`
**gx_widget_free**  
Release memory associated with a widget

**Prototype**

```c
UINT  gx_widget_free(GX_WIDGETG *widget);
```

**Description**

This service releases the memory associated with a widget control block.

**Parameters**

- **widget**  
  Pointer to widget control block  
- **resource_id**  
  Resource ID of font  
- **return_font**  
  Pointer to destination for font pointer

**Return Values**

- **GX_SUCCESS**  
  (0x00) Successfully freed widget  
- **GX_SYSTEM_MEMPRY_ERROR**  
  (0x30) Memory free function is not defined  
- **GX_PTR_ERROR**  
  (0x07) Invalid widget pointer

**Allowed From**

Initialization and threads
Example

GX_WIDGET widget;
UINT status;

status = gx_widget_allocate(&widget, sizeof(GX_WIDGET))

/* Free a runtime allocated widget. */
if (status == GX_SUCCESS)
{
    status = gx_widget_free(widget);
}

/* If status is GX_SUCCESS the memory that allocated to the widget
has been released. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_front_move

Move widget to front

Prototype

UINT gx_widget_front_move(GX_WIDGET *widget, GX_BOOL *return_moved);

Description

This service moves the widget to the front in the parent Z-order list of child widgets.

Parameters

widget Pointer to widget to move
return_moved Pointer to destination for indication widget was moved

Return Values

GX_SUCCESS (0x00) Successful widget move to front
GX_PTR_ERROR (0x07) Invalid pointer
GX_NO_CHANGE (0x08) Widget already in front
GX_CALLER_ERROR (0x11) Invalid caller of this function
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads
Example

GX_BOOL widget_moved;

/* Move widget "my_widget" to the front. */
status = gx_widget_front_move(&my_widget, &widget_moved);

/* If status is GX_SUCCESS and "widget_moved" is GX_TRUE, the widget "my_widget" was moved to the front. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_height_get**

Get widget height

**Prototype**

```c
UINT gx_widget_height_get(GX_WIDGET *widget,
                            UINT *return_height);
```

**Description**

This service gets the widget height.

**Parameters**

- **widget**: Pointer to widget
- **return_height**: Pointer to destination for widget height

**Return Values**

- **GX_SUCCESS** (0x00): Successful widget height get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads
Example

GX_VALUE widget_height;

/* Get height for widget ”my_widget”. */
status = gx_widget_height_get(&my_widget, &widget_height);

/* If status is GX_SUCCESS the height of the widget is contained in ”widget_height” . */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_hide, gx_widget_resize, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get


**gx_widget_hide**  
Hide widget

**Prototype**

```c
UINT  gx_widget_hide(GX_WIDGET *widget);
```

**Description**

This service hides the widget. This widget is still attached to it's parent, but it is not allowed to draw on the canvas.

**Parameters**

- `widget`  
  Pointer to widget

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  Successful widget hide
- **GX_CALLER_ERROR**  
  (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR**  
  (0x07)  
  Invalid pointer
- **GX_INVALID_WIDGET**  
  (0x12)  
  Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Hide widget "my_widget". */
status = gx_widget_hide(&my_widget);

/* If status is GX_SUCCESS the widget "my_widget" is hidden. */
```

**See Also**

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,  
gx_widget_border_draw, gx_widget_border_style_set,  
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,  
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,  
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,  
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,  
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,  
gx_widget_front_move, gx_widget_height_get, gx_widget_resize, gx_widget_shift,  
gx_widget_show, gx_widget_status_add, gx_widget_style_get,  
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,  
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,  
gx_widget_width_get
**gx_widget_last_child_get**

Return pointer to last child widget

Prototype

```c
UINT gx_widget_last_child_get(GX_WIDGET *parent,
                               GX_WIDGET **widget_return);
```

Description

GUIX maintains a tree structured list of parent and child widgets. This service returns a pointer to the last child widget of the parent.

Parameters

- **parent**
  - Pointer to parent widget
- **widget_return**
  - Pointer to return widget pointer

Return Values

- **GX_SUCCESS** (0x00)
  - Pointer returned
- **GX_PTR_ERROR** (0x07)
  - Invalid widget pointer
- **GX_INVALID_WIDGET** (0x12)
  - Invalid widget

Allowed From

Threads

Example

```c
/* Retrieve child widget pointer. */
GX_WIDGET *get_last_child_widget(GX_WIDGET *parent)
{
    GX_WIDGET *child;
    UINT status;

    status = gx_widget_last_child_get(parent, &child);
    if (status == GX_SUCCESS)
    {
        return child;
    }
    return GX_NULL;
}
```

See Also

- `gx_widget_first_child_get`
- `gx_widget_next_sibling_get`
- `gx_widget_parent_get`
- `gx_widget_previous_sibling_get`
- `gx_widget_top_visible_child_find`
**gx_widget_next_sibling_get**

Return pointer to next sibling of current widget

**Prototype**

```c
UINT gx_widget_next_sibling_get(GX_WIDGET *current,
                                GX_WIDGET **widget_return);
```

**Description**

GUIX maintains a tree structured list of parent and child widgets. This service returns a pointer to the next sibling of the current widget.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>Pointer to current widget</td>
</tr>
<tr>
<td>widget_return</td>
<td>Pointer to return widget pointer</td>
</tr>
</tbody>
</table>

**Return Values**

- **GX_SUCCESS**: (0x00) pointer returned
- **GX_PTR_ERROR**: (0x07) Invalid widget pointer
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

**Allowed From**

Threads

**Example**

```c
/* Retrieve next sibling widget pointer. */

GX_WIDGET *get_next(GX_WIDGET *current)
{
    GX_WIDGET *sibling;
    UINT status;

    status = gx_widget_next_sibling_get(current, &sibling);
    if (status == GX_SUCCESS)
    {
        return sibling;
    }
    return GX_NULL;
}
```

**See Also**
gx_widget_first_child_get, gx_widget_last_child_get, gx_widget_parent_get,
gx_widget_previous_sibling_get, gx_widget_top_visible_child_find
**gx_widget_parent_get**  
Return pointer to parent of current widget

**Prototype**

```c
UINT gx_widget_parent_get(GX_WIDGET *current,  
                         GX_WIDGET **widget_return);
```

**Description**

GUIX maintains a tree structured list of parent and child widgets. This service returns a pointer to the parent of the current widget.

**Parameters**

- `current`  
  Pointer to current widget

- `widget_return`  
  Pointer to return widget pointer

**Return Values**

- **GX_SUCCESS**  
  (0x00)  
  pointer returned

- **GX_PTR_ERROR**  
  (0x07)  
  Invalid widget pointer

- **GX_INVALID_WIDGET**  
  (0x12)  
  Invalid widget

**Allowed From**

Threads

**Example**

```c
/* Retrieve parent widget */

GX_WIDGET *get_parent(GX_WIDGET *current)  
{
   GX_WIDGET *parent;
   UINT status;

   status = gx_widget_parent_get(current, &parent);
   if (status == GX_SUCCESS)  
   {  
      return parent;
   }
   return GX_NULL;
}
```

**See Also**

- `gx_widget_first_child_get`, `gx_widget_last_child_get`, `gx_widget_next_sibling_get`, `gx_widget_previous_sibling_get`, `gx_widget_top_visible_child_find`
gx_widget_pixelmap_get

Get pixelmap

Prototype

UINT gx_widget_pixelmap_get(GX_WIDGET *widget,
    GX_RESOURCE_ID resource_id,
    GX_PIXELMAP **return_pixelmap);

Description

This service gets the pixelmap associated with the supplied resource ID. This service should only be called for visible widgets.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget control block</td>
</tr>
<tr>
<td>pixelmap_id</td>
<td>Pixelmap resource ID</td>
</tr>
<tr>
<td>return_pixelmap</td>
<td>Pointer to pixelmap destination pointer</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00) Successful pixelmap get
- **GX_INVALID_RESOURCE_ID** (0x33) Invalid resource ID
- **GX_INVALID_CANVAS** (0x20) Widget canvas not valid or widget is invisible
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer

Allowed From

Initialization and threads

Example

GX_PIXELMAP *my_pixelmap;

/* Get the pixelmap associated with MY_PIXELMAP_ID. */
status = gx_widget_pixelmap_get(widget, MY_PIXELMAP_RESOURCE_ID,
    &my_pixelmap);

/* If status is GX_SUCCESS, “my_pixelmap” contains the pixmap pointer. */

See Also

gx_widget_color_get, gx_widget_font_get
gx_widget_previous_sibling_get

Return pointer to previous sibling of the current widget

Prototype

UINT gx_widget_previous_sibling_get(GX_WIDGET *current,
                                   GX_WIDGET **widget_return);

Description

GUIX maintains a tree structured list of parent and child widgets. This service returns a pointer to the previous sibling of the current widget.

Parameters

Parameter | Description
---|---
**current** | Pointer to current widget
**widget_return** | Pointer to return widget pointer

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) pointer returned</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid widget pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Invalid widget</td>
</tr>
</tbody>
</table>

Allowed From

Threads

Example

```c
/* Retrieve previous sibling widget */

GX_WIDGET *get_previous(GX_WIDGET *current)
{
    GX_WIDGET *sibling;
    UINT status;

    status = gx_widget_previous_sibling_get(current, &sibling);
    if (status == GX_SUCCESS)
    {
        return sibling;
    } else
    {
        return GX_NULL;
    }
}
```

See Also
gx_widget_first_child_get, gx_widget_last_child_get, gx_widget_next_sibling_get,
gx_widget_parent_get, gx_widget_top_visible_child_find
gx_widget_resize

Prototype

UINT  gx_widget_resize(GX_WIDGET *widget, GX_RECTANGLE *new_size);

Description

This service resizes the widget. If the widget is visible, it is automatically invalidated and queued for re-drawing.

Parameters

- **widget**: Pointer to widget
- **new_size**: New widget size

Return Values

- **GX_SUCCESS**: (0x00) Successful widget resize
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

Initialization and threads
**Example**

GX_RECTANGLE new_size;

gx_utility_rectangle_define(&new_size, 0, 0, 100, 100);

/* Resize widget "my_widget". */
status = gx_widget_resize(&my_widget, &new_size);

/* If status is GX_SUCCESS the widget "my_widget" has been resized. */

**See Also**

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_shift,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
gx_widget_shift

Shift widget

Prototype

UINT gx_widget_shift(GX_WIDGET *widget, GX_VALUE x_shift,
                     GX_VALUE y_shift, GX_BOOL mark_dirty);

Description

This service shifts the widget and optionally marks it as dirty.

Parameters

- **widget**: Pointer to widget
- **x_shift**: Number of pixels to shift on x-axis
- **y_shift**: Number of pixels to shift on y-axis
- **mark_dirty**: GX_TRUE to indicate dirty, otherwise GX_FALSE

Return Values

- **GX_SUCCESS** (0x00): Successful widget shift
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads
Example

/* Shift widget "my_widget". */
status = gx_widget_shift(&my_widget, 10, 20, GX_FALSE);

/* If status is GX_SUCCESS the widget "my_widget" has been shifted. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set,
gx_widget_width_get
**gx_widget_show**

Show widget

**Prototype**

```c
UINT gx_widget_show(GX_WIDGET *widget);
```

**Description**

This service shows the widget. The widget will become visible only if it is attached to a parent and the parent widget is also visible.

**Parameters**

- **widget**: Pointer to widget

**Return Values**

- **GX_SUCCESS** (0x00) Successful widget show
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Show widget "my_widget". */
status = gx_widget_show(&my_widget);

/* If status is GX_SUCCESS the widget "my_widget" has been shown. */
```

**See Also**

gx_widget_attach, gx_widget_back_move, gx_widget_background_set, gx_widget_border_draw, gx_widget_border_style_set, gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect, gx_widget_children_draw, gx_widget_client_get, gx_widget_created, gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw, gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process, gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find, gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize, gx_widget_shift, gx_widget_status_add, gx_widget_status_get, gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add, gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set, gx_widget_width_get
**gx_widget_status_add**

Add widget status

**Prototype**

```c
UINT  gx_widget_status_add(GX_WIDGET *widget, ULONG status)
```

**Description**

This service adds any combination of status flags to the specified widget.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>status</td>
<td>Status to add</td>
</tr>
</tbody>
</table>

**Return Values**

- **GX_SUCCESS** (0x00) Successful widget status add
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Add status to widget "my_widget". */
status = gx_widget_status_add(&my_widget, status_to_add);

/* If status is GX_SUCCESS the widget "my_widget" status was. */
```

**See Also**

- `gx_widget_attach`, `gx_widget_back_move`, `gx_widget_background_set`, `gx_widget_border_draw`, `gx_widget_border_style_set`, `gx_widget_border_width_get`, `gx_widget_canvas_get`, `gx_widget_child_detect`, `gx_widget_children_draw`, `gx_widget_client_get`, `gx_widget_created`, `gx_widget_created_test`, `gx_widget_delete`, `gx_widget_detach`, `gx_widget_draw`, `gx_widget_draw_set`, `gx_widget_event_generate`, `gx_widget_event_process`, `gx_widget_event_process_set`, `gx_widget_event_to_parent`, `gx_widget_find`, `gx_widget_front_move`, `gx_widget_height_get`, `gx_widget_hide`, `gx_widget_resize`, `gx_widget_shift`, `gx_widget_show`, `gx_widget_status_get`, `gx_widget_status_remove`, `gx_widget_status_test`, `gx_widget_style_add`, `gx_widget_style_get`, `gx_widget_style_remove`, `gx_widget_style_set`, `gx_widget_width_get`
gx_widget_status_get

Get widget status

Prototype

UINT gx_widget_status_get(GX_WIDGET *widget,
                      ULONG *return_status)

Description

This service retrieves status flags from the widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>return_status</td>
<td>Pointer to the status being returned</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successful widget status get</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET (0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

ULONT get_status;

/* Retrieve status flag from widget ”my_widget”. */
status = gx_widget_status_get(&my_widget, &get_status);

/* If status is GX_SUCCESS the status from widget ”my_widget” is saved to ”get_status”. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_remove,
gx_widget_status_test, gx_widget_style_add, gx_widget_style_get,
gx_widget_style_remove, gx_widget_style_set, gx_widget_width_get
# gx_widget_status_remove

Remove widget status

## Prototype

```c
UINT gx_widget_status_remove(GX_WIDGET *widget, ULONG status)
```

## Description

This service removes the specified status flags from the widget's internal status variable.

## Parameters

- **widget**
  - Pointer to widget
- **status**
  - Status to remove

## Return Values

- **GX_SUCCESS** (0x00)
  - Successful widget status removal
- **GX_PTR_ERROR** (0x07)
  - Invalid pointer
- **GX_INVALID_WIDGET** (0x12)
  - Widget not valid

## Allowed From

- Initialization and threads

## Example

```c
/* Remove status of widget "my_widget". */
status = gx_widget_status_remove(&my_widget, status_to_remove);

/* If status is GX_SUCCESS, the status flags are removed from the widget "my_widget". */
```

## See Also

- `gx_widget_attach`
- `gx_widget_back_move`
- `gx_widget_background_set`
- `gx_widget_border_draw`
- `gx_widget_border_style_set`
- `gx_widget_border_width_get`
- `gx_widget_canvas_get`
- `gx_widget_child_detect`
- `gx_widget_children_draw`
- `gx_widget_client_get`
- `gx_widget_created`
- `gx_widget_created_test`
- `gx_widget_delete`
- `gx_widget_detach`
- `gx_widget_draw`
- `gx_widget_draw_set`
- `gx_widget_event_generate`
- `gx_widget_event_process`
- `gx_widget_event_process_set`
- `gx_widget_event_to_parent`
- `gx_widget_find`
- `gx_widget_front_move`
- `gx_widget_height_get`
- `gx_widget_hide`
- `gx_widget_resize`
- `gx_widget_shift`
- `gx_widget_show`
- `gx_widget_status_add`
- `gx_widget_status_get`
- `gx_widget_status_test`
- `gx_widget_style_add`
- `gx_widget_style_get`
- `gx_widget_style_remove`
- `gx_widget_style_set`
- `gx_widget_width_get`
gx_widget_status_test

Test widget status

Prototype

UINT gx_widget_status_test(GX_WIDGET *widget, ULONG status,
                            GX_BOOL *return_test);

Description

This service tests the status flags of the specified widget and stores
the result in the memory pointed by "return_test".

Parameters

widget Pointer to widget
status Status to test
return_status Pointer to destination for result of test

Return Values

GX_SUCCESS (0x00) Successful widget status test
GX_PTR_ERROR (0x07) Invalid pointer
GX_INVALID_WIDGET (0x12) Widget not valid

Allowed From

Initialization and threads

Example

GX_BOOL test_result;

/* Test status of widget "my_widget". */
status = gx_widget_status_test(&my_widget, status_to_test,
                              &test_result);

/* If status is GX_SUCCESS the widget "my_widget" status was tested
and the result in "test_result". */

See Also

gx_widget_status_add, gx_widget_status_get, gx_widget_status_remove,
gx_widget_style_add, gx_widget_style_get, gx_widget_style_remove,
gx_widget_style_set
gx_widget_string_get
Retrieve string associated with a visible widget and string ID (deprecated)

Prototype

UINT gx_widget_string_get(GX_WIDGET *widget,
                          GX_RESOURCE_ID string_id,
                          GX_CONST GX_CHAR **string);

Description

This service is deprecated in favor of gx_widget_string_get_ext().
This service returns the string table entry for the given string ID value. This service is similar to gx_display_string_get, except the active display is determined automatically rather than being passed in by the caller. This service can only be used for widgets which are visible, i.e. the display associated with this widget is known.

Parameters

widget
Pointer to widget
string_id
String ID value from resources header
string
Address of variable to return string

Return Values

GX_SUCCESS
(0x00) Successful widget status test
GX_PTR_ERROR
(0x07) Invalid pointer
GX_INVALID_WIDGET
(0x12) Widget not valid

Allowed From

Initialization and threads

Example

GX_CONST GX_CHAR *string;

/* Test status of widget "my_widget". */
status = gx_widget_string_get(&my_widget, GX_STRING_ID_SHUTDOWN, &string);

/* If status is GX_SUCCESS the string has been retrieved */

See Also

gx_display_string_get, gx_display_active_language_set
gx_widget_string_get_ext
Retrieve string associated with a visible widget and string ID

Prototype

UINT  gx_widget_string_get(GX_WIDGET *widget,
                           GX_RESOURCE_ID string_id,
                           GX_CONST GX_STRING *string);

Description

This service returns the string table entry for the given string ID value. This service is similar to gx_display_string_get, except the active display is determined automatically rather than being passed in by the caller. This service can only be used for widgets which are visible, i.e. the display associated with this widget is known.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>string_id</td>
<td>String ID value from resources header</td>
</tr>
<tr>
<td>string</td>
<td>Address of variable to return string</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>Successful widget status</td>
</tr>
<tr>
<td>(0x00)</td>
<td>test</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>(0x07)</td>
<td></td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>Widget not valid</td>
</tr>
<tr>
<td>(0x12)</td>
<td></td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_STRING string;

/* Test status of widget “my_widget” */
status = gx_widget_string_get_ext(&my_widget,
                                  GX_STRING_ID_SHUTDOWN, &string);

/* If status is GX_SUCCESS the string has been retrieved */

See Also

gx_display_string_get, gx_display_active_language_set
**gx_widget_style_add**

**Add widget style**

**Prototype**

```c
UINT gx_widget_style_add(GX_WIDGET *widget, ULONG style)
```

**Description**

This service adds a style to the widget. In addition, the following actions are taken.

If the added style is GX_STYLE_TRANSPARENT, status GX_STATUS_TRANSPARENT will be added.

If the added style is GX_STYLE_ENABLED, status GX_STATUS_SELECTABLE will be added.

If the widget is visible, it is automatically invalidated and queued for re-drawing.

**Parameters**

- **widget**
  - Pointer to widget

- **style**
  - New style to add. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.

**Return Values**

- **GX_SUCCESS** (0x00) Successful widget style add
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Add style to widget “my_widget”. */
status = gx_widget_style_add(&my_widget, GX_STYLE_BORDER_RAISED);

/* If status is GX_SUCCESS, the style was successfully applied to the widget “my_widget”. */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_get,
gx_widget_style_remove, gx_widget_style_set, gx_widget_width_get
**gx_widget_style_get**

Get widget style

**Prototype**

```c
UINT gx_widget_style_get(GX_WIDGET *widget, ULONG *return_style)
```

**Description**

This service retrieves style flag from the widget.

**Parameters**

- `widget` (Pointer to widget)
- `return_style` (Pointer to the style being returned)

**Return Values**

- **GX_SUCCESS** (0x00) Successfully retrieved widget style
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid

**Allowed From**

Initialization and threads
Example

/* Retrieve style from widget into “style”. */
status = gx_widget_style_get(&my_widget, &style);

/* If status is GX_SUCCESS the style flag from widget is saved in “style”. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_remove,
gx_widget_style_add, gx_widget_style_set, gx_widget_width_get
gx_widget_style_remove

Remove widget style

Prototype

UINT gx_widget_style_remove(GX_WIDGET *widget, ULONG style)

Description

This service removes a style from the widget. In addition, the following actions are taken.

If the removed style is GX_STYLE_TRANSPARENT, status GX_STATUS_TRANSPARENT will be removed.

If the removed style is GX_STYLE_ENABLED, status GX_STATUS_SELECTABLE will be removed.

If the widget is visible, it is automatically invalidated and queued for re-drawing.

Parameters

- **widget**: Pointer to widget
- **style**: Style to remove. Appendix D contains pre-defined general styles for all widgets as well as widget-specific styles.

Return Values

- **GX_SUCCESS**: (0x00) Successful widget style remove
- **GX_CALLER_ERROR**: (0x11) Invalid caller of this function
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

Allowed From

Initialization and threads
Example

/* Remove style from widget "my_widget". */
status = gx_widget_style_remove(&my_widget,
                               GX_STYLE_BORDER_RAISED);

/* If status is GX_SUCCESS the GX_STYLE_BORDER_RAISED style was
removed from the widget "my_widget". */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_set, gx_widget_width_get
**gx_widget_style_set**

Set widget style

**Prototype**

```c
UINT gx_widget_style_set(GX_WIDGET *widget, ULONG style)
```

**Description**

This service sets a style to the widget.

If the set style includes `GX_STYLE_TRANSPARENT`, status `GX_STATUS_TRANSPARENT` will be added, otherwise the status will be removed.

If the set style includes `GX_STYLE_ENABLED`, status `GX_STATUS_SELECTABLE` will be added, otherwise the status will be removed.

If the widget is visible, it is automatically invalidated and queued for re-drawing.

**Parameters**

- **widget**
  - Pointer to widget
- **style**
  - Style to set. Appendix D contains predefined general styles for all widgets as well as widget-specific styles.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td><code>GX_SUCCESS</code> Successful widget style set</td>
</tr>
<tr>
<td>0x11</td>
<td><code>GX_CALLER_ERROR</code> Invalid caller of this function</td>
</tr>
<tr>
<td>0x07</td>
<td><code>GX_PTR_ERROR</code> Invalid pointer</td>
</tr>
<tr>
<td>0x12</td>
<td><code>GX_INVALID_WIDGET</code> Widget not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

- Initialization and threads
Example

/* Set style GX_STYLE_TRANSPARENT to the widget “my_widget”. */
status = gx_widget_style_set(&my_widget, GX_STYLE_TRANSPARENT);

/* If status is GX_SUCCESS the widget “my_widget” style is set to
GX_STYLE_TRANSPARENT. */

See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_set, gx_widget_width_get
**gx_widget_text_blend**

Blend text assigned to widget (deprecated)

Prototype

```c
UINT gx_widget_text_blend(GX_WIDGET *widget, UINT *tColor,
                         UINT font_id, GX_CHAR *string,
                         INT x_offset, INT y_offset, UCHAR alpha)
```

Description

This service is deprecated in favor of gx_widget_text_blend_ext().

This service blends the specified text over a widget using current brush and text alignment.

Parameters

- **widget**: Pointer to widget
- **tColor**: Text color
- **font_id**: Font Id
- **string**: Drawing string
- **x_offset**: Drawing position adjustment
- **y_offset**: Drawing position adjustment
- **alpha**: Blending value 0-255

Return Values

- **GX_SUCCESS** (0x00): Successful widget width get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_STRING_LENGTH** (0x34): Invalid string length

Allowed From

Initialization and threads

Example

```c
/* Blend "my_string" over "my_widget" given alpha value 120. */
status = gx_widget_text_blend(&my_widget, my_text_color, my_font_id,
                              &my_string, xoffset, yoffset, 120);

/* If status is GX_SUCCESS "my_string" has been blend to "my_widget". */
```
See Also

gx_widget_text_blend_ext
**gx-widget-text-blend-ext**

Blend text assigned to widget

**Prototype**

```c
UINT gx_widget_text_blend(GX_WIDGET *widget, UINT *tColor,
    UINT font_id, GX_CONST GX_STRING *string,
    INT x_offset, INT y_offset, UCHAR alpha)
```

**Description**

This service is deprecated in favor of gx-widget-text-blend-ext().

This service renders a string over the specified widget using the current brush and text alignment and specified color, font, and x,y offset.

**Parameters**

- `widget`: Pointer to widget
- `tColor`: Text color
- `font_id`: Font Id
- `string`: Drawing string
- `x_offset`: Drawing position adjustment
- `y_offset`: Drawing position adjustment
- `alpha`: Blending value 0-255

**Return Values**

- **GX_SUCCESS** (0x00): Successful widget width get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_STRING_LENGTH** (0x34): Invalid string length

**Allowed From**

Initialization and threads

**Example**

```c
gx_string render_string;
render_string.gx_string_ptr = "Hello";
render_string.gx_string_length = strlen(render_string.gx_string_ptr);

/* Blend “my_string” over “my.widget” given alpha value 120. */
status = gx_widget_text_blend_ext(&my_widget,
```
my_text_color,
my_font_id,
&render_string, xoffset, yoffset, 120);

/* If status is GX_SUCCESS "my_string" has been blend to "my_widget". */

See Also

gx_widget_text_draw_ext
**gx_widget_text_draw**

Draw text assigned to widget (deprecated)

**Prototype**

```c
VOID gx_widget_text_draw(GX_WIDGET *widget, UINT *tColor,
                         UINT font_id, GX_CHAR *string,
                         INT x_offset, INT y_offset)
```

**Description**

This service is deprecated in favor of `gx_widget_text_draw_ext()`.

This service draws the specified text over a widget using current brush and text alignment.

**Parameters**

- `widget` Pointer to widget
- `tColor` Text color
- `font_id` Font Id
- `string` Drawing string
- `x_offset` Drawing position adjustment
- `y_offset` Drawing position adjustment

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
/* Write a custom widget draw function. */

VOID my_custom_widget_draw(GX_WIDGET *widget)
{
    /* Add your own background raw here. */

    /* Call widget text draw. */
    gx_widget_text_draw(widget, my_text_color, my_font_id
                        &my_string, xoffset, yoffset);
}
```

**See Also**

`gx_widget_text_blend`, `gx_widget_text_id_draw`,

GUIX User Guide 805
**gx_widget_text_draw_ext**

Draw text assigned to widget

**Prototype**

```c
VOID gx_widget_text_draw(GX_WIDGET *widget, UINT *tColor,
                         UINT font_id, GX_CONST GX_STRING *string,
                         INT x_offset, INT y_offset)
```

**Description**

This service draws the specified text over a widget using current brush and text alignment.

**Parameters**

- **widget**: Pointer to widget
- **tColor**: Text color
- **font_id**: Font Id
- **string**: Drawing string
- **x_offset**: Drawing position adjustment
- **y_offset**: Drawing position adjustment

**Return Values**

None

**Allowed From**

Threads

**Example**

```c
gx_string render_string;
render_string.gx_string_ptr = "Hello";
render_string.gx_string_length = strlen(render_string.gx_string_ptr);

/* Write a custom widget draw function. */
VOID my_custom_widget_draw(GX_WIDGET *widget)
{
    /* Add your own background draw here. */
    /* Call widget text draw. */
    gx_widget_text_draw_ext(widget, my_text_color, my_font_id
                            &render_string, xoffset, yoffset);
}
```

**See Also**

gx_widget_text_blend, gx_widget_text_id_draw,
**gx_widget_text_id_draw**

Draw text assigned to widget

**Prototype**

```c
VOID gx_widget_text_id_draw(GX_WIDGET *widget, UINT *tColor,
                             UINT font_id, UINT text_id,
                             INT x_offset, INT y_offset)
```

**Description**

This service draws text over a widget given a text id.

**Parameters**

- `widget`: Pointer to widget
- `tColor`: Text color
- `font_id`: Font Id
- `text_id`: Text Id
- `x_offset`: Drawing position adjustment
- `y_offset`: Drawing position adjustment

**Return Values**

None

**Allowed From**

Initialization and threads

**Example**

```c
/* Write a custom widget draw function. */

VOID my_custom_widget_draw(GX_WIDGET *widget)
{
    /* Add your own background raw here. */

    /* Call widget text id draw. */
    gx_widget_text_id_draw(widget, my_text_color, my_font_id
                            &my_string_id, xoffset, yoffset);
}
```

**See Also**

`gx_widget_text_blend, gx_widget_text_draw`
**gx_widget_top_visible_child_find**

Return pointer to visible child that is top of Z order

Prototype

```c
UINT gx_widget_top_visible_child_find(GX_WIDGET *current,
                                        GX_WIDGET **widget_return);
```

Description

GUIX maintains a tree structured list of parent and child widgets. This service returns a pointer to the topmost visible child of the current widget.

Parameters

- **current**: Pointer to current widget
- **widget_return**: Pointer to return widget pointer

Return Values

- **GX_SUCCESS**: (0x00) pointer returned
- **GX_PTR_ERROR**: (0x07) Invalid widget pointer
- **GX_INVALID_WIDGET**: (0x12) Invalid widget

Allowed From

Threads

Example

```c
/* Retrieve topmost visible widget on the display */

GX_WIDGET *get_top_window(GX_WINDOW_ROOT *root) {
    GX_WIDGET *top_window;
    UINT status;

    status = gx_widget_top_visible_child_find(root, &top_window);
    if (status == GX_SUCCESS) {
        return top_window;
    }
    return GX_NULL;
}
```

See Also

- `gx_widget_first_child_get`
- `gx_widget_last_child_get`
- `gx_widget_next_sibling_get`
- `gx_widget_previous_sibling_get`
gx_widget_type_find

Search for a widget of the requested type

Prototype

UINT gx_widget_type_find(GX_WIDGET *parent, USHORT widget_id,
                         GX_WIDGET **return_widget)

Description

This service searches for a widget of the requested type.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget</td>
</tr>
<tr>
<td>return_width</td>
<td>Pointer to destination for widget width</td>
</tr>
</tbody>
</table>

Return Values

- **GX_SUCCESS** (0x00)  
  Successful widget width get
- **GX_PTR_ERROR** (0x07)  
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)  
  Widget not valid

Allowed From

Initialization and threads

Example

```c
GX_WIDGET *retrieved_widget;

/* Find horizontal scrollbar under "parent_widget". */
status = gx_widget_type_find(&parent_widget,
                             GX_TYPE_HORIZONTAL_SCROLL_BAR, &retrieved_widget);

/* If status is GX_SUCCESS, the horizontal scrollbar widget is retrieved. */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set
**gx_widget_width_get**

Get widget width

**Prototype**

```c
UINT gx_widget_width_get(GX_WIDGET *widget,
                          GX_VALUE *return_width)
```

**Description**

This service gets the width of the widget.

**Parameters**

- **widget**
  Pointer to widget

- **return_width**
  Pointer to destination for widget width

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful widget width get</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

**Allowed From**

Initialization and threads

**Example**

```c
GX_VALUE my_widget_width;

/* Get width of widget "my_widget". */
status = gx_widget_width_get(&my_widget, &my_widget_width);

/* If status is GX_SUCCESS the width of widget "my_widget" is in "my_widget_width". */
```
See Also

gx_widget_attach, gx_widget_back_move, gx_widget_background_set,
gx_widget_border_draw, gx_widget_border_style_set,
gx_widget_border_width_get, gx_widget_canvas_get, gx_widget_child_detect,
gx_widget_children_draw, gx_widget_client_get, gx_widget_created,
gx_widget_created_test, gx_widget_delete, gx_widget_detach, gx_widget_draw,
gx_widget_draw_set, gx_widget_event_generate, gx_widget_event_process,
gx_widget_event_process_set, gx_widget_event_to_parent, gx_widget_find,
gx_widget_front_move, gx_widget_height_get, gx_widget_hide, gx_widget_resize,
gx_widget_shift, gx_widget_show, gx_widget_status_add, gx_widget_status_get,
gx_widget_status_remove, gx_widget_status_test, gx_widget_style_add,
gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set
**gx_window_client_height_get**

Get window client height

**Prototype**

```c
UINT gx_window_client_height_get(GX_WINDOW *window, 
                                GX_VALUE *return_height);
```

**Description**

This service gets the client height of the window.

**Parameters**

- `window` Pointer to window
- `return_height` Pointer to destination for client height

**Return Values**

- `GX_SUCCESS` (0x00) Successful window client height get
- `GX_PTR_ERROR` (0x07) Invalid pointer
- `GX_INVALID_WIDGET` (0x12) Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
GX_RECTANGLE my_client_height;

/* Get client height of “my_window”. */
status = gx_window_client_height_get(&my_window, 
                                  &my_client_height);

/* If status is GX_SUCCESS the window “my_window” client height is 
 contained in “my_client_height”. */
```

**See Also**

- `gx_window_canvas_set`, `gx_window_client_scroll`, `gx_window_client_width_get`,
- `gx_window_create`, `gx_window_draw`, `gx_window_event_process`,
- `gx_window_root_create`, `gx_window_root_delete`,
- `gx_window_root_event_process`, `gx_window_root_find`,
- `gx_window_scroll_info_get`, `gx_window_scrollbar_find`, `x_window_wallpaper_get`,
- `gx_window_wallpaper_set`
**gx_window_client_scroll**

Scroll window clients

**Prototype**

```
UINT gx_window_client_scroll(GX_WINDOW *window, GX_VALUE x_scroll,
                              GX_VALUE y_scroll);
```

**Description**

This service scrolls the window clients by the specified amount.

**Parameters**

- **window**: Pointer to window
- **x_scroll**: Amount to scroll on the x-axis
- **y_scroll**: Amount to scroll on the y-axis

**Return Values**

- **GX_SUCCESS** (0x00): Successful window client scroll
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_VALUE** (0x22): Scroll value(s) not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Scroll clients of “my_window”. */
status = gx_window_client_scroll(&my_window, 10, 0);

/* If status is GX_SUCCESS the clients of window “my_window” have been scrolled. */
```

**See Also**

`gx_window_canvas_set`, `gx_window_client_height_get`,
`gx_window_client_width_get`, `gx_window_create`, `gx_window_draw`,
`gx_window_event_process`, `gx_window_root_create`, `gx_window_root_delete`,
`gx_window_root_event_process`, `gx_window_root_find`,
`gx_window_scroll_info_get`, `gx_window_scrollbar_find`,
`gx_window_wallpaper_get`, `gx_window_wallpaper_set`
**gx_window_client_width_get**

Get window client width

**Prototype**

```c
UINT gx_window_client_width_get(GX_WINDOW *window,
                                GX_VALUE *return_width);
```

**Description**

This service gets the client width of the specified window.

**Parameters**

- `window`: Pointer to window
- `return_height`: Pointer to destination for client width

**Return Values**

- **GX_SUCCESS** (0x00): Successful window client width get
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
GX_VALUE my_client_width1

/* Get client width of "my_window". */
status = gx_window_client_width_get(&my_window, &my_client_width);

/* If status is GX_SUCCESS "my_client_width" contains the client width of window "my_window". */
```

**See Also**

- `gx_window_canvas_set`, `gx_window_client_height_get`, `gx_window_client_scroll`,
- `gx_window_create`, `gx_window_draw`, `gx_window_event_process`,
- `gx_window_root_create`, `gx_window_root_delete`,
- `gx_window_root_event_process`, `gx_window_root_find`,
- `gx_window_scroll_info_get`, `gx_window_scrollbar_find`,
- `gx_window_wallpaper_get`, `gx_window_wallpaper_set`
**gx_window_close**

Close modal window

**Prototype**

```c
UINT gx_window_close(GX_WINDOW *window);
```

**Description**

This service forces a modal window to detach from its parent and return from the modal execution loop.

**Parameters**

`window`  
Pointer to window control block

**Return Values**

- **GX_SUCCESS** (0x00)  
  Successfully closed window
- **GX_CALLER_ERROR** (0x11)  
  Invalid caller of this function
- **GX_PTR_ERROR** (0x07)  
  Invalid pointer
- **GX_INVALID_WIDGET** (0x12)  
  Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Close window "my_window". */
status = gx_window_close(&my_window);
/* If status is GX_SUCCESS window "my_window" has been closed. */
```

**See Also**

- `gx_window_canvas_set`, `gx_window_client_height_get`, `gx_window_client_scroll`,  
- `gx_window_client_width_get`, `gx_window_draw`, `gx_window_event_process`,  
- `gx_window_root_create`, `gx_window_root_delete`,  
- `gx_window_root_event_process`, `gx_window_root_find`,  
- `gx_window_scroll_info_get`, `gx_window_scrollbar_find`,  
- `gx_window_wallpaper_get`, `gx_window_wallpaper_set`
gx_window_create

Create window

Prototype

UINT gx_window_create(GX_WINDOW *window, GX_CONST GX_CHAR *name,
                        GX_WIDGET *parent, ULONG style,
                        USHORT window_id, GX_CONST GX_RECTANGLE
                        *size);

Description

This service creates a window.

GX_WINDOW is derived from GX_WIDGET and supports all
gx_widget API services.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>Pointer to window control block</td>
</tr>
<tr>
<td>name</td>
<td>Logical name of window</td>
</tr>
<tr>
<td>parent</td>
<td>Pointer to parent widget</td>
</tr>
<tr>
<td>style</td>
<td>Window style. <strong>Appendix D</strong> contains pre-defined general styles for all widgets as well as widget-specific styles.</td>
</tr>
<tr>
<td>window_id</td>
<td>Application-defined ID of the window</td>
</tr>
<tr>
<td>size</td>
<td>Size of the window</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00)</td>
<td>Successful window create</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED</td>
<td>(0x13)</td>
<td>Widget already created</td>
</tr>
<tr>
<td>GX_INVALID_SIZE</td>
<td>(0x19)</td>
<td>Invalid widget control block size</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12)</td>
<td>Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads
Example

GX_WINDOW my_window;

/* Create window "my_window". */
status = gx_window_create(&my_window, "my window",
                           &my_parent_window, GX_STYLE_BORDER_RAISED,
                           MY_WINDOW_ID, &size);

/* If status is GX_SUCCESS window "my_window" has been created. */

See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_draw, gx_window_event_process,
gx_window_root_create, gx_window_root_delete,
gx_window_root_event_process, gx_window_root_find,
gx_window_scroll_info_get, gx_window_scrollbar_find,
gx_window_wallpaper_get, gx_window_wallpaper_set
**gx_window_draw**

**Draw window**

**Prototype**

VOID gx_window_draw(GX_WINDOW *window);

**Description**

This service draws a window. This service is normally called internally during canvas refresh, but can also be called from custom window drawing functions.

**Parameters**

- **window**
  Pointer to window control block

**Return Values**

None

**Allowed From**

Threads

**Example**

/* Write a custom window draw function. */

VOID my_custom_window_draw(GX_WINDOW *window)
{
  /* Call default window draw. */
  gx_window_draw(window);
  /* Add your own drawing here. */
}

**See Also**

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll, gx_window_client_width_get, gx_window_create, gx_window_event_process, gx_window_root_create, gx_window_root_delete, gx_window_root_event_process, gx_window_root_find, gx_window_scroll_info_get, gx_window_scrollbar_find, gx_window_wallpaper_get, gx_window_wallpaper_set
gx_window_event_process

Process window event

Prototype

UINT gx_window_event_process(GX_WINDOW *window, GX_EVENT *event);

Description

This service processes an event for this window.

Parameters

<table>
<thead>
<tr>
<th>window</th>
<th>Pointer to window control block</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Pointer to event to process</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful window event processing</td>
</tr>
<tr>
<td>GX_CALLER_ERROR</td>
<td>(0x11) Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Threads
Example

/* Call generic window event processing as part of custom event processing function. */

UINT custom_window_event_process(GX_WINDOW *window,  
    GX_EVENT *event)
{
    UINT status = GX_SUCCESS;

    switch(event->gx_event_type)
    {
    case xyz:
        /* Insert custom event handling here */
        break;

    default:
        /* Pass all other events to the default window event processing */
        status = gx_window_event_process(window, event);
        break;
    }

    return status;
}

See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,  
gx_window_client_width_get, gx_window_create, gx_window_draw,  
gx_window_root_create, gx_window_root_delete,  
gx_window_root_event_process, gx_window_root_find,  
gx_window_scroll_info_get, gx_window_scrollbar_find,  
gx_window_wallpaper_get, gx_window_wallpaper_set
**gx_window_execute**

Modally execute a window

**Prototype**

```c
UINT gx_window_execute(GX_WINDOW *window,
                        ULONG *return_ptr)
```

**Description**

This service modally executes a window. Any user input (pen events, etc) outside of the window client area will be ignored. Note that this function enters a continuous blocking execution loop, and does not return to the caller until the model execution is terminated.

Modal execution terminates when the event processing for any received event returns a non-zero status value, or when the gx_window_close API function is invoked. The non-zero event processing return code is returned to the caller through the return_ptr passed into this API.

**Parameters**

- **window**: Pointer to window control block
- **return_ptr**: Location to save modal execution exit status. May be GX_NULL.

**Return Values**

- **GX_SUCCESS** (0x00): Successful execution
- **GX_SYSTEM_EVENT_RECEIVE_ERROR** (0x05): Pickup event from event queue fail
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

**Allowed From**

Initialization and threads

**Example**

```c
/* Execute a modal window. */
status = gx_window_execute(&my_window, &return_code);

/* If status is GX_SUCCESS the window was executed, and return_code holds the execution return code. */
```
See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_create, gx_window_draw,
gx_window_event_process, gx_window_root_delete,
gx_window_root_event_process, gx_window_root_find,
gx_window_scroll_info_get, gx_window_scrollbar_find,
gx_window_wallpaper_get, gx_window_wallpaper_set
gx_window_root_create

Create a root window

Prototype

UINT gx_window_root_create(GX_WINDOW_ROOT *root_window,
    GX_CONST GX_CHAR *name,
    GX_CANVAS *canvas, ULONG style,
    USHORT id,
    GX_CONST GX_RECTANGLE *size)

Description

This service creates a root window.

Parameters

root_window Pointer to root window control block

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS (0x00)</td>
<td>Successfully created root window</td>
</tr>
<tr>
<td>GX_CALLER_ERROR (0x11)</td>
<td>Invalid caller of this function</td>
</tr>
<tr>
<td>GX_PTR_ERROR (0x07)</td>
<td>Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_SIZE (0x19)</td>
<td>Invalid widget control block size</td>
</tr>
<tr>
<td>GX_ALREADY_CREATED (0x13)</td>
<td>Widget already created</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

GX_ROOT_WINDOW root_window;
GX_CANVAS canvas;

/* Create canvas here. */

/* Create a root window */
status = gx_window_root_create(&root_window, "root", &canvas,
    GX_STYLE_BORDER_NONE, GX_NULL, &size);

/* If status is GX_SUCCESS, the "root_window" is successfully created. */

See Also

gx_window_root_delete, gx_window_root_event_process, gx_window_root_find
**gx_window_root_delete**

Destroy a root window

**Prototype**

```c
UINT  gx_window_root_delete(GX_WINDOW_ROOT *root_window)
```

**Description**

This service deletes a root window.

**Parameters**

- `root_window` Pointer to root window control block

**Return Values**

- **GX_SUCCESS** (0x00) Successfully deleted root window
- **GX_CALLER_ERROR** (0x11) Invalid caller of this function
- **GX_PTR_ERROR** (0x07) Invalid pointer
- **GX_INVALID_WIDGET** (0x12) Widget not valid
- **GX_SYSTEM_MEMORY_ERROR** (0x30) Memory free function is not defined

**Allowed From**

- Initialization and threads

**Example**

```c
/* Delete a root window */
status = gx_window_root_delete(&root_window);
/* If status is GX_SUCCESS the "root_window" is destroyed. */
```

**See Also**

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_create, gx_window_draw,
gx_window_event_process, gx_window_root_create,
gx_window_root_event_process, gx_window_root_find,
gx_window_scroll_info_get, gx_window_scrollbar_find,
gx_window_wallpaper_get, gx_window_wallpaper_set
**gx_window_root_event_process**

Process event for the root window

**Prototype**

```c
UINT gx_window_root_create(GX_WINDOW_ROOT *root_window,
                           GX_EVENT *event)
```

**Description**

This service processes events for the specified root window.

**Parameters**

- `root_window`: Pointer to root window control block
- `event`: Pointer to the event to be processed

**Return Values**

- `GX_SUCCESS` (0x00): Successfully processed root window event
- `GX_CALLER_ERROR` (0x11): Invalid caller of this function
- `GX_PTR_ERROR` (0x07): Invalid pointer

**Allowed From**

Threads

**Example**

```c
/* Call generic root window event processing as part of custom event processing function. */

UINT custom_root_window_event_process(GX_ROOT_WINDOW *root,
                                       GX_EVENT *event)
{
    UINT status = GX_SUCCESS;

    switch(event->gx_event_type)
    {
      case xyz:
        /* Insert custom event handling here */
        break;

      default:
        /* Pass all other events to the default root window event processing */
        status = gx_window_root_event_process(root, event);
        break;
    }

    return status;
}
```
See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_create, gx_window_draw,
gx_window_event_process, gx_window_root_create, gx_window_root_delete,
gx_window_root_find, gx_window_scroll_info_get, gx_window_scrollbar_find,
gx_window_wallpaper_get, gx_window_wallpaper_set
gx_window_root_find

Find root window

Prototype

UINT  gx_window_root_find (GX_WIDGET *widget,
                           GX_WINDOW_ROOT **return_root_window);

Description

This service finds the root window for the specified widget.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widget</td>
<td>Pointer to widget control block</td>
</tr>
<tr>
<td>return_root_window</td>
<td>Pointer to destination for found root window</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_SUCCESS</td>
<td>(0x00) Successful root window find</td>
</tr>
<tr>
<td>GX_FAILURE</td>
<td>(0x00) Root window not exist</td>
</tr>
<tr>
<td>GX_PTR_ERROR</td>
<td>(0x07) Invalid pointer</td>
</tr>
<tr>
<td>GX_INVALID_WIDGET</td>
<td>(0x12) Widget not valid</td>
</tr>
</tbody>
</table>

Allowed From

Initialization and threads

Example

/* Find root window associated with window “my_window”. */
status = gx_window_root_find(&my_window, &root_window);

/* If status is GX_SUCCESS the “root_window” contains the root window for window “my_window”. */

See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll, 
gx_window_client_width_get, gx_window_create, gx_window_draw, 
gx_window_event_process, gx_window_root_create, gx_window_root_delete, 
gx_window_root_event_process, gx_window_scroll_info_get, 
gx_window_scrollbar_find, gx_window_wallpaper_get, gx_window_wallpaper_set
gx_window_scroll_info_get

Get window scroll info

Prototype

UINT gx_window_scroll_info_get(GX_WINDOW *window, ULONG style,
                         GX_SCROLL_INFO *return_scroll_info);

Description

This service gets the window scroll information.

Parameters

- window Pointer to window
- style GX_SCROLLER_HORIZOTIONAL or GX_SCROLLER_VERTICAL
- return_scroll_info Pointer to destination for scroll info. The parent window initializes this structure to inform the scrollbar of the parent window total size, viewable area, and scrolling increment and limits. The default implementation uses the windows client area as the viewable area and scrolls by pixels, but customized window implementation can utilize the scroll parameters. Appendix I contains the definition of the GX_SCROLL_INFO structure.

Return Values

- GX_SUCCESS (0x00) Successful window scroll info get
- GX_PTR_ERROR (0x07) Invalid pointer
- GX_INVALID_WIDGET (0x12) Widget not valid
- GX_INVALID_TYPE (0x1B) Invalid type

Allowed From

Initialization and threads
Example

GX_SCROLL_INFO scroll_info;

/* Get scroll information for window "my_window". */
status = gx_window_scroll_info_get(&my_window,
    GX_SCROLLBAR_HORIZONTAL, &scroll_info);

/* If status is GX_SUCCESS the "scroll_info" contains the scroll
information for window "my_window". */

See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_create, gx_window_draw,
gx_window_event_process, gx_window_root_create, gx_window_root_delete,
gx_window_root_event_process, gx_window_root_find,
gx_window_scrollbar_find, gx_window_wallpaper_get, gx_window_wallpaper_set
**gx_window_scrollbar_find**

This service finds the scrollbar for the specified window.

**Prototype**

```c
UINT gx_window_scrollbar_find(GX_WINDOW *window, USHORT type, 
                              GX_SCROLLBAR **return_scrollbar);
```

**Description**

This service finds the scrollbar for the specified window.

**Parameters**

- **window**: Pointer to window
- **type**: GX_TYPE_VERTICAL_SCROLL or GX_TYPE_HORIZONTAL_SCROLL
- **return_scrollbar**: Pointer to destination for scrollbar

**Return Values**

- **GX_SUCCESS** (0x00): Successful window scrollbar find
- **GX_NOT_FOUND** (0x09): Scrollbar not found
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid
- **GX_INVALID_TYPE** (0x1B): Invalid type

**Allowed From**

Initialization and threads

**Example**

```c
/* Find horizontal scrollbar for window "my_window". */
status = gx_window_scrollbar_find(&my_window, 
                                 GX_SCROLLBAR_HORIZONTAL, &my_scrollbar);

/* If status is GX_SUCCESS the "my_scrollbar" contains the 
horizontal scrollbar for window "my_window". */
```

**See Also**

- gx_window_canvas_set
- gx_window_client_height_get
- gx_window_client_scroll
- gx_window_client_width_get
- gx_window_create
- gx_window_draw
- gx_window_event_process
- gx_window_root_create
- gx_window_root_delete
- gx_window_root_event_process
- gx_window_root_find
- gx_window_scroll_info_get
- gx_window_wallpaper_get
- gx_window_wallpaper_set
**gx_window_wallpaper_get**

Get window wallpaper

### Prototype

```c
UINT gx_window_wallpaper_get(GX_WINDOW *window,
                              GX_RESOURCE_ID *return_wallpaper_id);
```

### Description

This service gets the wallpaper for the specified window.

### Parameters

- **window**: Pointer to window
- **return_wallpaper_id**: Pointer to destination for resource ID of wallpaper

### Return Values

- **GX_SUCCESS**: (0x00) Successful window wallpaper get
- **GX_PTR_ERROR**: (0x07) Invalid pointer
- **GX_INVALID_WIDGET**: (0x12) Widget not valid

### Allowed From

Initialization and threads

### Example

```c
GX_RESOURCE_ID my_window_wallpaper;

/* Get wallpaper for window "my_window". */
status = gx_window_wallpaper_get(&my_window, &my_window_wallpaper);

/* If status is GX_SUCCESS the "my_window_wallpaper" contains the wallpaper resource ID for window "my_window". */
```

### See Also

- `gx_window_canvas_set`, `gx_window_client_height_get`, `gx_window_client_scroll`, `gx_window_client_width_get`, `gx_window_create`, `gx_window_draw`, `gx_window_event_process`, `gx_window_root_create`, `gx_window_root_delete`, `gx_window_root_event_process`, `gx_window_root_find`, `gx_window_scroll_info_get`, `gx_window_scrollbar_find`, `gx_window_wallpaper_set`
gx_window_wallpaper_set

Set window wallpaper

Prototype

UINT gx_window_wallpaper_set(GX_WINDOW *window,
           GX_RESOURCE_ID wallpaper_id,
           GX_BOOL tile);

Description

This service sets the wallpaper for the specified window.

Parameters

- **window**: Pointer to window
- **wallpaper_id**: Resource ID of wallpaper to use
- **tile**: Wallpaper is tiled if GX_TRUE, otherwise wallpaper is not tiled

Return Values

- **GX_SUCCESS** (0x00): Successful window wallpaper set
- **GX_CALLER_ERROR** (0x11): Invalid caller of this function
- **GX_PTR_ERROR** (0x07): Invalid pointer
- **GX_INVALID_WIDGET** (0x12): Widget not valid

Allowed From

Initialization and threads

Example

/* Set wallpaper for window "my_window". */
status = gx_window_wallpaper_set(&my_window,
       MY_WALLPAPER_RESOURCE_ID,
       GX_TRUE);
/* If status is GX_SUCCESS the wallpaper for window "my_window" is set. */
See Also

gx_window_canvas_set, gx_window_client_height_get, gx_window_client_scroll,
gx_window_client_width_get, gx_window_create, gx_window_draw,
gx_window_event_process, gx_window_root_create, gx_window_root_delete,
gx_window_root_event_process, gx_window_root_find,
gx_window_scroll_info_get, gx_window_scrollbar_find, gx_window_wallpaper_get
Chapter 5: GUIX Display Drivers

GUIX Display drivers define the software interface between the abstract drawing canvas and the physical display hardware. The GUIX display driver implements the lowest-level drawing functions that actually change pixel color information in the canvas memory and transfer the canvas memory to the physical display frame buffer in double-buffered systems.

GUIX Display drivers are defined by a structure containing the physical display parameters and a set of function pointers to the low-level driver functions. By using these indirect function pointers, the abstract canvas and widget drawing functions are made completely independent of the hardware details.

GUIX provides a complete, fully functional, default set of drawing functions for each supported color depth and color format. When implementing a display driver with no specific hardware acceleration capability or other hardware specific considerations, these default drawing functions are normally sufficient for the final driver implementation. For these simplest of drivers, the only function that normally needs to be implemented in the driver software is a function to configure the hardware device. This often involves initializing various hardware registers to define the LCD display clock, display dimensions etc. For all other functions, the driver implementation simply initialize the GX_DISPLAY function pointers to the default function implementations for the desired color depth and format.

When implementing a custom display driver, the best practice is to first initialize your display driver drawing function pointers with the default software implementation for the color depth you want to support, then replace those function pointers where desired to call your custom function implementations (if any). To assist with this, there is a default setup function available for each supported color depth and format. For example, if you are writing a 16 bit 5:6:5 format RGB display driver, the first thing your custom driver would normally do is invoke the generic setup routine for this color depth:

```c
UINT my_custom_565_display_driver(GX_DISPLAY *display)
{
    // perform standard function pointer setup
    _gx_display_driver_565rgb_setup(display, GX_NULL, my_buffer_toggle);
}
```
The parameter my_buffer_toggle above is a pointer to your display driver buffer toggle function (which may be GX_NULL if your driver is single-buffered and drawing directly to the hardware frame buffer).

If you are writing a custom display driver, you will need to include the gx_display.h header file in your custom driver source, which is an internal use header file not available to application level software.

The GUIX display level drawing functions receive as input a pointer to a GX_DRAW_CONTEXT structure. The GX_DRAW_CONTEXT structure defines the clipping coordinates for the current drawing operation along with the brush and colors being used. Each drawing function receives as input additional parameters specific to the function requirements.

The signature of the GX_DISPLAY driver entry point is defined as

UINT <device>_graphics_driver_<format>(GX_DISPLAY *diplay)

While the name of this function is completely up to the implementor, the convention for the drivers provided with GUIX is to use a hardware specific device name in the <device> field and color format for <format> field above.

This function must initialize the GX_DISPLAY structure provided as input and perform any hardware setup that is required. The GX_DISPLAY structure contains the following fields:

ULONG gx_display_id- This is a field for use by the application, in cases where more than one instance of a particular driver is created.

CHAR  *gx_display_name- An optional name used to identify the driver.

GX_DISPLAY *gx_display_created_next: This field is initialized by GUIX, and is used to create and maintain a list of all GX_DISPLAY instances.

GX_DISPLAY *gx_display_created_previous: This field is initialized by GUIX, and is used to create and maintain a list of all GX_DISPLAY instances.

GX_VALUE gx_display_color_format: This field should reflect the graphics data format supported by this driver. The color format types are defined in the gx_api.h header file.

GX_VALUE gx_display_width: This field should be initialized to hold the physical display width, in pixels.
GX_VALUE gx_display_height: This field should be initialized to hold the physical display height, in pixels.

GX_COLOR *gx_display_color_table: This is a pointer to a table used to convert color Id values to color format specific color values.

GX_PIXELMAP *gx_display_pixelmap_table: This is a pointer to the active pixelmap table for this display.

GX_FONT *gx_display_font_table: This is a pointer to the active font table for this display.

GX_COLOR *gx_display_palette: For palette mode drivers, this is a pointer to the active color palette. For drivers that do not use a color palette, this pointer is GX_NULL.

UINT gx_display_color_table_size: Size of the active color table.

UINT gx_display_pixelmap_table_size: Number of entries in the active pixelmap table.

UINT gx_display_font_table_size: Number of entries in the active font table.

UINT gx_display_palette_size: Number of entries in color palette (if any).

ULONG gx_display_handle:

UINT gx_display_driver_ready: This field is use to signal to GUIX when the driver is ready for operation. In some cases, the driver may require several levels of initialization and configuration, during which time GUIX must not attempt to utilize the driver. This flag should be set to 1 when the driver is ready to service drawing requests.

VOID *gx_display_driver_data: This field is for use by the driver implementation. If the driver needs to create and reference additional information not available in the GX_DISPLAY structure, the driver should allocate space for and point to this additional data using this structure field. An example of driver-specific extra data might include the DMA channel being used by the driver or the SPI channel to which the display frame buffer is connected.

VOID (*gx_display_driver_drawing_initiate)(struct GX_DISPLAY_STRUCT *display, struct GX_CANVAS_STRUCT *canvas). This is a function pointer that, if not NULL, is invoked by the gx_canvas_drawing_initiate function. For display drivers that utilize a graphics accelerator or hardware graphics
display list, this function might be used to begin a new display list. This function pointer can be NULL.

VOID (*gx_display_driver_drawing_complete)(struct GX_DISPLAY_STRUCT *display, struct GX_CANVAS_STRUCT *canvas). This is a function pointer that, if not NULL, is invoked by the gx_canvas_drawing_complete function. For display drivers that utilize a graphics accelerator or hardware graphics display list, this function might be used to begin rendering the currently open display list. This function pointer can be NULL.

VOID (*gx_display_driver_palette_set)(struct GX_DISPLAY_STRUCT *display, GX_COLOR *palette, INT count): This is a pointer to a function to install a color palette. This function is NULL unless the driver operates in palette (also called color lookup table or CLUT) mode.

VOID (*gx_display_driver_simple_line_draw)(GX_DRAW_CONTECT *context, INT x1, INT y1, INT x2, INT y2): This is a pointer to a function to implement generic line drawing, no anti-aliasing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_simple_wide_line_draw)(GX_DRAW_CONTECT *context, INT x1, INT y1, INT x2, INT y2): This is a pointer to a function to implement generic wide line drawing, no anti-aliasing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_line_draw)(GX_DRAW_CONTECT *context, INT x1, INT y1, INT x2, INT y2): This is a pointer to a function to implement generic anti-aliased line drawing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_wide_line_draw)(GX_DRAW_CONTECT *context, INT x1, INT y1, INT x2, INT y2): This is a pointer to a function to implement generic anti-aliased wide line drawing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_horizontal_line_draw)(GX_DRAW_CONTECT *context, INT x1, INT x2, INT y): This is a pointer to a function to implement the special case of horizontal line drawing. Default implementations of this function are provided for each supported color depth and color format.
VOID (*gx_display_driver_horizontal_pixelmap_line_draw)(GX_DRAW_CONTEXT *context, INT x1, INT x2, INT y, GX_PIXELMAP *map): This is a pointer to a function to implement drawing a single pixelmap row. This function is used internally for pattern filling shapes. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_vertical_line_draw)(GX_DRAW_CONTEXT *context, INT y1, INT y2, INT x): This is a pointer to a function to implement the special case of horizontal line drawing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_horizontal_pattern_line_draw)(GX_DRAW_CONTEXT *context, INT x1, INT x2, INT y): This is a pointer to a function to implement horizontal pattern line drawing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_vertical_pattern_line_draw)(GX_DRAW_CONTEXT *context, INT y1, INT y2, INT x): This is a pointer to a function to implement vertical pattern line drawing. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_canvas_copy)(struct GX_CANVAS_STRUCT *source, struct GX_CANVAS_STRUCT *dest): This is a pointer to a function to copy canvas data from one canvas to another. The source canvas invalid rectangle is used to define the copy area. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_canvas_blend)(struct GX_CANVAS_STRUCT *source, struct GX_CANVAS_STRUCT *dest): This is a pointer to a function to alpha-blend canvas data from the source canvas with the existing data in the destination canvas. The source canvas invalid rectangle is used to define the blend area. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_pixelmap_blend)(GX_DRAW_CONTEXT *context, INT xpos, INT ypos, GX_PIXELMAP *pmp, GX_UBYTE alpha): This is a pointer to a function to blend a pixelmap on the background canvas defined by the draw context. The supplied alpha value may be in addition to an alpha channel contained in the pixelmap data. Default implementations of this function are provided for each supported color depth and color format.
VOID (*gx_display_driver_pixelmap_draw)(GX_DRAW_CONTEXT *context, INT xpos, INT ypos, GX_PIXELMAP *pmp): This is a pointer to a function to draw a pixelmap into the canvas defined by the draw context. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_jpeg_draw)(GX_DRAW_CONTEXT *context, INT xpos, INT ypos, GX_PIXELMAP *pmp): This is a pointer to a function to decode a jpg image and render it directly to the canvas. This function is only provided if GX_SOFTWARE_DECODER_SUPPORT is defined. This function pointer an be NULL. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_png_draw)(GX_DRAW_CONTEXT *context, INT xpos, INT ypos, GX_PIXELMAP *pmp): This is a pointer to a function to decode a png image and render it directly to the canvas. This function is only provided if GX_SOFTWARE_DECODER_SUPPORT is defined. This function pointer an be NULL. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_pixelmap_rotate)(GX_DRAW_CONTEXT *context, INT xpos, INT ypos, GX_PIXELMAP *pmp, INT angle, INT rot_cx, INT rot_cy): This is a pointer to a function to rotate a pixmap and render the result directly to the canvas. This function is invoked by the gx_canvas_pixelmap_rotate API. Default implementations of this function are provided for each supported color depth and color format.

VOID *gx_display_driver_pixel_write)(GX_DRAW_CONTEXT *context, INT x, INT y, GX_COLOR color): This is a pointer to a function to write one pixel into the canvas memory. Default implementations of this function are provided for each supported color depth and color format.

VOID *gx_display_driver_block_move)(GX_DRAW_CONTEXT *context, GX_RECTANGLE *block, INT xshift, INT yshift): This is a pointer to a function to move or shift a block of pixels within a canvas. This function is primarily used for rapidly scrolling a window contents. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_pixel_blend)(GX_DRAW_CONTEXT *context,
INT x, INT y, GX_COLOR color, GX_UBYTE alpha): This function is used to alpha-blend the incoming pixel color value with the existing color value in the canvas memory at position x,y. Default implementations of this function are provided for each supported color depth and color format.

GX_COLOR (*gx_display_driver_native_color_get)(GX_COLOR rawcolor): This function converts a color from the 32-bit A:R:G:B color format used internally by GUIX to the native color format of the canvas and display. Some loss of color information is expected for display drivers running at lower color depths. Default implementations of this function are provided for each supported color depth and color format.

USHORT (*gx_display_driver_row_pitch_get)(USHORT width): Returns the byte count or stride of one row of graphics data given the requested canvas width. This function is used to calculate the size of the memory area needed to create a canvas. The row pitch and width are not always the same due to hardware scan line alignment constraints. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_buffer_toggle)(struct GX_CANVAS_STRUCT *canvas, GX_RECTANGLE *dirty_area): This is a pointer to a function to toggle between the working and visible frame buffers for double-buffered memory systems. This function must first instruct the hardware to begin using the new frame buffer, then copy the modified portion of the new visible buffer to the companion buffer, to insure the two buffers stay in synch.

VOID (*gx_display_driver_polygon_draw)(GX_DRAW_CONTEXT *context, INT num_points, GX_POINT *vertices): Pointer to a function to draw a polygon. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_polygon_fill)(GX_DRAW_CONTEXT *context, INT num_points, GX_POINT *vertices): Pointer to a function to draw a filled polygon. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_circle_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw a circle.
Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_circle_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw an anti-aliased circle. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_wide_circle_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw a circle with a wide outline. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_wide_anti_aliased_circle_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw an anti-aliased circle with a wide outline. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_circle_fill)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw a filled circle. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_arc_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer to a function to draw an arc. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_arc_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer to a function to draw an anti-aliased arc. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_wide_arc_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer to a function to draw an arc with a wide outline. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_wide_arc_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer to a function to draw an anti-aliased arc with a wide outline. Default implementations of this function are provided for each supported color depth and color format.
VOID (*gx_display_driver_arc_fill)(GX_DRAW_CONTEXT *context, 
INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer 
to a function to draw a filled arc. Default implementations of this function 
are provided for each supported color depth and color format.

VOID (*gx_display_driver_pie_fill)(GX_DRAW_CONTEXT *context, 
INT xcenter, INT ycenter, UINT r, INT start_angle, INT end_angle): Pointer 
to a function to draw a filled pie. Default implementations of this function 
are provided for each supported color depth and color format.

VOID (*gx_display_driver_ellipse_draw)(GX_DRAW_CONTEXT *context, 
INT xcenter, INT ycenter, INT a, INT b): Pointer to a function to draw an 
ellipse. Default implementations of this function are provided for each 
supported color depth and color format.

VOID(*gx_display_driver_anti_aliased_ellipse_draw)(GX_DRAW_CONTE 
XT *context, INT xcenter, INT ycenter, INT a, INT b): Pointer to a function 
to draw an ellipse. Default implementations of this function are provided for 
each supported color depth and color format.

VOID (*gx_display_driver_anti_aliased_anti_aliased_ellipse_draw)(GX_DRAW_CO 
NTXT *context, INT xcenter, INT ycenter, INT a, INT b): Pointer to a function 
to draw an ellipse with a wide outline. Default implementations of this function 
are provided for each supported color depth and color format.

VOID (*gx_display_driver_ellipse_fill)(GX_DRAW_CONTEXT *context, 
INT xcenter, INT ycenter, INT a, INT b): Pointer to a function to draw a 
filled ellipse. Default implementations of this function are provided for each 
supported color depth and color format.

VOID (*gx_display_driver_8bit_glyph_draw)(GX_DRAW_CONTEXT 
*context, GX_RECTANGLE *draw_area, VX_POINT *map_offset, VX 
GLYPH *glyph): Pointer to function to draw one 8-bit aliased text glyph
to the canvas using the brush of the current drawing context. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_4bit_glyph_draw)(GX_DRAW_CONTEXT *context, GX_RECTANGLE *draw_area, GX_POINT *map_offset, const GX_GLYPH *glyph): Pointer to function to draw one 4-bit aliased text glyph to the canvas using the brush of the current drawing context. Default implementations of this function are provided for each supported color depth and color format.

VOID (*gx_display_driver_1bit_glyph_draw)(GX_DRAW_CONTEXT *context, GX_RECTANGLE *draw_area, GX_POINT *map_offset, const GX_GLYPH *glyph): Pointer to function to draw one 1-bit monochrome text glyph to the canvas using the brush of the current drawing context. Default implementations of this function are provided for each supported color depth and color format.
GUIX Example

The GUIX demonstration system is delivered with a small example, defined in examples/helloworld/helloworld.c. This example illustrates the steps needed to take to initialize the GUIX system, to set up display drivers. The source code is listed on the following pages.
/* This is a small demonstration of the high-performance GUIX embedded UI run-time environment. This demonstration consists of a simple "Hello World" prompt on top of the root window. */

/* Include necessary system files. */
#include "tx_api.h"
#include "gx_api.h"

/* Define constants for the GUIX Win32 demo. */
/* Define the display dimensions specific to this implementation. */
#define DEMO_DISPLAY_WIDTH 320
#define DEMO_DISPLAY_HEIGHT 240

/* Define the number of pixels on the canvas */
#define DEFAULT_CANVAS_PIXELS (DEMO_DISPLAY_WIDTH * DEMO_DISPLAY_HEIGHT)

/* Define the ThreadX demo thread control block. */
TX_THREAD demo_thread;

/* Define the stack for the demo thread. */
ULONG demo_thread_stack[4096 / sizeof(ULONG)];

/* Define the GUIX resources for this demo. */
/* GUIX display represents the physical display device */
GX_DISPLAY demo_display;

/* GUIX canvas is the frame buffer on top of GUIX display. */
GX_CANVAS default_canvas;

/* The root window is a special GUIX background window, right on top of the canvas. */
GX_WINDOW_ROOT demo_root_window;

/* GUIX Prompt displays a string. */
GX_PROMPT demo_prompt;

/* Memory for the frame buffer. */
GX_COLOR default_canvas_memory[DEFAULT_CANVAS_PIXELS];

/* Define GUIX strings ID for the demo. */
enum demo_string_ids
{
    SID_HELLO_WORLD = 1,
    SID_MAX
};

/* Define GUIX string for the demo. */
CHAR *demo_strings[3] = { NULL, "Hello World" };

/* User-defined color ID */
#define GX_COLOR_ID_BLACK    GX_FIRST_USER_COLOR
#define GX_COLOR_ID_WHITE    (GX_FIRST_USER_COLOR + 1)

/* User-defined color table. */
static GX_COLOR demo_color_table[] =
{
    /* First, bring in GUIX default color table. These colors are used by GUIX internals. */
    GX_SYSTEM_DEFAULT_COLORS_DECLARE,

    /* In this demo, two color entries are added to the color table. */
    GX_COLOR_BLACK,
    GX_COLOR_WHITE
};
/* Define prototypes. */

VOID demo_thread_entry(ULONG thread_input);

int main(void)
{
/* Enter ThreadX. */
    tx_kernel_enter();

    return (0);
}

VOID tx_application_define(void *first_unused_memory)
{
/* Create the main demo thread. */
    tx_thread_create(&demo_thread, "GUIX Demo Thread", demo_thread_entry, 0,
        demo_thread_stack, sizeof(demo_thread_stack),
        1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
}

VOID demo_thread_entry(ULONG thread_input)
{
    GX_RECTANGLE root_window_size;
    GX_RECTANGLE prompt_position;

    /* Initialize GUIX. */
    gx_system_initialize();

    /* Install the demo string table. */
    gx_system_string_table_set(demo_strings, SID_MAX);

    /* Install the demo color table. */
    gx_system_color_table_set(demo_color_table, sizeof(demo_color_table) /
        sizeof(GX_COLOR));

    /* Create the demo display and associated driver. */
    gx_display_create(&demo_display, "demo display",
        win32_graphics_driver_setup_16bpp,
        DEMO_DISPLAY_WIDTH, DEMO_DISPLAY_HEIGHT);

    /* Create the default canvas. */
    gx_canvas_create(&default_canvas, "demo canvas", &demo_display,
        GX_CANVAS_MANAGED | GX_CANVAS_VISIBLE,
        DEMO_DISPLAY_WIDTH, DEMO_DISPLAY_HEIGHT,
        default_canvas_memory, sizeof(default_canvas_memory));

    /* Define the size of the root window. */
    gx_utility_rectangle_define(&root_window_size, 0, 0,
        DEMO_DISPLAY_WIDTH - 1, DEMO_DISPLAY_HEIGHT - 1);

    /* Create a background root window and attach to the canvas. */
    gx_window_root_create(&demo_root_window, "demo root window", &default_canvas,
        GX_STYLE_BORDER_NONE, GX_ID_NONE, &root_window_size);

    /* Set the root window to be black. */
    gx_widget_background_set(&demo_root_window, GX_COLOR_ID_BLACK,
        GX_COLOR_ID_BLACK);

    /* Create a text prompt on the root window. Set the text color to white,
      and the background to black. */

}
/* Define the size and the position of the prompt. */
gx_utility_rectangle_define(&prompt_position, 100, 90, 220, 130);

/* Create the prompt on top of the root window using the string defined by
 string ID SID_HELLO_WORLD. */
gx_prompt_create(&demo_prompt, NULL, &demo_root_window, SID_HELLO_WORLD,
                 GX_STYLE_NONE, GX_ID_NONE, &prompt_position);

/* Set the text color to be white, and the background color to be black. */
gx_prompt_text_color_set(&demo_prompt, GX_COLOR_ID_WHITE, GX_COLOR_ID_WHITE);
gx_widget_background_set(&demo_prompt, GX_COLOR_ID_BLACK, GX_COLOR_ID_BLACK);

/* Show the root window. */
gx_widget_show(&demo_root_window);

/* let GUIX run! */
gx_system_start();
# Appendix A: GUIX Color Definitions

## Pre-defined color values

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_COLOR_BLACK</td>
<td>0xff000000</td>
</tr>
<tr>
<td>GX_COLOR_RED</td>
<td>0xffb80000</td>
</tr>
<tr>
<td>GX_COLOR_GREEN</td>
<td>0xff00b800</td>
</tr>
<tr>
<td>GX_COLOR_BROWN</td>
<td>0xffb8bc00</td>
</tr>
<tr>
<td>GX_COLOR_BLUE</td>
<td>0xff0000b8</td>
</tr>
<tr>
<td>GX_COLOR_MAGENTA</td>
<td>0xffb800b8</td>
</tr>
<tr>
<td>GX_COLOR_CYAN</td>
<td>0xff00bcb8</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTGRAY</td>
<td>0xffc0c0c0</td>
</tr>
<tr>
<td>GX_COLOR_DARKGRAY</td>
<td>0xff808080</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTRED</td>
<td>0xff000000</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTGREEN</td>
<td>0xff00ff00</td>
</tr>
<tr>
<td>GX_COLOR_YELLOW</td>
<td>0xfffffff0</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTBLUE</td>
<td>0xff0000ff</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTMAGENTA</td>
<td>0xffff00ff</td>
</tr>
<tr>
<td>GX_COLOR_LIGHTCYAN</td>
<td>0xffff00ff</td>
</tr>
<tr>
<td>GX_COLOR_WHITE</td>
<td>0xffffffff</td>
</tr>
</tbody>
</table>
Pre-defined color IDs

GX_COLOR_ID_CANVAS 0
GX_COLOR_ID_WIDGET_FILL 1
GX_COLOR_ID_WINDOW_FILL 2
GX_COLOR_ID_DEFAULT_BORDER 3
GX_COLOR_ID_WINDOW_BORDER 4
GX_COLOR_ID_TEXT 5
GX_COLOR_ID_SELECTED_TEXT 6
GX_COLOR_ID_SELECTED_FILL 7
GX_COLOR_ID_SHADOW 8
GX_COLOR_ID_SHINE 9
GX_COLOR_ID_BUTTON_BORDER 10
GX_COLOR_ID_BUTTON_UPPER 11
GX_COLOR_ID_BUTTON_LOWER 12
GX_COLOR_ID_BUTTON_TEXT 13
GX_COLOR_ID_SCROLL_FILL 14
GX_COLOR_ID_SCROLL_BUTTON 15
GX_COLOR_ID_TEXT_INPUT_TEXT 16
GX_COLOR_ID_TEXT_INPUT_FILL 17
GX_COLOR_ID_SLIDER_TICK 18
GX_COLOR_ID_SLIDER_GROOVE_TOP 19
GX_COLOR_ID_SLIDER_GROOVE_BOTTOM 20
GX_COLOR_ID_SLIDER_NEEDLE_OUTLINE 21
GX_COLOR_ID_SLIDER_NEEDLE_FILL 22
GX_COLOR_ID_SLIDER_NEEDLE_LINE1 23
GX_COLOR_ID_SLIDER_NEEDLE_LINE2 24
GX_COLOR_ID_DISABLED_TEXT 25
GX_COLOR_ID_DISABLED_FILL 26
GX_COLOR_ID_READONLY_TEXT 27
GX_COLOR_ID_READONLY_FILL 28
## Appendix B: GUIX Color Formats

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX_COLOR_FORMAT_MONOCHROME</td>
<td>1</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_MONOCHROME_INVERTED</td>
<td>2</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_2BIT_4GRAY</td>
<td>3</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_2BIT_GRAY_INVERTED</td>
<td>4</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_4BIT_GRAY</td>
<td>5</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_4BIT_GRAY_INVERTED</td>
<td>6</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_4BIT_VGA</td>
<td>7</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_8BIT_GRAY</td>
<td>8</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_8BIT_GRAY_INVERTED</td>
<td>9</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_8BIT_PALETTE</td>
<td>10</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_8BIT_PACKED PIXEL</td>
<td>11</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_15BIT_BGR</td>
<td>12</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_15BIT_RGB</td>
<td>13</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_16BIT_RGB</td>
<td>14</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_16BIT_ARGB</td>
<td>15</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_16BIT_BGRA</td>
<td>16</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_16BIT_BGR</td>
<td>17</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_24BIT_RGB</td>
<td>18</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_24BIT_BGR</td>
<td>19</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_24BIT_XRGB</td>
<td>20</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_24BIT_BGRX</td>
<td>21</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_32BIT_ARGB</td>
<td>22</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_32BIT_RGBA</td>
<td>23</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_32BIT_ABGR</td>
<td>24</td>
</tr>
<tr>
<td>GX_COLOR_FORMAT_32BIT_BGRA</td>
<td>25</td>
</tr>
</tbody>
</table>
Appendix C: GUIX Widget Styles

General Styles (Used with most widget types):

**GX_STYLE_BORDER_NONE**
- Value: 0x00000000
- Description: Use this style to draw a widget with no border.

**GX_STYLE_BORDER_RAISED**
- Value: 0x00000001
- Description: Draw widget with a raised border.

**GX_STYLE_BORDER_RECESSED**
- Value: 0x00000002
- Description: Draw widget with a recessed border.

**GX_STYLE_BORDER_THIN**
- Value: 0x00000004
- Description: Draw a one-pixel width border.

**GX_STYLE_BORDER_THICK**
- Value: 0x00000008
- Description: Draw widget with a thick border.

**GX_STYLE_BORDER_MASK**
- Value: 0x0000000f
- Description: Mask value used to test only the style fields of the widget style member.

**GX_STYLE_TRANSPARENT**
- Value: 0x10000000
- Description: Create a widget that is at least partially transparent. This style should be used when a widget does not draw itself fully opaque, including widgets that draw a semi-transparent pixelmap as the widget background. This style flag informs GUIX that the widget parent must be drawn to refresh the widget background area.

**GX_STYLE_DRAW_SELECTED**
- Value: 0x20000000
- Description: Specify that the widget should be drawn using selected state colors and fonts. Different widget types use the DRAW_SELECTED style in different ways to indicate the widget is currently selected.
**GX_STYLE_DYNAMICALLY_ALLOCATED**
- Value: 0x80000000
- Description: Indicates the widget control block memory is dynamically allocated using the gx_system_memory_allocator service when the widget is created, and the control block memory is freed if the widget is destroyed.

**GX_STYLE_USE_LOCAL_ALPHA**
- Value: 0x01000000
- Description: Instructs GUIX drawing functions to use the local widget alpha value when drawing the widget. This flag is normally used by the internal GUIX logic to implement widget fading animations.

**GX_STYLE_ENABLED**
- Value: 0x40000000
- Description: Mark the widget as enabled, which allows the widget to accept user input events and generate output signals.

**Text Alignment Styles (styles applied to all widgets that draw text):**

**GX_STYLE_TEXT_LEFT**
- Value: 0x00001000
- Description: Text is drawn left-aligned within the widget client area.

**GX_STYLE_TEXT_RIGHT**
- Value: 0x00002000
- Description: Text is drawn right-aligned within the widget client area.

**GX_STYLE_TEXT_CENTER**
- Value: 0x00004000
- Description: Text is drawn center-aligned within the widget client area.

**GX_STYLE_TEXT_COPY**
- Value: 0x00008000
- Description: By default, widgets that draw text keep only a pointer to the text which is passed in by the application. For statically defined text that is defined within the string table, there is no reason for the widget to make a private copy of the text assigned. However, if the text assigned to a widget is created dynamically using functions like sprint() or gx_utility_ltoa, then it is often convenient to tell the widget to keep its own private copy of any text assigned. This allows the application to use automatic or temporary variables when defining the text string, when the application would otherwise be required to define statically defined character arrays for each text widget that is using dynamically defined text.
When this style flag is set, the widget will use the
`gx_system_memory_allocator` function to dynamically allocate the
memory block needed to hold a private copy of the assigned string.
Therefore using this style flag is predicated on the application
defining `memory_allocator` and `memory_deallocator` functions.

`GX_STYLE_TEXT_COPY` should not be cleared after it has been
set, and doing so will cause unpredictable results.

**Button Styles (apply only to GUIX button widget types):**

**GX_STYLE_BUTTON_PUSHED**
- Value: 0x00000010
- Description: Indicates the button is in the pushed or selected state.

**GX_STYLE_BUTTON_TOGGLE**
- Value: 0x00000020
- Description: Button will switch status between pushed and unpushed
  on every click event. This style is commonly used with “checkbox”
  style buttons.

**GX_STYLE_BUTTON_RADIO**
- Value: 0x00000040
- Description: This style indicates the button will be exclusive, and
deselect any button siblings when selected. This style is commonly
  used with “radio button” style buttons.

**GX_STYLE_BUTTON_EVENT_ON_PUSH**
- Value: 0x00000080
- Description: Indicates the button generates a click event when
  initially pushed. The default operation is to generate a click event
  when the button is released.

**GX_STYLE_BUTTON_REPEAT**
- Value: 0x00000100
- Description: Indicates the button should send repeated click events
to the button parent when the button is held in the pushed state.

**List Styles (apply only to GUIX list widget types):**

**GX_STYLE_CENTER_SELECTED**
- Value: 0x00000010
- Description: Reserved

**GX_STYLE_WRAP**
- Value 0x00000020
- Description: The list children wrap from start to end when the list is dragged or scrolled past the starting or ending list index.

**GX_STYLE_FLICKABLE**
- Value: 0x00000040
- Description: Reserved

**Pixelmap Button and Icon Button Styles:**

**GX_STYLE_HALIGN_CENTER**
- Value: 0x00010000
- Description: The button pixelmap should be center aligned within the button boundary on the horizontal axis.

**GX_STYLE_HALIGN_LEFT**
- Value: 0x00020000
- Description: The button pixelmap should be left aligned within the button boundary on the horizontal axis.

**GX_STYLE_HALIGN_RIGHT**
- Value: 0x00040000
- Description: The button pixelmap should be right aligned within the button boundary on the horizontal axis.

**GX_STYLE_VALIGN_CENTER**
- Value: 0x00080000
- Description: The button pixelmap should be center aligned within the button boundary on the vertical axis.

**GX_STYLE_VALIGN_TOP**
- Value: 0x00100000
- Description: The button pixelmap should be top aligned within the button boundary on the vertical axis.

**GX_STYLE_VALIGN_BOTTOM**
- Value: 0x00200000
- Description: The button pixelmap should be bottom aligned within the button boundary on the vertical horizontal axis.

**Slider Styles (Appy only to GX_SLIDER and derived widget types):**

**GX_STYLE_SHOW_NEEDLE**
- Value: 0x00000200
- Description: This style must be included for the slider to draw the needle indicator. This style can be disabled if the application wants to disable the slider needle or draw a custom needle indicator.

**GX_STYLE_SHOW_TICKMARKS**
- Value: 0x00000400
- Description: The slider widget will do software drawing of dashed tickmark lines when this style is enabled.

**GX_STYLE_SLIDER_VERTICAL**
- Value 0x00000800
- Description: Set this style flag to create a vertical slider, and clear this style flag to create a horizontal slider.

**Sprite Styles (Apply only to GX_SPRITE widget types):**

**GX_STYLE_SPRITE_AUTO**
- Value: 0x00000010
- Description: Indicates the sprite animation will run automatically when the sprite widget received the GX_EVENT_SHOW event.

**GX_STYLE_SPRITE_LOOP**
- Value: 0x00000020
- Description: With this style, the sprite widget will continuously loop through sprite animation frames until the sprite is stopped by the application.

**Pixelmap Slider Styles:**

**GX_STYLE_TILE_BACKGROUND**
- Value 0x00001000
- Description: The slider background image is tiled to fill the sprite bounding rectangle. This allows a small vertical or horizontal stripe image to be used to fill the slider background.

**Additional Progress Bar Styles:**

**GX_STYLE_PROGRESS_PERCENT**
- Value: 0x00000010
- Description: When this style is set, the progress bar will draw the bar value as a percentage rather than a raw value. The text is centered in the progress bar bounding rectangle.

**GX_STYLE_PROGRESS_TEXT_DRAW**
- Value: 0x00000020
- **Description:** Draw the current progress bar value as decimal text centered within the progress bar.

**GX_STYLE_PROGRESS_VERTICAL**
- **Value:** 0x0000040
- **Description:** Indicate the progress is vertically oriented. The default is horizontal orientation.

**GX_STYLE_PROGRESS_SEGMENT_FILL**:
- **Value:** 0x00000100
- **Description:** The progress bar value is indicated with segmented filled rectangles, rather than a solid fill.

**Additional Radial Progress Bar Styles:**

**GX_STYLE_RADIAL_PROGRESS_ALIAS**
- **Value:** 0x00000200
- **Description:** Draw the radial progress bar using anti-aliased brush styles. This requires more CPU bandwidth but also produces a nicer appearance. For lower performance CPU targets, clearing this style flag will result in faster drawing speed.

**GX_STYLE_RADIAL_PROGRESS_ROUND**
- **Value:** 0x00000400
- **Description:** Use a round line end brush style when drawing the radial progress bar arc. The default is a square line end.

**Additional Text Input Styles:**

**GX_STYLE_CURSOR_BLINK**
- **Value:** 0x00000040
- **Description:** The text input widget cursor will flash on and off rather than being steady.

**GX_STYLE_CURSOR_ALWAYS**
- **Value:** 0x00000080
- **Description:** The text input widget cursor is normally only displayed when the widget owns input focus. This style flag will make the cursor always visible regardless of input focus.

**GX_STYLE_TEXT_INPUT_NOTIFY_ALL**
- **Value:** 0x00000100
- **Description:** With this style flag set the GX_EVENT_TEXT_EDITED event every time key down event is received by the text input widget.
Additional Window Styles:

GX_STYLE_TILE_WALLPAPER
- Value: 0x00040000
- Description: The window will tile any assigned wallpaper image to fill the window client rectangle.

GX_STYLE_AUTO_HSCROLL
- Value: 0x00100000
- Description: Reserved for future use.

GX_STYLE_AUTO_VSCROLL
- Value: 0x00200000
- Description: Reserved for future use.

Additional Menu Styles:

GX_STYLE_MENU_EXPANDED
- Value: 0x00000010
- Description: Accordion menu widget is initially in expanded state.

Additional Tree View Styles:

GX_STYLE_TREE_VIEW_SHOW_ROOT_LINES
- Value: 0x00000010
- Description: Tree view widget should draw lines from node icon to root tree node.

Additional Scrollbar Styles:

GX_SCROLLBAR_BACKGROUND_TILE
- Value: 0x00010000
- Description: Reserved for future use.

GX_SCROLLBAR_RELATIVE_THUMB
- Value: 0x00020000
- Description: The scrollbar thumb width (for a horizontal scroll bar) or height (for a vertical scroll bar) are calculated based on the ratio of the visible area of the parent window to the min and max scrollbar range.

GX_SCROLLBAR_END_BUTTONS
- Value: 0x00040000
- Description: The scrollbar automatically creates and attaches buttons at each end of the scrollbar region.
**GX_SCROLLBAR_VERTICAL**
- Value: 0x01000000
- Description: The scrollbar is vertically oriented.

**GX_SCROLLBAR_HORIZONTAL**
- Value: 0x02000000
- Description: The scrollbar is horizontally oriented.

**Text Scroll Wheel Styles:**

**GX_STYLE_TEXT_SCROLL_WHEEL_ROUND**
- Value: 0x00000200
- Description: The scroll wheel uses a Sinusoidal algorithm to make the scroll wheel appear to have a rounded shape. This style flag can add significant overhead to the performance of the scroll wheel widget, but can also give the wheel a 3D realistic appearance.
Appendix D: GUIX Brush, Canvas and Gradient Attributes

Brush Styles:

GX_BRUSH_OUTLINE
- Value: 0x0000
- Description: This brush style applies to shape drawing functions such as gx_canvas_rectangle_draw or gx_canvas_polygon_draw. This style indicates the shape should be outlined, in addition to optionally being fill. If the GX_BRUSH_OUTLINE style is set and the GX_BRUSH_SOLID_FILL is cleared, the shape is only outlined.

GX_BRUSH_SOLID_FILL
- Value: 0x0001
- Description: This brush style applies to shape drawing functions, and indicates that the requested shape should be filled with a solid color using the current brush fill color.

GX_BRUSH_PIXELMAP_FILL
- Value: 0x0002
- Description: This brush style applies to shape drawing functions, and indicates that the requested shape should be pattern filled with the current brush pixelmap.

GX_BRUSH_ALIAS
- Value: 0x0004
- Description: This brush style applies to all line drawing and shape outlines. If this flag is set, lines and outlines are drawing with the more accurate but also more time consuming anti-aliased drawing algorithms. This style flag is only used for 16-bpp color depths and higher.

GX_BRUSH_UNDERLINE
- Value: 0x0008
- Description: This flag applies to text drawing, and indicates that subsequent text drawn should be underlined.

GX_BRUSH_ROUND
- Value: 0x0010
- Description: This flag applies to line drawing, and indicates that line ends are drawn with a round or circular shape, rather than the default square shape.
Canvas Flags:

GX_CANVAS_SIMPLE
- Value: 0x01
- Description: A memory canvas which is used to off-screen drawing.

GX_CANVAS_MANAGED
- Value: 0x02
- Description: A canvas which automatically flushed to the active display, either as part of the composite building process or as part of the buffer toggle operation for single-canvas architectures.

GX_CANVAS_VISIBLE
- Value: 0x04
- Description: This flag can be used to turn on and off a canvas, without losing the canvas drawing contents.

GX_CANVAS_MODIFIED
- Value: 0x08
- Description: Reserved for future use.

GX_CANVAS_COMPOSITE
- Value: 0x20
- Description: This flag is used by the application when configuring a multiple-canvas system which will composite multiple managed canvases into the composite canvas, and the composite is the driven to the hardware frame buffer.

Gradient Types:

GX_GRADIENT_TYPE_VERTICAL
- Value: 0x01
- Description: Creates a vertical alphamap gradient.

GX_GRADIENT_TYPE_ALPHA
- Value: 0x02
- Description: Creates an alpha-map style gradient. This is currently the only gradient style supported.

GX_GRADIENT_TYPE_MIRROR
- Value: 0x04
- Description: This flag indicates that the gradient should peak at the center of the width/height range, and return to the starting value as it reaches the right/bottom edge. Without this style flag, the gradient will be a linear gradient from top-to-bottom or left-to-right, depending on the GX_GRADIENT_TYPE_VERTICAL flag.
Appendix E: GUIX Event Description

**GX_EVENT_TERMINATE**
- **Description:** This event can be sent by the application to intentionally terminate the GUIX execution thread. This event will also cause a modally executing window to terminate modal execution and return GX_EVENT_TERMINATE. This event is used internally by the GUIX Win32 binding to terminate the GUIX application when the desktop window is closed.
  - **Payload:** None

**GX_EVENT_REDRAW**
- **Description:** This event can be generated to force GUIX to redraw every root window (and all child windows/widgets). This event marks every root window as dirty, forcing a complete system redraw when the next canvas refresh operation occurs. This event is also used internally for desktop operation to force a GUIX canvas refresh when the desktop operating system requests a re-draw.
  - **Payload:** None

**GX_EVENT_SHOW**
- **Description:** This event is internally generated whenever a widget is made visible, either by being attached to a visible widget or by invocation of the gx_widget_show() API. The event is received before the widget is drawn.
  - **Payload:** None

**GX_EVENT_HIDE**
- **Description:** This event is internally generated whenever a widget is made hidden either by being detached from its parent or through invocation of the gx_widget_hide() API. The event is received before the widget is made hidden.
  - **Payload:** None

**GX_EVENT_RESIZE**
- **Description:** This event is generated when a widget is resized via the gx_widget_resize API. The event is only generated if the widget gx_widget_status member includes GX_STATUS_RESIZE_NOTIFY.
  - **Payload:** None

**GX_EVENT_SLIDE**
- **Description:** Reserved for future use.
  - **Payload:** None

**GX_EVENT_FOCUS_GAINED**
- **Description:** This event is internally generated when a widget receives input focus.
  - **Payload:** None

**GX_EVENT_FOCUS_LOST**
- **Description:** This event is internally generated when a widget loses input focus.
  - **Payload:** None
GX_EVENT_HORIZONTAL_SCROLL
- Description: This event is generated by a horizontal scrollbar to inform the parent window of a scrolling operation. The event can also be generated by the application to force a window to scroll its child widgets.
- Payload: gx_event_intdata[0] contains the current scrollbar value.
  gx_event_intdata[1] contains the previous scrollbar value.

GX_EVENT_VERTICAL_SCROLL
- Description: This event is generated by a vertical scrollbar to inform the parent window of a scrolling operation. The event can also be generated by the application to force a window to scroll its child widgets.
- Payload: gx_event_intdata[0] contains the current scrollbar value.
  gx_event_intdata[1] contains the previous scrollbar value.

GX_EVENT_TIMER
- Description: This event is sent to a timer owner to notify the owner of timer expiration.
- Payload: gx_event_timer_id contains the user-assigned timer id.
  gx_event_target contains a pointer to the timer owner.

GX_EVENT_PEN_DOWN
- Description: This event is generated by touch screen and mouse input drivers to indicate user pen-down (or left mouse button click) event.
- Payload: gx_event_pointdata.gx_point_x = pen x position in pixels
  gx_event_pointdata.gx_point_y = pen y position in pixels
  gx_event_display_handle = handle of the target display

GX_EVENT_PEN_UP
- Description: This event is generated by touch screen and mouse input drivers to indicate user pen-up (or left mouse button released) event.
- Payload: gx_event_pointdata.gx_point_x = pen x position in pixels
  gx_event_pointdata.gx_point_y = pen y position in pixels
  gx_event_display_handle = handle of the target display

GX_EVENT_PEN_MOVE
- Description: This event is generated by mouse input driver to indicate the mouse has been moved to a new location, but no buttons are pressed.
- Payload: gx_event_pointdata.gx_point_x = pen x position in pixels
  gx_event_pointdata.gx_point_y = pen y position in pixels
  gx_event_display_handle = handle of the target display

GX_EVENT_PEN_DRAG
- Description: This event is generated by mouse and touch input drivers to indicate the pen is being dragged across the screen, or the mouse is being moved while the left mouse button is pressed.
- Payload: gx_event_pointdata.gx_point_x = pen x position in pixels
  gx_event_pointdata.gx_point_y = pen y position in pixels
  gx_event_display_handle = handle of the target display

GX_EVENT_KEY_DOWN:
- Description: This event is generated by keyboard input drivers to indicate a keyboard key has been pressed.
- Payload: gx_event_ushortdata[0] holds the Unicode key value.
GX_EVENT_KEY_UP
- Description: This event is generated by keyboard input drivers to indicate a keyboard key has been released.
- Payload: gx_event_ushortdata[0] holds the Unicode key value.

GX_EVENT_CLOSE
- Description: This event can be sent to any GX_WINDOW derived widget to cause that window to detach from it's parent (i.e. become hidden). If the window is executing modally, it will exit the modal execution loop and return GX_EVENT_CLOSE.
- Payload: None.

GX_EVENT_DELETE
- Description: This event is sent to any widget when the _gx_widget_delete API is used. This event informs the widget that it is about to be deleted, allowing the widget to do an necessary cleanup or memory release.
- Payload: gx_event_target points to the widget being deleted.

GX_EVENT_SLIDER_VALUE
- Description: This is a GX_SIGNAL event type generated by GX_SLIDER based child controls. It informs the slider parent that the slider has been manipulated by the user.
- Payload: gx_event_longdata holds the new slider value.
  gx_event_sender holds the ID of the slider widget.

GX_EVENT_TOGGLE_ON
- Description: This is a GX_SIGNAL event type generated by checkbox style (i.e. toggle style) GX_BUTTON widgets. It informs the button parent that the checkbox has been changed to the checked state.
- Payload: gx_event_sender holds the ID of the button widget.

GX_EVENT_TOGGLE_OFF
- Description: This is a GX_SIGNAL event type generated by checkbox style (i.e. toggle style) GX_BUTTON widgets. It informs the button parent that the checkbox has been changed to the unchecked state.
- Payload: gx_event_sender holds the ID of the button widget.

GX_EVENT_RADIO_SELECT
- Description: This is a GX_SIGNAL event type generated by radio button style (i.e. exclusive style) GX_BUTTON widgets. It informs the button parent that the radio button has been changed to the on state.
- Payload: gx_event_sender holds the ID of the button widget.

GX_EVENT_RADIO_DESELECT
- Description: This is a GX_SIGNAL event type generated by radio button style (i.e. exclusive style) GX_BUTTON widgets. It informs the button parent that the radio button has been changed to the off state.
- Payload: gx_event_sender holds the ID of the button widget.

GX_EVENT_CLICKED
- Description: This is a GX_SIGNAL event type generated by all enabled widget types. This event informs the widget parent that the user has clicked on the child widget.
- Payload: gx_event_sender holds the ID of the widget.

GX_EVENT_LIST_SELECT
- **Description:** This is a GX_SIGNAL event type generated by all horizontal list, vertical list, scroll wheel, and drop-list style child widgets. This event informs the widget parent that the user has selected a new list entry.
  - **Payload:** `gx_event_sender` holds the ID of the widget.
    - `gx_event_longdata` holds the new list selection index.

**GX_EVENT_VERTICAL_FLICK**
- **Description:** This event is generated internally when the pen is dragged and released while moving in a vertical direction. `gx_scroll_wheel` and `gx_vertical_list` widgets catch this event to implement animated flicking of the list.
  - **Payload:** `gx_event_intdata[0]` holds the pen speed.

**GX_EVENT_HORIZONTAL_FLICK**
- **Description:** This event is generated internally when the pen is dragged and released while moving in a horizontal direction. `gx_horizontal_list` widgets catch this event to implement animated flicking of the list.
  - **Payload:** `gx_event_intdata[0]` holds the pen speed.

**GX_EVENT_PARENT_SIZED**
- **Description:** This event is generated internally when any GX_WINDOW derived widget type is resized using `gx_widget_resize()`. This allows child widgets like scroll bars to resize themselves as need to fit within the new parent window dimensions.
  - **Payload:** None

**GX_EVENT_CLOSE_POPUP**
- **Description:** This event is used internally to close the popup list that is owned by a drop down list widget.
  - **Payload:** None

**GX_EVENT_ZOOM_IN**
- **Description:** This event is generated by multi-touch touch input drivers to indicate a zoom-in gesture has been input by the user.
  - **Payload:** None

**GX_EVENT_ZOOM_OUT**
- **Description:** This event is generated by multi-touch touch input drivers to indicate a zoom-out gesture has been input by the user.
  - **Payload:** None

**GX_EVENT_LANGUAGE_CHANGE**
- **Description:** This event is generated and delivered to all visible widgets when the active language is changed by calling `gx_display_active_language_set()`. This allows text based widgets to retrieve the new string associated with the active language.
  - **Payload:** None

**GX_EVENT_RESOURCE_CHANGE**
- **Description:** This event is generated and delivered to all visible widgets when the active theme is changed. This allows widgets using pixmap and font resources to mark themselves dirty and redraw using the new theme.
  - **Payload:** None
GX_EVENT_ANIMATION_COMPLETE
- Description: This event is generated when an animation being executed
  by the gx_animation_manager is completed
- Payload: gx_event_target is set to the animation_parent
  gx_event_sender holds the animation id

GX_EVENT_SPRITE_COMPLETE
- Description: This GX_SIGNAL event is generated by gx_sprite widgets
  when the sprite animation sequence is completed.
- Payload: gx_event_sender holds the sprite widget id

GX_EVENT_TEXT_EDITED
- Description: This GX_SIGNAL event is generated by single line and
  multi line text input widgets when the text string is edited by the
  user.
- Payload: gx_event_sender holds the text input widget id

GX_EVENT_FOCUS_NEXT
- Description: This event can be generated by the application or by
  input driver(s) to move the widget input focus to the next widget in
  the widget focus list. When a gx_window type widget is made visible,
  it automatically creates a linked list of child widgets that accept
  input focus. This event can be used to move focus from one child
  widget to the next.
- Payload: None.

GX_EVENT_FOCUS_PREVIOUS
- Description: This event can be generated by the application or by
  input driver(s) to move the widget input focus to the previous widget
  in the widget focus list. When a gx_window type widget is made
  visible, it automatically creates a linked list of child widgets that
  accept input focus. This event can be used to move focus from one
  child widget to the previous widget.
- Payload: None.

GX_EVENT_FOCUS_GAIN_NOTIFY
- Description: This GX_SIGNAL style event can be generated by a child
  widgets when they gain input focus. In order for a child widget to
  generate this signal, the child widget must have a non-zero ID and it
  must have the GX_STATUS_NOTIFY_ON_GAIN_FOCUS status flag set.
- Payload: gx_event_sender holds the child widget ID.

GX_EVENT_SELECT
- Description: This event can be generated by the application to place
  a button in the selected or pushed state.
- Payload: None.

GX_EVENT_DESELECT
- Description: This event can be generated by the application to place
  a button in the non selected or not pushed state.
- Payload: None.

GX_EVENT_PROGRESS_VALUE
- Description: This is a GX_SIGNAL type event generated by progress_bar
  type widgets when the progress bar value is changed.
- Payload: gx_event_longdata holds the new progress bar value.
GX_EVENT_TOUCH_CALIBRATION_COMPLETE
- Description: This event is sent by the generic resistive touch screen input driver when the touch screen calibration sequence is completed. This notifies the application that the normal screen display can begin or resume after a calibration sequence has been performed.
- Payload: None.

GX_EVENT_INPUT_RELEASE
- Description: This event is a command telling any widget that has captured the user input (touch, keypad) to release it. This command event is used by the screen drag animation event handler to force child widgets to release an input capture, but can also be generated by the application.
- Payload: None.

GX_EVENT_TREE_SELECT
- Description: This event is generated by gx_tree_view widgets when a tree node is selected by the user.
- Payload: gx_event_sender contains the tree widget ID. gx_event_longdata holds the ID of the selected tree node.

GX_EVENT_STYLE_CHANGED
- Description: This event is generated when a widget style is changed using gx_widget_style_add() or gx_widget_style_remove() APIs. This allows the target widget to redraw if required by the style change.
- Payload: gx_event_ulongdata holds the previous widget style flags. gx_event_target points at the modified widget.

GX_EVENT_CLIENT_UPDATED
- Description: This event is generated when the client area of a window is modified by the addition or removal of non-client children, such as the addition or removal of a scroll bar.
- Payload: None.

GX_EVENT_CUT
- Description: This event is generated by input device drivers to command a text input widget to cut the selected text to the GUIX clipboard.
- Payload: None.

GX_EVENT_COPY
- Description: This event is generated by input device drivers to command a text input widget to copy the selected text to the GUIX clipboard.
- Payload: None.

GX_EVENT_PASTE
- Description: This event is generated by input device drivers to command a text input widget to paste the selected text to the GUIX clipboard.
- Payload: None.

GX_EVENT_MARK_NEXT
- Description: This event is generated by input device drivers to command a text input widget to mark the next character in the input string.
GX_EVENT_MARK_PREVIOUS
- Description: This event is generated by input device drivers to command a text input widget to mark the previous character in the input string.
- Payload: None.

GX_EVENT_MARK_UP
- Description: This event is generated by input device drivers to command a text input widget to mark the previous row of characters in the input string.
- Payload: None.

GX_EVENT_MARK_DOWN
- Description: This event is generated by input device drivers to command a text input widget to mark the following row of characters in the input string.
- Payload: None.

GX_EVENT_MARK_END
- Description: This event is generated by input device drivers to command a text input widget to move the end marker to the end of the input string.
- Payload: None.

GX_EVENT_MARK_HOME
- Description: This event is generated by input device drivers to command a text input widget to move the start marker to the beginning of the input string.
- Payload: None.
Appendix F: GUIX RTOS Binding Services

GUIX requires thread or tasking services, mutex, event queue, and timing services providing by the underlying RTOS. By default GUIX is configured to utilize the ThreadX real time operating system to provide these services. To port GUIX to another operating system, the developer should define the pre-processor directive GX_DISABLE_THREADX_BINDING and rebuild the GUIX library to remove the ThreadX dependencies. In addition, the developer will need to provide the following macro definitions and supporting functions. Examples of these macro definitions and supporting functions can be found in the files gx_system_rtos_bind.h and gx_system_rtos_bind.c, which provide an example generic rtos integration.

System Integration macros:

GX_RTOS_BINDING_INITIALIZE

This macro is invoked during system initialization. The macro should be defined to call any function needed to prepare your rtos system services or rtos resources prior to use. This is the binding’s opportunity to prepare the rtos resources that GUIX will use.

GX_SYSTEM_THREAD_START

This macro is invoked when the GUIX task or thread should start executing. This macro should be defined to call a function which will start the GUIX thread running. The entry point to the GUIX thread is passed to the called function. The signature of the called function must be

UINT function_name(VOID (thread_entry_point)(VOID));

This function should return GX_SUCCESS if the thread is successfully started, or GX_FAILURE.

GX_EVENT_PUSH

This macro is invoked to push an event into the FIFO event queue used by GUIX. When porting to a new rtos, it is your responsibility to implement this event queue in a thread-safe manner. GX_EVENT structures must be copied into this queue and copied out of this queue, i.e. a queue of GX_EVENT pointers will not work, since GUIX events can be automatic variables from the view of the event producer. The signature of the function called by this macro must be:
UINT function_name(GX_EVENT *event_ptr);

This function should return GX_SUCCESS if the event is pushed into the event queue, otherwise it should return GX_FAILURE.

GX_EVENT_POP

This macro is invoked to remove the head (oldest) event from the GUIX event queue and copy it into the requested location. This function must be able to optionally block or wait for an event if no events are currently in the event queue. The signature of the function invoked by this macro must be

UINT function_name(GX_EVENT *put_event, GX_BOOL wait)

If the wait parameter == GX_TRUE, the function should not return until an event is provided. If the wait parameter is GX_FALSE, the function should return immediately with or without an event.

If an event is retrieved from the queue, it should copied into the put_event location and the return status is GX_SUCCESS. Otherwise the return status should be GX_FAILURE.

GX_EVENT_FOLD

This macro is invoked by GUIX to fold an event into the FIFO event queue. Folding an event means that if an event of the same type already exists in the queue, that entry is update to contain the payload of the new event. If an existing event of the same type is not found in the queue, a new event is pushed into the queue.

For bindings that cannot implement the event fold feature, it is acceptable to simply invoke the GX_EVENT_PUSH.

GX_TIMER_START

This macro is invoked when GUIX needs to receive periodic timer input. This macro should invoke a service that starts the low-level RTOS periodic timer service. If the RTOS timer service cannot be easily stopped and started, it is acceptable but less efficient to leave this service running at all times.

When the low-level RTOS timer service periodically expires, the binding must call the GUIX system function _gx_system_timer_expiration(0); Calling this function periodically is what drives the high-level GUIX timer widget timer services.

GX_TIMER_STOP
This macro is invoked when GUIX no longer needs a periodic timer (i.e. there are no active GUIX timers running). If the RTOS timer service cannot be easily stopped and started, it is acceptable but less efficient to leave this service running at all times and define this macro to do nothing.

**GX_SYSTEM_MUTEX_LOCK**

This macro is invoked by GUIX during critical code sections to prevent another task from pre-empting and modifying common data structures, potentially causing corruption. This macro should call a function that implements the suitable RTOS resource locking service.

If you never utilize any GUIX API services outside of the GUIX thread, you can define this macro to do nothing.

**GX_SYSTEM_MUTEX_UNLOCK**

This macro is invoked at the end of critical code sections, and should unlock the GUIX resource using the suitable underlying RTOS service. If you never utilize any GUIX API services outside of the GUIX thread, you can define this macro to do nothing.

**GX_SYSTEM_TIME_GET**

This macro should call a function that returns the current system time is “system ticks”, which is usually the number of low-level timer interrupts that have occurred since system startup. This service is used to calculate touch event pen speed for touch input gestures. The signature of the function invoked by this macro must be:

```c
ULONG function_name(VOID);
```

**GX_CURRENT_THREAD**

This macro is invoked to identify the currently executing thread. The service called by this macro must return a void *, meaning that the data type used by your operating system to identify the current execution thread must be cast to a void * to be returned to GUIX.

A complete example of a generic RTOS binding is implemented in the file gx_system_rtos_bind.h and gx_system_rtos_bind.c.
Appendix G: GUIX Font Structure

GUIX fonts are normally produced by the GUIX Studio application, and font glyphs are rendered by the GUIX display driver. The application software need only specify the font and colors that each text display widget should use. The GUIX font data structures are documented here for completeness, and to enable developers to create their own methods for generating or converting other fonts into the GUIX font format.

Each GUIX font starts with a GX_FONT structure. The GX_FONT structure defines global font parameters, such as the character included within the font and the line height of the font. The GX_FONT structure points at an array of GX_GLYPH structures. Each GX_GLYPH structure defines the width, height, and baseline offset of one specific character glyph. The GX_GLYPH structure also points to the actual glyph bitmap data (which may be NULL for whitespace characters).

The GX_FONT structure, contained in gx_api.h, is declared as follows:

```c
typedef struct GX_FONT_STRUCT
{
    GX_UBYTE                     gx_font_format
    GX_UBYTE                     gx_font_prespace
    GX_UBYTE                     gx_font_postspace
    GX_UBYTE                     gx_font_line_height
    GX_UBYTE                     gx_font_baseline
    USHORT                       gx_font_first_glyph
    USHORT                       gx_font_last_glyph
    GX_CONST GX_GLYPH           *gx_font_glyphs
    const struct GX_FONT_STRUCT *gx_font_next_page
} GX_FONT;
```

The gx_font_format field defines the font bits-per-pixel and other flags, as defined in the gx_api.h header file.

The gx_font_prespace defines the pixel space to skip above each line of text in a multi-line text display.

The gx_font_postspace field defines the pixel space to skip below each line of text in a multi-line text display.

The gx_font_line_height field defines the height of the tallest glyph in the font.

The gx_font_baseline field defines the distance, in pixels, from the top row of glyph pixels to the font baseline.
The `gx_font_first_glyph` field defines the first Unicode character encoding included in this font page.

The `gx_font_last_glyph` field defines the last Unicode character encoding included in this font page.

The `gx_font_glyphs` pointer points to an array of `GX_GLYPH` structures. This array must be equal in size to the number of characters contained on this font page, i.e. \((gx\_font\_last\_glyph - gx\_font\_first\_glyph) + 1\).

The `gx_font_next_page` member is used for multiple page fonts. Multiple page fonts are used for extended character sets and to optimize the size of the `GX_GLYPH` structure arrays. If all of the characters of the font are contained within one font page, or if this is the last page of the font in question, the `gx_font_next_page` member is set to `GX_NULL`.

As noted above, the `GX_FONT` structure above contains a pointer to an array of `GX_GLYPHS` structures. There must be one `GX_GLYPH` structure for each character on the font page. The `GX_GLYPH` structure is defined as:

```c
typedef struct GX_GLYPH_STRUCT
{
    GXCONST GX_UBYTE *gx_glyph_map;
    GX_BYTE             gx_glyph_ascent;
    GX_BYTE             gx_glyph_descent;
    GX_BYTE             gx_glyph_advance;
    GX_BYTE             gx_glyph_leading;
    GX_UBYTE            gx_glyph_width;
    GX_UBYTE            gx_glyph_height;
} GX_GLYPH;
```

The `gx_glyph_map` pointer points to the glyph bitmap. This pointer may be `GX_NULL` for whitespace characters. The bitmap data is encoded as 1 bpp, 2 bpp, 4 bpp, or 8 bpp alpha values. For 1 bit data, a value of 1 indicates that the pixel should be written in the foreground color, and a value of 0 indicates that the pixel is transparent. For 8 bit data, the values range from 0 (fully transparent) to 255 (fully opaque). All intermediate value represent a blending value for anti-aliased fonts. The glyph bitmap data is always padded to full byte alignment for formats using less than 8bpp data values.

The `gx_glyph_ascent` and `gx_glyph_descent` values position the glyph vertically with respect to the font baseline.

The `gx_glyph_width` and `gx_glyph_height` values specify the size of the glyph bitmap data.
The `gx_glyph_advance` value specifies the pixel width to advance the drawing position after drawing the glyph (this may not be equal to the glyph width).

The `gx_glyph_leading` value specifies the pixels to advance in the x-direction prior to rendering the glyph.
Appendix H: GUIX Build-Time Configuration flags

GUIX support several conditional compilation options and configuration values. The default setting for these conditionals and configuration values can be overridden by pre-defining the value, either in your gx_user.h header file or on your compiler command line.

**GX_DISABLE_THREADX_BINDING**
- Default: Undefined
- Description: This conditional can be used to disable the default ThreadX RTOS binding. If you want to run GUIX with an RTOS other than ThreadX, you should #define GX_DISABLE_THREADX_BINDING and provide your own RTOS binding services.

**GX_SYSTEM_TIMER_MS**
- Default: 20
- Description: This value defines the desired GUIX timer interval or precision.

**TX_TIMER_TICKS_PER_SECOND**
- Default: 100
- Description: This value defines the number of TX timer interrupt frequency. Since the default ThreadX interval timer is usually 10ms, this value defaults to a 100 Hz frequency.

**GX_SYSTEM_TIMER_TICKS**
- Default: \(((\text{GX_SYSTEM_TIMER_MS} \times \text{TX_TIMER_TICKS_PER_SECOND}) / 1000)\)
- Description: This value defines the number of underlying RTOS timer ticks per GUIX timer tick. The default value is 2, meaning the GUIX timer interval is 2 ThreadX timer interrupt intervals, or 20 ms by default.

**GX_DISABLE_MULTITHREAD_SUPPORT**
- Default: Not defined
- Description: This compile-time conditional can be used to disable the GUIX API support for multiple threads invoking the GUIX API concurrently. If only one application thread will ever utilize the GUIX API, you should define this flag to reduce the system overhead associated with protecting critical code sections.

**GX_DISABLE_UTF8_SUPPORT**
- Default: Not Defined.
- Description: This compile-time conditional can be used to remove the GUIX internal support for UTF8 format string encoding. If you are using only character values M- 0xff in your application, turning on this #define will reduce the code size and overhead associated with supporting UTF8 format string encoding.

**GX_DISABLE_ARC_DRAWING_SUPPORT**
- Default: Not defined.
- Description: This conditional can be used to reduce the GUIX library code size and GX_DISPLAY structure size by removing support for the arc-drawing functions circle, arc, pie, and ellipse. These functions are not required by the default GUIX widget set.

**GX_DISABLE_SOFTWARE_DECODER_SUPPORT**
- Default: Not defined.
- Description: This conditional can be defined to remove the GUIX library runtime jpeg and png software decoder support. If your application does not require runtime decode of jpg or png files, meaning your application does not use RAW format pixelmaps produced by Studio and does not read image files from an external filesystem, you can turn on this #define to reduce the GUIX library footprint.

**GX_DISABLE_BINARY_RESOURCE_SUPPORT**
- Default: Not defined
- Description: This conditional can be used to remove the GUIX library support for loading binary resource data.
Binary resources can be used to do runtime binding of resource data with your GUIX application. If you are using only C source code format resource files, you can define this conditional to reduce your GUIX library footprint.

**GX_DISABLE_BRUSH_ALPHA_SUPPORT**

Default: Not defined.
Description: When running at 16 bpp and higher color depths, GUIX optionally supports drawing non-arc graphics, pixelmaps, and fonts with an alpha value defined by the drawing context brush. Supporting this drawing mode introduces a small runtime overhead and library footprint increase, which can be eliminated by defining this flag if you do not require alpha-blending drawing support. Note that pixelmaps with alpha channel, anti-aliased fonts, and other anti-aliasing drawing modes are still supported regardless of this conditional setting.

**GX_REPEAT_BUTTON_INITIAL_TICS**

Default: 10.
Description: If a button has style `GX_STYLE_BUTTON_REPEAT`, this value defines how long the button waits before beginning to send repeated `GX_EVENT_CLICKED` events.

**GX_MAX_QUEUE_EVENTS**

Default: 48.
Description: Defines the size of the GUIX event queue in units of event structure entries. If the event queue overflows, events being pushed into the queue are discarded and `GX_SYSTEM_ERROR` is returned by the `gx_system_event_send()` function.

**GX_MAXDIRY_AREA**

Default: 64.
Description: Defines the maximum number of unique dirty list entries that can be maintained by one canvas. When the dirty list overflows, GUIX will default to marking the canvas root window as dirty, which is less efficient than drawing individual child widgets.
GX_MAX_CONTEXT_NESTING
Default: 8.
Description: Defines the maximum nesting of the drawing context stack. This is equivalent to the maximum nesting of parent/child/child/child widgets within the UI definition.

GX_MAX_INPUT_CAPTURE_NESTING
Default: 4.
Description: Defines the size of the stack used to maintain the list of widgets that have captures the user input (mouse and keyboard).

GX_SYSTEM_THREAD_PRIORITY
Default: 16.
Description: Defines the priority of the GUIX thread created during gx_system_initialize().

GX_SYSTEM_THREAD_TIMESLICE
Default: 10.
Description: Defines the GUIX thread timeslice in terms of RTOS timer ticks. If other threads are defined with the same priority as the GUIX thread, this value determines how often those competing threads are granted CPU control.

GX_CURSOR_BLINK_INTERVAL
Default: 20.
Description: Defines the rate at which the input cursor blinks for text input widgets. This value is in terms of GUIX timer ticks, which by default is defined as 50ms, so a value of 20 indicates that the input cursor blinks once per second.

GX_MULTI_LINE_INDEX_CACHE_SIZE
Default: 32.
Description: Defines the size of the list-start index cache maintained by the multi-line text view and multi-line text input widgets. This cache is used to accomplish fast vertical scrolling of multi line text widgets. For best
performance, the cache size should be set greater than the number of visible rows of the largest multi line text widget defined by the application. For example, if the most visible rows for any text widget is 20 rows, the application might define a cache size of 32 (the default), which allows GUIX to scroll vertically without re-calculating all line start indexes.

**GX_MULTI_LINE_TEXT_BUTTON_MAX_LINES**
Default: 4.
Description: The multi-line text button control block maintains a pointer to each line of text to be displayed by the button. This value determines the number of text pointers needed by the worst case multi-line text button.

**GX_POLYGON_MAX_EDGE_NUM**
Default: 10.
Description: This value determines the most complex polygon that can be drawn by GUIX. The polygon drawing algorithm determines the lines needed to define the polygon edges, and this definition defines the maximum number of edges that can be supported.

**GX_NUMERIC_SCROLL_WHEEL_STRING_BUFFER_SIZE**
Default: 16.
Description: For a number scroll wheel, the scroll wheel widget converts integer values to ascii strings. This value determines the maximum length of the string required to display the assigned integer values.

**GX_DEFAULT_CIRCULAR_GAUGE_ANIMATION_DELAY**
Default: 5.
Description: Defines the number of GUIX timer ticks (50ms) between updates of a circular gauge configured to animate the needle movement between last and current angular position.

**GX_NUMERIC_PROMPT_BUFFER_SIZE**
Default: 16.
Description: A numeric prompt allocates a buffer to convert an integer value assigned to the prompt to an ascii string. This definition defines the size of this character buffer.

GX_ANIMATION_POOL_SIZE
Default: 6.
Description: GUIX defines an animation pool from which animation information structures can be dynamically allocated and returned, using gx_system_animation_get and gx_system_animation_free() APIs. This definition defines the size of this animation control block pool.

GX_MOUSE_SUPPORT
Default: Not defined.
Description: This definition enables support for mouse input. Software mouse requires that the display driver draw and track the mouse cursor, which adds extra overhead to the display driver. This definition should only be defined when a mouse (not a touch screen) must be supported.

GX_HARDWARE_MOUSE_SUPPORT
Default: Not defined.
Description: When this definition is defined, the GUIX display driver utilizes hardware mouse cursor drawing support. This reduces the memory required to capture the canvas memory beneath the mouse cursor and improves system performance for those hardware targets support a mouse overlay graphics layer.

GX_FONT_KERNING_SUPPORT
Default: Not defined.
Description: This definition can be defined to enable font kerning support. Font kerning improves glyph spacing for certain glyph combinations. This support adds a small amount of overhead to the runtime string drawing functions, and also adds a small amount of size to the font data structures.

GX_WIDGET_USER_DATA
Default: Not defined.
Description: If defined, this adds a user-defined data field to the GX_WIDGET control block. This data field can be assigned using the properties view within GUIX Studio. This data field is ignored by GUIX internally, but can be used by application software for many purposes.

GUIX_5_4_0_COMPATIBILITY
Default: Not defined.
Description: Certain GUI APIs were modified after release 5.4.0 to add support for disabled text colors and to improve the accuracy of certain math functions by using fixed point match parameters. These changes make GUIX library releases after 5.4.0 incompatible with previous releases. However, by turning on this #define, the library can be built such that the APIs fully compatible with releases <= 5.4.0, meaning that no changes are needed in existing applications to compile with the latest GUIX library release.

GX_MAX_STRING_LENGTH
Default: 102400
Description: Defines the maximum length of a string, which is used to test invalid strings. If the input string is exceeding the maximum string length, it will be regard as invalid.
Appendix I: GUIX Information Structures

GX_CIRCULAR_GAUGE_INFO

Definition

typedef struct GX_CIRCULAR_GAUGE_INFO_STRUCT
{
    INT         gx_circular_gauge_info_animation_steps;
    INT         gx_circular_gauge_info_animation_delay;
    GX_VALUE    gx_circular_gauge_info_needle_xpos;
    GX_VALUE    gx_circular_gauge_info_needle_ypos;
    GX_VALUE    gx_circular_gauge_info_needle_xcor;
    GX_VALUE    gx_circular_gauge_info_needle_ycor;
    GX_RESOURCE_ID gx_circular_gauge_info_needle_pixelmap;
} GX_CIRCULAR_GAUGE_INFO;

Members

gx_circular_gauge_info_animation_steps  Total steps the needle will travel through when moving from the current needle angle to a newly assigned needle angle

gx_circular_gauge_info_animation_delay  The number of GUIX clock ticks to delay between animation steps

gx_circular_gauge_info_needle_xpos     The distance from the left of the gauge widget to the center-of-rotation of the gauge needle

gx_circular_gauge_info_needle_ypos     The distance from the top of the gauge widget to the center-of-rotation of the gauge needle

gx_circular_gauge_info_needle_xcor     The distance from the left of the needle image to the center-of-rotation of the gauge needle

gx_circular_gauge_info_needle_ycor     The distance from the top of the needle image to the center-of-rotation of the gauge needle

gx_circular_gauge_info_needle_pixelmap Resource ID of the pixelmap which will be used to draw the gauge needle. This image will be rotated as needed by the gauge widget to display the gauge needle in any position

The diagram below illustrates the xpos, ypos and xcor, ycor coordinates:
GX_LINE_CHART_INFO

Definition

typedef struct GX_LINE_CHART_INFO_STRUCT
{
    INT            gx_line_chart_min_val;
    INT            gx_line_chart_max_val;
    INT           *gx_line_chart_data;
    GX_VALUE       gx_line_left_margin;
    GX_VALUE       gx_line_top_margin;
    GX_VALUE       gx_line_right_margin;
    GX_VALUE       gx_line_bottom_margin;
    GX_VALUE       gx_line_chart_max_data_count;
    GX_VALUE       gx_line_chart_active_data_count;
    GX_VALUE       gx_line_chart_axis_line_width;
    GX_VALUE       gx_line_chart_data_line_width;
    GX_RESOURCE_ID gx_line_chart_axis_color;
    GX_RESOURCE_ID gx_line_chart_line_color;
} GX_LINE_CHART_INFO;

Members

gx_line_chart_min_val    The minimum data value, which is used to calculate scaling

gx_line_chart_max_val    The maximum data value, which is used to calculate scaling

gx_line_chart_data       Pointer to an array of integer values. These are the integer values plotted by the line chart widget

gx_line_<side>_margin    The offset from the chart window outer bound to the actual chart rendering area. The chart axis and data line are always plotted within this inner boundery, which allows the application to draw labels and other information inside the chart window but outside the chart graphing area

gx_line_chart_max_data_count    The number of data values which may be present. This parameter is used for calculating the x-axis scaling or interval for plotting data points.

gx_line_active_data_count    The number of data values that actually present in the data array. A line chart may be scaled to draw a maximum of 100 values (for example), but on any particular update a smaller number of data values may actually be present.

gx_line_axis_line_width    Width of the line used to draw the horizontal and vertical axis

gx_line_data_line_width    Width of the plotted data line
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gx_line_chart_axis_color</code></td>
<td>Resource ID of the color used to draw the axis lines</td>
</tr>
<tr>
<td><code>gx_line_chart_line_color</code></td>
<td>Resource ID of the color used to draw the chart data line</td>
</tr>
</tbody>
</table>
**GX_MOUSE_CURSOR_INFO**

**Definition**

typedef struct GX_MOUSE_CURSOR_INFO_STRUCT
{
    GX_RESOURCE_ID            gx_mouse_cursor_image_id;
    GX_VALUE                  gx_mouse_cursor_hotspot_x;
    GX_VALUE                  gx_mouse_cursor_hotspot_y;
} GX_MOUSE_CURSOR_INFO;

**Members**

- **gx_mouse_cursor_image_id**  
  Resource ID of the mouse image
- **gx_mouse_cursor_hotspot_x**  
  The offset from the left of the mouse image to the mouse image hotspot
- **gx_mouse_cursor_hotspot_y**  
  The offset from the top of the mouse image to the mouse image hotspot
GX_PEN_CONFIGURATION

Definition

typedef struct GX_PEN_CONFIGURATION_STRUCT
{
    GX_FIXED_VAL gx_pen_configuration_min_drag_dist;
    UINT     gx_pen_configuration_max_pen_speed_ticks;
}GX_PEN_CONFIGURATION;

Members

gx_pen_configuration_min_drag_dist
    The minimum drag distance per GUIX timer tick to trigger an FLICK event. Call GX_FIXED_VAL_MAKE to make a fixed point data type value

gx_pen_configuration_max_pen_speed_ticks
    The maximum drag speed in GUIX timer ticks to trigger an FLICK event
GX_PIXELMAP_SLIDER_INFO

**Definition**

typedef struct GX_PIXELMAP_SLIDER_INFO_STRUCT
{
    GX_RESOURCE_ID gx_pixelmap_slider_info_lower_background_pixelmap;
    GX_RESOURCE_ID gx_pixelmap_slider_info_upper_background_pixelmap;
    GX_RESOURCE_ID gx_pixelmap_slider_info_needle_pixelmap;
} GX_PIXELMAP_SLIDER_INFO;

**Members**

- **gx_pixelmap_slider_info_lower_background_pixelmap**
  Resource ID of the pixelmap for filling the background before the needle. If upper background pixelmap is not set, it’s used for filling background both before and after the needle

- **gx_pixelmap_slider_info_upper_background_pixelmap**
  Resource ID of the pixelmap for filling background after the needle

- **gx_pixelmap_slider_info_needle_pixelmap**
  Resource ID of the needle pixelmap
**GX_PROGRESS_BAR_INFO**

**Definition**

typedef struct GX_PROGRESS_BAR_INFO_STRUCT  
{  
    INT  gx_progress_bar_info_min_val;  
    INT  gx_progress_bar_info_max_val;  
    INT  gx_progress_bar_info_current_val;  
    GX_RESOURCE_ID  gx_progress_bar_font_id;  
    GX_RESOURCE_ID  gx_progress_bar_normal_text_color;  
    GX_RESOURCE_ID  gx_progress_bar_selected_text_color;  
    GX_RESOURCE_ID  gx_progress_bar_disabled_text_color;  
    GX_RESOURCE_ID  gx_progress_bar_fill_pixelmap;  
} GX_PROGRESS_BAR_INFO;

**Members**

- **gx_progress_bar_info_min_val**: Minimum reported value
- **gx_progress_bar_info_max_val**: Maximum reported value
- **gx_progress_bar_info_current_val**: Current value
- **gx_progress_bar_info_font_id**: Resource ID of the font, used to draw the optional text value within the progress bar widget
- **gx_progress_bar_normal_text_color**: Resource ID of the text color in normal state, used to define the optional text drawing within the progress bar widget
- **gx_progress_bar_selected_text_color**: Resource ID of the text color when the widget gain focus, used to define the optional text drawing within the progress bar widget
- **gx_progress_bar_disabled_text_color**: Resource ID of the text color when GX_STYLE_ENABLED is not active, used to define the optional text drawing within the progress bar widget
- **gx_progress_bar_fill_pixelmap**: Resource ID of the pixelmap for background filling
**GX_RADIAL_PROGRESS_BAR_INFO**

**Definition**

typedef struct GX_RADIAL_PROGRESS_BAR_INFO_STRUCT
{
    GX_VALUE    gx_radial_progress_bar_info_xcenter;
    GX_VALUE    gx_radial_progress_bar_info_ycenter;
    GX_VALUE    gx_radial_progress_bar_info_radius;
    GX_VALUE    gx_radial_progress_bar_info_current_val;
    GX_VALUE    gx_radial_progress_bar_info_anchor_val;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_font_id;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_normal_text_color;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_selected_text_color;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_disabled_text_color;
    GX_VALUE    gx_radial_progress_bar_info_normal_brush_width;
    GX_VALUE    gx_radial_progress_bar_info_selected_brush_width;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_normal_brush_color;
    GX_RESOURCE_ID  gx_radial_progress_bar_info_selected_brush_color;
} GX_RADIAL_PROGRESS_BAR_INFO;

**Members**

- **gx_radial_progress_bar_info_xcenter**: Widget position in x coordinate
- **gx_radial_progress_bar_info_ycenter**: Widget position in y coordinate
- **gx_radial_progress_bar_info_radius**: Radius of the progress circle
- **gx_radial_progress_bar_info_current_val**: Current value, limited to the range \([-360, 360]\), indicates the angular delta between the anchor position and the end point of the upper arc. Negative value causes the arc to be drawn in a clockwise direction starting at the anchor position. Positive value causes the arc to be drawn in a counter-clockwise direction starting at the anchor position. The application must scale the real-word value being indiated to assign an angular value to the progress bar widget
- **gx_radial_progress_bar_info_anchor_val**: Starting angle of the upper progress arc. The value is defined in terms of integer degree with 0 degree pointing to the right and 90 degree indicating straight up position.
- **gx_radial_progress_bar_info_font_id**: Resource ID of the font used to draw the optional text value within the progress bar widget
- **gx_radial_progress_bar_info_normal_text_color**: Resource ID of the text color in normal state, used to define the optional text drawing within the progress bar widget
- **gx_radial_progress_bar_info_selected_text_color**: Resource ID of the text color when widget gain focus, used to define the
optional text drawing within the progress bar widget

gx_radial_progress_bar_disabled_text_color
Resource ID of the text color when GX_STYLE_ENABLED is not active, used to define the optional text drawing within the progress bar widget

gx_radial_progress_bar_normal_brush_width
Width of the lower progress circle

gx_radial_progress_bar_selected_brush_width
Width of the upper progress arc, the upper arc may be narrower, the same as, or wider than the lower circle

gx_radial_progress_bar_normal_brush_color
Resource ID of the color to fill lower progress circle

gx_radial_progress_bar_selected_brush_color
Resource ID of the color to fill upper progress arc
GX_RADIAL_SLIDER_INFO

Definition

typedef struct GX_RADIAL_SLIDER_INFO_STRUCT
{
    GX_VALUE       gx_radial_slider_info_xcenter;
    GX_VALUE       gx_radial_slider_info_ycenter;
    USHORT         gx_radial_slider_info_radius;
    USHORT         gx_radial_slider_info_track_width;
    GX_VALUE       gx_radial_slider_info_current_angle;
    GX_VALUE       gx_radial_slider_info_min_angle;
    GX_VALUE       gx_radial_slider_info_max_angle;
    GX_VALUE       *gx_radial_slider_info_angle_list;
    USHORT         gx_radial_slider_info_list_count;
    GX_RESOURCE_ID gx_radial_slider_info_background_pixelmap;
    GX_RESOURCE_ID gx_radial_slider_info_needle_pixelmap;
} GX_RADIAL_SLIDER_INFO;

Members

gx_radial_slider_info_xcenter       Distance from the left of the slider widget to the center-of-rotation of the slider needle

gx_radial_slider_info_ycenter       Distance from the top of the slider widget to the center-of-rotation of the slider needle

gx_radial_slider_info_radius        Radius of the radial slider circle

gx_radial_slider_info_track_width   Width of radial slider track

gx_radial_slider_info_current_angle Current slider angle

gx_radial_slider_info_min_angle     Minimum slider angle

gx_radial_slider_info_max_angle     Maximum slider angle

gx_radial_slider_info_angle_list    Angle value list, defines anchor angles, if set, slider angle can only be one of the defined anchor angles

gx_radial_slider_info_list_count    Number of anchor angles

gx_radial_slider_info_background_pixelmap Resource ID of background pixelmap

gx_radial_slider_info_needle_pixelmap Resource ID of needle pixelmap
**Definition**

typedef struct GX_RECTANGLE_STRUCT
{
    GX_VALUE gx_rectangle_left;
    GX_VALUE gx_rectangle_top;
    GX_VALUE gx_rectangle_right;
    GX_VALUE gx_rectangle_bottom;
} GX_RECTANGLE;

**Members**

- **gx_rectangle_left**: Left of the rectangle
- **gx_rectangle_top**: Top of the rectangle
- **gx_rectangle_right**: Right of the rectangle
- **gx_rectangle_bottom**: Bottom of the rectangle
GX_SCROLL_INFO

Definition

typedef struct GX_SCROLL_INFO_STRUCT
{
  INT       gx_scroll_value;
  INT       gx_scroll_minimum;
  INT       gx_scroll_maximum;
  GX_VALUE  gx_scroll_visible;
  GX_VALUE  gx_scroll_increment;
} GX_SCROLL_INFO;

Members

gx_scroll_value                  Current scroll position
gx_scroll_minimum                Minimum reported position
gx_scroll_maximum                Maximum reported position
gx_scroll_visible                Parent window visible range
gx_scroll_increment              Scrollbar minimum delta value
GX_SCROLLBAR_APPEARANCE

Definition

typedef struct GX_SCROLLBAR_APPEARANCE_STRUCT
{
    GX_VALUE       gx_scroll_width;
    GX_VALUE       gx_scroll_thumb_width;
    GX_VALUE       gx_scroll_thumb_travel_min;
    GX_VALUE       gx_scroll_thumb_travel_max;
    GX_UBYTE       gx_scroll_thumb_border_style;
    GX_RESOURCE_ID gx_scroll_fill_pixelmap;
    GX_RESOURCE_ID gx_scroll_thumb_pixelmap;
    GX_RESOURCE_ID gx_scroll_up_pixelmap;
    GX_RESOURCE_ID gx_scroll_down_pixelmap;
    GX_RESOURCE_ID gx_scroll_thumb_border_color;
    GX_RESOURCE_ID gx_scroll_button_color;
} GX_SCROLLBAR_APPEARANCE;

Members

gx_scroll_width
Width of the scrollbar widget, in pixels

gx_scroll_thumb_width
Width of the thumb button which slides on the scrollbar, in pixels. This value is usually some number of pixels less than the total scrollbar width

gx_scroll_thumb_travel_min
Offset from the end of scrollbar to minimum thumb button travel point. This limit can be used to prevent the thumb button from travelling to the very end of the scrollbar

gx_scroll_thumb_travel_max
Offset from the end of scrollbar to maximum thumb button travel point. This limit can be used to prevent the thumb button from travelling to the very end of the scrollbar

gx_scroll_thumb_border_style
Border styles of thumb button

gx_scroll_fill_pixelmap
Optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar uses this pixelmap to draw the scrollbar background

gx_scroll_thumb_pixelmap
Optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar thumb button uses this pixelmap to draw itself

gx_scroll_up_pixelmap
Optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar uses this pixelmap ID to draw the scrollbar left/up end button

gx_scroll_down_pixelmap
Optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar uses this
**gx_scroll_thumb_color**
Pixelmap ID to draw the scrollbar right/down end button

**gx_scroll_thumb_border_color**
Resource ID of color used to fill thumb button

**gx_scroll_button_color**
Resource ID of color used to draw the border of thumb button

Resource ID of color used to fill scrollbar end buttons
GX_SLIDER_INFO

Definition

typedef struct GX_SLIDER_INFO_STRUCT
{
    INT      gx_slider_info_min_val;
    INT      gx_slider_info_max_val;
    INT      gx_slider_info_current_val;
    INT      gx_slider_info_increment;
    GX_VALUE gx_slider_info_min_travel;
    GX_VALUE gx_slider_info_max_travel;
    GX_VALUE gx_slider_info_needle_width;
    GX_VALUE gx_slider_info_needle_height;
    GX_VALUE gx_slider_info_needle_inset;
    GX_VALUE gx_slider_info_needle_hotspot_offset;
} GX_SLIDER_INFO;

Members

gx_slider_info_min_val                        Minimum reported value
gx_slider_info_max_val                        Maximum reported value
gx_slider_info_current_value                 Current value
gx_slider_info_min_travel                    Needle travel limit
gx_slider_info_max_travel                    Needle travel limit
gx_slider_info_needle_width                  Needle width in pixel
gx_slider_info_needle_height                 Needle height in pixel
gx_slider_info_needle_inset                  Needle draw position. If
                                              GX_STYLE_SLIDER_VERTICAL is set,
                                              used to specify the offset from the needle
draw start position to the slider left. Else,
                                              used to specify the offset from the needle
draw start position to the slider top.
                                              
gx_slider_info_needle_hotspot_offset        Needle hotspot_offset, used to specify the
                                              offset from the needle draw start position
to the slider hotspot.
**GX_SPRITE_FRAME**

**Definition**

typedef struct GX_SPRITE_FRAME_STRUCT
{
    GX_RESOURCE_ID gx_sprite_frame_pixelmap;
    GX_VALUE       gx_sprite_frame_x_offset;
    GX_VALUE       gx_sprite_frame_y_offset;
    UINT           gx_sprite_frame_delay;
    UINT           gx_sprite_frame_background_operation;
    UCHAR          gx(sprite_frame_alpha;
} GX_SPRITE_FRAME;

**Members**

- **gx_sprite_frame_pixelmap**: Resource ID of the pixelmap to be displayed for this frame. The ID can be 0.
- **gx_sprite_frame_x_offset**: Offset from the sprite widget left to display the pixelmap.
- **gx_sprite_frame_y_offset**: Offset from the sprite widget top to display the pixelmap.
- **gx_sprite_frame_delay**: Delay value, in GUIX timer ticks, after displaying this frame before advancing to the next sprite frame.
- **gx_sprite_frame_background_operation**: Define how the background should be erased. Possible values for this field are:
  - GX_SPRITE_BACKGROUND_NO_ACTION: No fill between frames.
  - GX_SPRITE_BACKGROUND_SOLID_FILL: Re-draw sprite background.
  - GX_SPRITE_BACKGROUND_RESTORE: Restore previous pixelmap.
- **gx_sprite_frame_alpha**: Alpha value to be added to the displayed pixelmap. The value 255 specifies that no extra alpha value should be imposed. If the pixelmap includes an alpha channel, this alpha channel will be added to the frame alpha value.
Index

alpha channel ............ 20, 34, 37
ANSI C .................. 1, 2, 3, 5, 45, 55
anti-aliasing ............. 2, 12, 43
API
  call ....................... 9, 45, 55
  drawing ................... 35, 36
GUIX API function ....... 14, 15, 19, 31
  object creation .......... 15
  service ................... 48, 532
ASCII .................... 7, 24, 71, 600, 605, 606
buffer
  composite ................ 23
  frame 20, 21, 22, 29, 33, 34, 44, 56, 113, 129, 145, 756, 758, 762, 767
  local frame ............. 21, 22
  ping-pong ............... 22, 23
canvas
  alpha channel .......... 11, 34, 35
  blend ..................... 34
  control block .. 11, 34, 107, 113, 114, 115, 116, 118, 129, 143, 145, 146
  creation .................. 11, 34
  drawing .... 33, 36, 64, 113, 116, 117, 118, 119, 130, 146
GUIX canvas component 11, 33
  managed ................. 33, 107, 113
memory
  architecture ........... 756
  buffer ................... 20
  canvas ...... 27, 36, 43, 114, 756, 761, 762
  object ................... 34
  overlay ................... 34
  simple ................... 33
  Z-order ................... 34
color depth ... 2, 20, 36, 43, 107, 756, 759, 760, 761, 762, 763, 764, 765
color format 20, 29, 185, 756, 759, 760, 761, 762, 763, 764, 765
compiler ............ xxii, 3, 9, 26, 27
configuration ........... 9, 758
data type ............... xxii, 9, 48
demo thread ............ 767, 768
dirty list .................. 17
display driver xx, 2, 11, 20, 30, 34, 38, 44, 193, 195, 756, 762, 766
display memory architecture ... 21
draw pixelmap ... 64, 67, 132, 134, 135, 136, 137, 138, 139, 321, 322, 328, 334
event notification .... 12, 48, 49, 57
event processing ... 11, 15, 17, 19, 50, 51, 57, 72, 78, 152, 395, 480, 522, 524, 592, 688, 690, 691, 741
event queue .... 14, 15, 17, 19, 30, 48, 529, 551
global 7, 10, 27, 30, 34, 38, 46, 55, 114
GRAM .................... 21, 23
GUIX components ...... 45
GUIX objects ............... 15
GUIX system mutex ....... 30
GUIX thread ... 7, 9, 14, 15, 19, 27, 30, 31, 44, 50, 52, 57
GUIX widget .... 14, 15, 29, 41, 43, 45, 55
input drivers ............ 14, 19
LCD display .............. 756
memory
  architecture ........... 21
  buffer ................... 20
  canvas ...... 27, 36, 43, 114, 756, 761, 762
  constraints ............. 21
  dynamic ................... 27
  frame buffer ............ 20
  object 3, 7, 15, 29, 34, 38, 45, 54, 55
overlay .................... 23, 34
periodic processing ....... 7
pixelmaps ... 27, 38, 154, 155, 324, 325, 329, 330, 331, 337, 338, 384
queue, event ... 14, 15, 17, 19, 30, 48, 529, 551
runtime library .......................... 8
screen control block ........ 12, 38
screen driver ........ 34, 43, 53, 107
screen refresh .. 16, 23, 30, 31, 53
screens......................... 2, 14, 15
scrolling ........ 55, 56, 57, 750, 761
setup .. 6, 190, 191, 192, 193, 196, 197, 198, 199, 200, 201, 202, 757
skinning.................. 2, 11, 41
stack size .................... 9, 27
static text .................... 40
string table .. 38, 70, 553, 554, 768
system error handling ....... 11, 32
ThreadX ii, xx, xxii, xxiii, 1, 2, 3, 4, 5, 7, 8, 9, 14, 20, 31, 767
ThreadX timer ........ 7, 14, 20, 31
tile pixelmap .............. 64, 140
user interface ... 3, 4, 6, 11, 14, 15
utility component ........ 12, 58, 61
version_id .................. xxiii, 10
widget component ........ 12, 45
widget control block ...... 12, 45, 46
widget creation .......... 12, 45
widget defaults .......... 11, 39
window
  background .................. 56
  border ........................ 55
  children .................. 44, 55, 56
  component ................ 54
  control block ........ 737, 738
  event handler .......... 57
GUIX .... 15, 19, 31, 54, 55, 56
  object .................. 55
  processing .............. 54
root .. 8, 17, 43, 44, 55, 56, 73, 562, 743, 745, 746, 747, 749, 767, 768, 769