User's Manual

GUIX^{TM}

User's Manual: Software

Renesas Synergy^{TM} Platform

synergygallery.renesas.com

Renesas Electronics
www.renesas.com

Rev. 5

GUIX

the high-performance embedded GUI

User Guide

Version 5

Express Logic, Inc.
858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
http://www.expresslogic.com

Part Number: 000-1115
Revision 5.2.5

Table of Contents

Table of Contents iii

About This Guide xv

  Organization xv

Guide Conventions xvi

GUIX Data Types xvii

Customer Support Center xviii

  What We Need From You xviii

  Where to Send Comments About This Guide xix

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 9

GUIX Version ID 10
Chapter 3: Functional Overview of GUIX 11
Execution Overview 13
Initialization 13
Application Interface Calls 14
Internal GUIX Thread 14
Event Processing 15
Drawing 15
User Input 17
Modal Dialog Execution 17
Periodic Processing 18
Display Driver 18
Display Memory Architectures 19
Memory Usage 21
Static Memory Usage 21
Dynamic Memory Usage 22
GUIX Components 22
GUIX System Component 23
RTOS Binding 23
Initialization 23
String Table Management 24
Thread Processing 25
Multithread Safety 25
Periodic Processing 25
System Error Handling 26
GUIX Canvas Component 26
Canvas Creation 27
Canvas Control Block 27
Canvas Alpha Channel 27
Canvas Drawing 28
Color Depth 28
GUIX Display Component 29
Display Creation 29
Display Control Block 29

Resource Management ... 29
Widget Defaults ... 30
Scrollbar Appearance ... 32
Skinning and Themes ... 32
Drawing APIs ... 35
Root Window ... 36
Anti-Aliasing ... 36
Clipping ... 36
Views ... 37
Display Driver Interface ... 37
GUIX Widget Component ... 38
Widget Creation ... 38
Widget Control Block ... 38
Dynamic Widget Control Block Allocation and De-allocation ... 39
Types ... 39
Styles ... 40
Colors ... 41
Event Notification ... 41
Event Processing ... 43
Implementing Custom Event Processing (example) ... 43
Drawing Function ... 44
Implementing Custom Drawing (example) ... 45
GUIX Drawing Context Component ... 46
GUIX Window Component ... 47
Window Creation ... 48
Window Control Block ... 48
Root Window ... 48
Background ... 49
Scrolling ... 49
Event Notification ... 50
Event Processing ... 50
GUIX Animation Component ... 50
GUIX Utility Component ... 52
Chapter 4: Description of GUIX Services ... 53

gx_animation_canvas_define 60
gx_animation_create 62
gx_animation_delete........ 63
gx_animation_start 64
gx_animation_stop........ 66
gx_brush_default 67
gx_brush_define 68
gx_button_background_draw 70
gx_button_create 71
gx_button_deselect........ 73
gx_button_draw 74
gx_button_event_process........ 75
gx_button_select........ 76
gx_canvas_alpha_set 77
gx_canvas_arc_draw........ 78
gx_canvas_block_move 79
gx_canvas_circle_draw 81
gx_canvas_create........ 82
gx_canvas_delete........ 84
gx_canvas_drawing_complete 85
gx_canvas_drawing_initiate........ 86
gx_canvas_ellipse_draw........ 89
gx_canvas_line_draw 90
gx_canvas_pie_draw........ 91
gx_canvas_offset_set 92
gx_canvas_pixelmap_blend 93
gx_canvas_pixel_draw 95
gx_canvas_pixelmap_draw 97
gx_canvas_pixelmap_tile........ 98
gx_canvas_polygon_draw 99
gx_canvas_rectangle_draw........ 100
gx_canvas_shift........ 101
gx_canvas_text_draw........ 102
gx_checkbox_create........ 104

gx_checkbox_draw 106
gx_checkbox_event_process 107
gx_checkbox_pixelmap_set 108
gx_checkbox_select 110
gx_context_brush_default 111
gx_context_brush_define 117
gx_context_brush_get 121
gx_context_brush_set 122
gx_context_brush_pattern_set 123
gx_context_brush_style_set 124
gx_context_brush_width_set 125
gx_context_color_get 126
gx_context_fill_color_set 128
gx_context_font_get 129
gx_context_font_set 131
gx_context_line_color_set 132
gx_context_pixelmap_get 133
gx_context_pixelmap_set 135
gx_context_raw_brush_define 136
gx_context_raw_fill_color_set 138
gx_context_raw_line_color_set 139
gx_display_color_set 141
gx_display_color_table_set 142
gx_display_create 143
gx_display_delete 145
gx_display_font_table_set 146
gx_display_pixelmap_table_set 147
gx_drop_list_close 148
gx_drop_list_create 149
gx_drop_list_event_process 151
gx_drop_list_open 152
gx_drop_list_pixelmap_set 153
gx_drop_list_popup_get 154
gx_horizontal_list_children_position 155

gx_horizontal_list_create 156
gx_horizontal_list_event_process 158
gx_horizontal_list_selected_index_get 160
gx_horizontal_list_selected_widget_get 161
gx_horizontal_list_selected_set 163
gx_horizontal_list_total_columns_set 164
gx_horizontal_scrollbar_create 165
gx_icon_button_create 168
gx_icon_button_draw 170
gx_icon_button_pixelmap_set 171
gx_icon_create 172
gx_icon_draw 174
gx_icon_event_process 175
gx_icon_pixelmap_set 176
gx_multi_line_text_button_create 177
gx_multi_line_text_button_draw 179
gx_multi_line_text_button_event_process 180
gx_multi_line_text_button_text_id_set 181
gx_multi_line_text_button_text_set 182
gx_multi_line_text_input_create 183
gx_multi_line_text_input_buffer_get 185
gx_multi_line_text_input_style_add 186
gx_multi_line_text_input_style_remove 187
gx_multi_line_text_view_char_index_set 189
gx_multi_line_text_view_create 191
gx_multi_line_text_view_event_process 193
gx_multi_line_text_view_font_set 194
gx_multi_line_text_view_scroll 195
gx_multi_line_text_view_text_color_set 196
gx_multi_line_text_view_text_id_set 197
gx_multi_line_text_view_text_set 198
gx_pixelmap_button_create 199
gx_pixelmap_button_draw 201
gx_pixelmap_button_event_process 202

gx_pixelmap_button_pixelmap_set..204
gx_pixelmap_prompt_create..206
gx_pixelmap_prompt_draw..208
gx_pixelmap_prompt_pixelmap_set..209
gx_pixelmap_slider_create..212
gx_pixelmap_slider_draw..214
gx_pixelmap_slider_event_process..215
gx_pixelmap_slider_pixelmap_set..216
gx_progress_bar_create..218
gx_progress_bar_draw..220
gx_progress_bar_event_process..221
gx_progress_bar_font_set..222
gx_progress_bar_info_set..223
gx_progress_bar_pixelmap_set..224
gx_progress_bar_range_set..225
gx_progress_bar_text_color_set..226
gx_progress_bar_value_set..227
gx_prompt_create..228
gx_prompt_draw..230
gx_prompt_font_set..231
gx_prompt_text_color_set..232
gx_prompt_text_get..234
gx_prompt_text_id_set..235
gx_prompt_text_set..236
gx_radio_button_create..237
gx_radio_button_draw..239
gx_radio_button_pixelmap_set..240
gx_scroll_thumb_create..242
gx_scroll_thumb_draw..244
gx_scroll_thumb_event_process..245
gx_scrollbar_draw..246
gx_scrollbar_event_process..247
gx_scrollbar_limit_check..248
gx_scrollbar_reset..249

gx_single_line_text_input_backspace 251
gx_single_line_text_input_buffer_clear 252
gx_single_line_text_input_buffer_get 253
gx_single_line_text_input_character_delete 255
gx_single_line_text_input_character_insert 256
gx_single_line_text_input_create 257
gx_single_line_text_input_draw 259
gx_single_line_text_input_end 260
gx_single_line_text_input_event_process 261
gx_single_line_text_input_home 263
gx_single_line_text_input_left_arrow 264
gx_single_line_text_input_position_get 265
gx_single_line_text_input_right_arrow 267
gx_single_line_text_input_style_add 268
gx_single_line_text_input_style_remove 269
gx_single_line_text_input_style_set 270
gx_slider_create 272
gx_slider_draw 274
gx_slider_event_process 275
gx_slider_info_set 276
gx_slider_needle_draw 278
gx_slider_needle_position_get 279
gx_slider_tickmarks_draw 281
gx_slider_travel_get 282
gx_slider_value_calculate 284
gx_slider_value_set 286
gx_sprite_create 288
gx_sprite_current_frame_set 290
gx_sprite_frame_list_set 291
gx_sprite_start 292
gx_sprite_stop 293
gx_studio_widget_create 294
gx_studio_named_widget_create 295
gx_studio_display_configure 296

gx_system_active_language_set 298
gx_system_canvas_refresh 299
gx_system_dirty_mark 300
gx_system_dirty_partial_add 301
gx_system_draw_context_get 302
gx_system_event_fold 303
gx_system_event_send 304
gx_system_focus_claim 305
gx_system_initialize 306
gx_system_language_table_get 307
gx_system_language_table_set 308
gx_system_memory_allocator_set 309
gx_system_scroll_appearance_get 310
gx_system_scroll_appearance_set 311
gx_system_start 313
gx_system_string_get 314
gx_system_string_table_get 315
gx_system_string_width_get 317
gx_system_timer_start 319
gx_system_timer_stop 320
gx_system_version_string_get 321
gx_system_widget_find 322
gx_text_button_create 324
gx_text_button_draw 326
gx_text_button_font_set 327
gx_text_button_text_color_set 328
gx_text_button_text_get 330
gx_text_button_text_id_set 331
gx_text_button_text_set 332
gx_utility_ltoa 333
gx_utility_math_cos 334
gx_utility_math_sin 335
gx_utility_math_sqrt 336
gx_utility_pixelmap_rotate 337

gx_utility_pixelmap_simple_rotate 339
gx_utility_rectangle_center 341
gx_utility_rectangle_center_find 342
gx_utility_rectangle_combine 343
gx_utility_rectangle_compare 344
gx_utility_rectangle_define 345
gx_utility_rectangle_overlap_detect 346
gx_utility_rectangle_point_detect 347
gx_utility_rectangle_resize 348
gx_utility_rectangle_shift 349
gx_vertical_list_children_position 350
gx_vertical_list_create 351
gx_vertical_list_event_process 353
gx_vertical_list_selected_index_get 355
gx_vertical_list_selected_widget_get 356
gx_vertical_list_selected_set 357
gx_vertical_list_total_rows_set 358
gx_vertical_scrollbar_create 359
gx_widget_allocate 361
gx_widget_attach 363
gx_widget_background_draw 365
gx_widget_back_attach 366
gx_widget_back_move 368
gx_widget_block_move 370
gx_widget_border_draw 372
gx_widget_border_style_set 374
gx_widget_border_width_get 376
gx_widget_canvas_get 378
gx_widget_child_detect 380
gx_widget_children_draw 382
gx_widget_client_get 384
gx_widget_color_get 386
gx_widget_create 388
gx_widget_created_test 390

gx_widget_delete 391
gx_widget_detach 392
gx_widget_draw 393
gx_widget_draw_set 394
gx_widget_event_generate 395
gx_widget_event_process 397
gx_widget_event_process_set 399
gx_widget_event_to_parent 401
gx_widget_fill_color_set 402
gx_widget_find 404
gx_widget_focus_next 406
gx_widget_focus_previous 407
gx_widget_font_get 408
gx_widget_front_move 409
gx_widget_height_get 411
gx_widget_hide 412
gx_widget_pixelmap_get 413
gx_widget_resize 414
gx_widget_shift 415
gx_widget_show 417
gx_widget_status_add 418
gx_widget_status_get 420
gx_widget_status_remove 422
gx_widget_status_test 424
gx_widget_style_add 426
gx_widget_style_get 428
gx_widget_style_remove 430
gx_widget_style_set 432
gx_widget_width_get 434
gx_window_client_height_get 435
gx_window_client_scroll 436
gx_window_client_width_get 437
gx_window_create 438
gx_window_draw 440

gx_window_event_process 441
gx_window_root_create 442
gx_window_root_delete 444
gx_window_root_event_process 445
gx_window_root_find 446
gx_window_scroll_info_get 447
gx_window_scrollbar_find 449
gx_window_wallpaper_get 450
gx_window_wallpaper_set 451
Chapter 5: GUIX Display Drivers 452
GUIX Example 459
Appendix A: GUIX Pre-Defined Colors 463
Appendix B: GUIX Pre-Defined Color Resource IDs 464
Appendix C: GUIX Color Formats 465
Appendix D: GUIX Widget Styles 466
Appendix E: GUIX Events 469
Appendix F: GUIX RTOS Binding Services 470
Appendix G: GUIX Font Structure 473
Index 476

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.
ℹ	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.
CHAR	Most often a standard 8-bit character 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.

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.

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
    - Button (Bitmap, icon, text)
    - Dialog
    - Drop Down List
    - Keyboard
    - Prompt (and Multi-Line Prompts)
    - Scroll Bar
    - Scrollable List
    - Slider
    - 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

Two types of GUIX packages are available—*standard* and *premium*. The *standard* package includes minimal source code, while the *premium* package contains complete GUIX source code. Either package is shipped on a single CD. The exact contents of the distribution CD depends on the target processor, development tools, and the GUIX package purchased. 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. It is distributed with the *standard* package.

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: If installing the *standard* package, GUIX installation is now complete. Otherwise, if installing the premium package, you must build the GUIX runtime 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. The options below can be defined in the application source, on the command line, or within the ***gx_user.h*** include file.

Review the ***readme_guix_generic.txt*** file for additional options for your specific version of GUIX. The following sections describe configuration options available in GUIX.

System Configuration Options

Define	**Meaning**
GX_THREAD_STACK_SIZE	stack size, in bytes, of the internal GUIX thread created at system startup.
GX_SYSTEM_TIMER_TICKS	defines the GUIX timer rate as a multiple of the ThreadX tick interrupt rate.
GX_TICKS_SECOND	useful definition for application logic, must correlate with the GX_SYSTEM_TIMER_TICKS definition.
GX_CONST	compiler specific definition of the "const" data type, usually just "const".
GX_MAX_ACTIVE_TIMERS	maximum number of simultaneous GUIX timers
GX_MAX_VIEWS	maximum number of simultaneous GUIX views.
GX_UTF8_SUPPORT	this definition enables internal support for UTF8 format string encoding.

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.

```
#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
    - Multithread Safety
    - Periodic Processing
    - Widget Defaults
    - Scrollbar Appearance
    - Skinning
    - System Error Handling
  - GUIX Canvas Component
    - Canvas Creation
    - Canvas Control Block
    - Canvas Alpha Channel
    - Color Depth
    - Transitions
  - GUIX Display Component
    - Display Creation
    - Display Control Block
    - Installing Themes
    - Drawing APIs
    - 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

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.

GUIX assumes the existence of ThreadX and depends on its thread execution, suspension, periodic timers, and mutual exclusion facilities.

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 must be created by the application software, usually as part of the tx_application_define function implementation. The priority of this thread, as well as it’s preemption-threshold value, time-slice selection, and stack size, are determined by the application software.

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_destroy***. 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 50ms via the constant ***GX_SYSTEM_TIMER_FREQUENCY***, which is defined in ***gx_port.h*** by default. 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 ThreadX timer ticks. Typically, a ThreadX timer tick is 10ms, so the default value of ***GX_SYSTEM_TIMER_FREQUENCY*** is 5, yielding 50ms. However, if the ThreadX timer tick represents a different value, the GUIX timer frequency must be adjusted accordingly.

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. Or the drawing functions may only write to a 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.

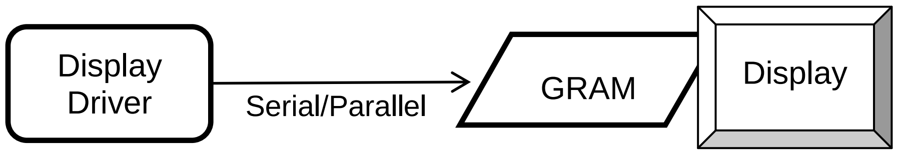
A typical hardware display driver is implemented by first creating the default GUIX display driver for the required color depth and format. Then the hardware driver will replace those functions that need to be optimized or customized for the particular hardware implementation.

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

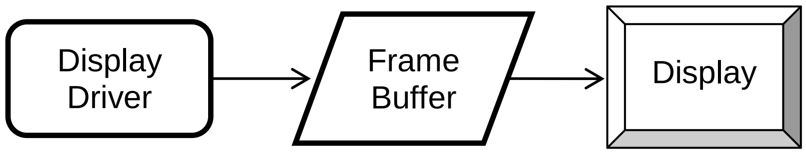
Various hardware targets and displays utilize a variety of different display memory architectures, depending on the memory constraints of the target and the functionality requirements of the application. We will outline some of the common memory architectures here with a brief description of each.

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



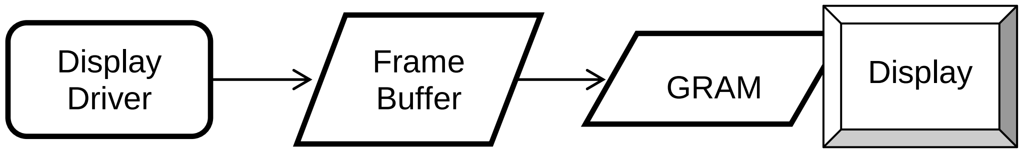
In the model above, no memory for a frame buffer exists in memory local to the CPU. All graphics data is stored in an external GRAM which is incorporated into the display itself. The interface to the external GRAM can be parallel or serial. This type of architecture is very low cost; however it can exhibit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

Model 2) One local frame buffer:



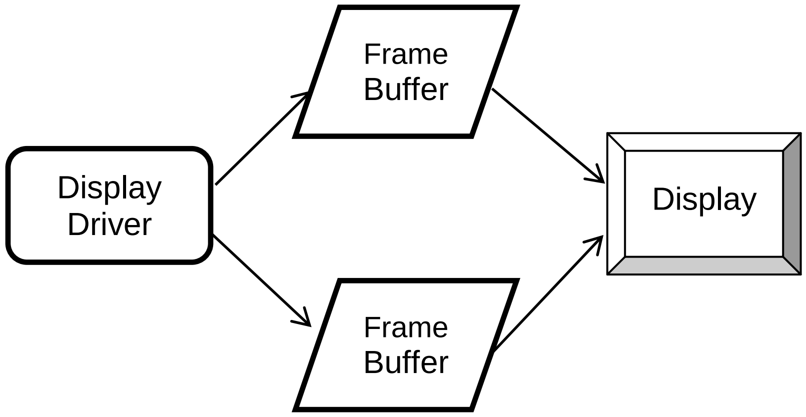
In this model, memory for the graphics data is allocated from a random-access memory that is directly accessible the CPU. Dedicated hardware must be present to repeatedly transmit the graphics data (along with timing signals) from the local memory to the display. This model differs from model 1 in that 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:



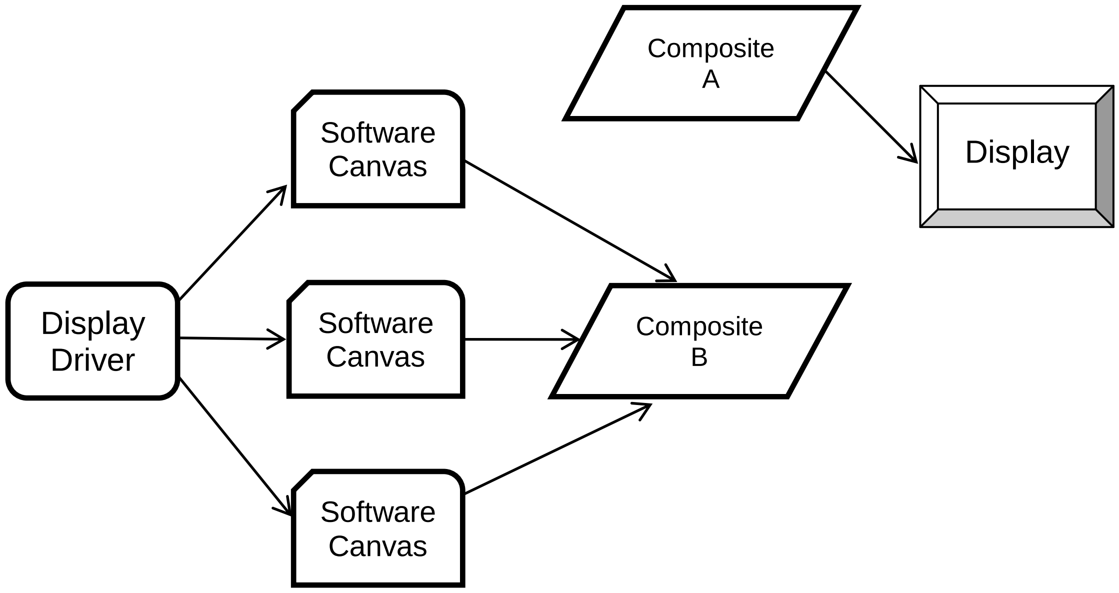
Model 3 is a combination of the first two. In this model, sufficient local memory exists to hold one frame buffer. In addition, the display device provides an external GRAM and automatically refreshes itself using the data provided in the GRAM. This architecture benefits from improved update efficiency because we can transfer the modified portion of the local frame buffer to the external GRAM in one block transfer, often utilizing onboard 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:



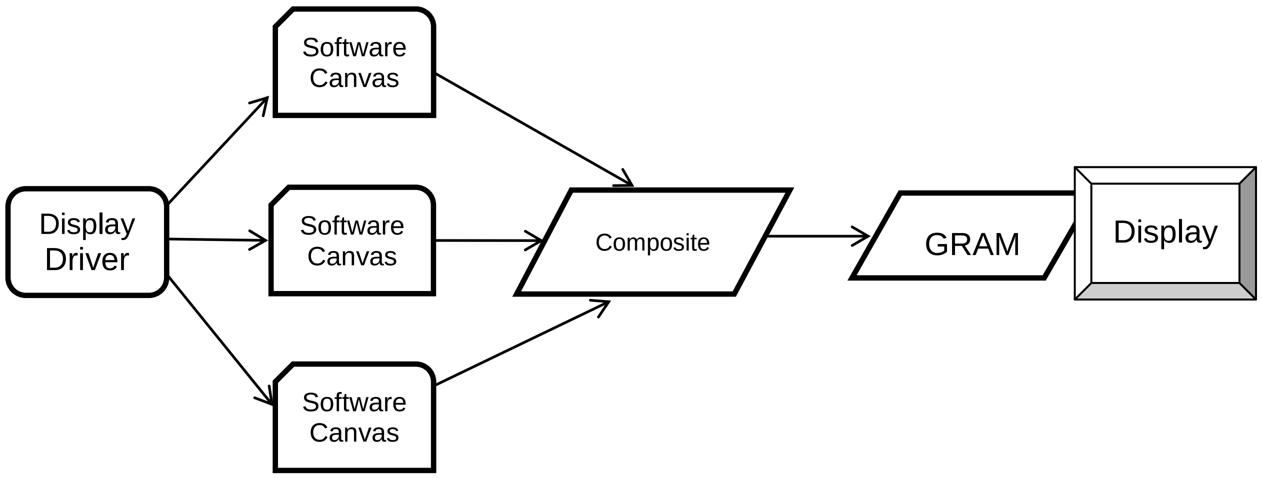
In model 4, sufficient memory is present to provide two local frame buffers. In this case, GUIX treats one frame buffer as the active frame buffer, and the other as the working frame buffer. When a display update or drawing operation is in progress, it takes place in the working buffer. When the drawing 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.

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:

_gx_system_thread_stack[GX_THREAD_STACK_SIZE];

The size of the GUIX thread stack is defined in ***gx_port.h***, but may be overridden by the application by changing ***gx_user.h*** or via project options or command line parameters. The stack size must be large enough to handle the worst-case event handling and drawing nested 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.

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 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. GX_DISPLAY is implemented to correctly render graphics on each canvas using that canvas' required color format. The 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 GX_WIDGET type.
GX_UTILITY:	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.

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 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 proviced by the target system. Appendix F, ***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.

Initialization

GUIX initialization is accomplished by the application calling the service ***gx_system_initialize***, which may be called by the application from the ThreadX ***tx_application_define*** routine (initialization context) or from application threads. The ***gx_system_initialize*** function initializes all global GUIX data structures and creates the GUIX system mutex, event queue, timer, and thread. Once ***gx_system_initialize*** returns, the application can use GUIX.

String Table Management

The GUIX string table and string resources are maintained by the GUIX system component. The other GUIX resource types (colors, fonts, and pixelmaps) are 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 system 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 system string table is a multi-column table of pointers to UTF8 encoded 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.

GUIX optionally supports UTF8 format string encoding. Support for UTF8 string encoding is enabled by defining the value GX_UTF8_SUPPORT in the gx_user.h header file. If this definition is not enabled, GUIX will internally use only standard 8-bit ASCII plus Latin-1 code page character encoding.

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.

The active string table is assigned by calling the gx_system_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_system_language_set() API function. This function determines which column of the system 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_system_string_get() API function. This function returns the string associated with a given string ID and the currently active system language.

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.

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.

Periodic Processing

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***.

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:

```
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 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, color alpha which specifies the blending ratio for a brush, 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.

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 blending is provided.

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.

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.

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.

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

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 ***GX_DISPLAY*** and are defined in ***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.

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 ***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 **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:

GX_COLOR_ID_CANVAS	Default canvas (i.e. display background) color
GX_COLOR_ID_WIDGET_FILL	Default widget fill color
GX_COLOR_ID_WINDOW_FILL	Default window fill color
GX_COLOR_ID_DEFAULT_BORDER	Default widget border color
GX_COLOR_ID_WINDOW_BORDER	Default window border color
GX_COLOR_ID_TEXT	Default text color
GX_COLOR_ID_SELECTED_TEXT	Default selected text color
GX_COLOR_ID_SELECTED_TEXT_FILL	Default selected text fill color
GX_COLOR_ID_SCROLL_FILL	Scrollbar fill color
GX_COLOR_ID_SCROLL_BUTTON	Scrollbar button fill color
GX_COLOR_ID_SHADOW	Default shadow color
GX_COLOR_ID_SHINE	Default highlight color
GX_COLOR_ID_BUTTON_BORDER	Button widget border color
GX_COLOR_ID_BUTTON_UPPER	Button widget upper fill color
GX_COLOR_ID_BUTTON_LOWER	Button widget lower fill color
GX_COLOR_ID_BUTTON_TEXT	Button widget text color
GX_COLOR_ID_TEXT_INPUT_TEXT	Text input widget text color
GX_COLOR_ID_TEXT_INPUT_FILL	Text input fill color
GX_COLOR_ID_SLIDER_GROOVE_TOP	Color used to draw slider groove.
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 hightlight
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***.

```
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().

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 top-level 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 primitives are defined as follows:

Hardware Initialization
Draw basic line
Draw horizontal line
Draw vertical line
Draw Rectangle
Draw Quadragon
Draw Polygon
Draw Text
Draw pixelmap
Tile pixelmap
Draw pixel
Blend pixel
Buffer Toggle
Canvas copy
Canvas blend
Block Move

The above application level drawing APIs don't do actual drawing to the canvas, but instead verify the caller's parameters before invoking the driver level drawing function. The driver level drawing function actually writes pixel data into the canvas memory. GUIX provides stock driver-level drawing functions for various color depths, including 1, 2, 4, 8, 16, 24, and 32 bits per pixel (bpp). Alternatively, the underlying hardware driver can utilize its own drawing functions, which is useful in cases where the hardware has the needed drawing primitive. The following are the stock driver level drawing functions provided by GUIX:

```
VOID        gx_display_driver_XXBPP_canvas_copy(GX_CANVAS *source,
                    GX_CANVAS *destination);
VOID        gx_display_driver_XXBPP_canvas_blend(GX_CANVAS *source,
                    GX_CANVAS *dest);
VOID        gx_display_driver_XXBPP_horizontal_line_draw(GX_DRAW_CONTEXT *context,
                    INT x_start, INT x_end, INT y_position, INT width, GX_COLOR color);
VOID        gx_display_driver_XXBPP_line_draw(GX_DRAW_CONTEXT *context,
                    INT x_start, INT y_start, INT x_end, INT y_end);
VOID        gx_display_driver_XXBPP_line_anti_aliasing(GX_DRAW_CONTEXT *context,
                    INT x_start, INT y_start, INT x_end, INT y_end);
VOID        gx_display_driver_XXBPP_pixel_blend(GX_DRAW_CONTEXT *context,
                    INT x_position, INT y_position, GX_COLOR color, GX_UBYTE alpha);
VOID        gx_display_driver_XXBPP_pixel_write(GX_DRAW_CONTEXT *context,
                    INT x_position, INT y_position, GX_COLOR color);
VOID        gx_display_driver_XXBPP_vertical_line_draw(GX_DRAW_CONTEXT *context,
                    INT y_start, INT y_end, INT x_position, INT width, GX_COLOR color);
```

```
VOID            gx_display_driver_XXBPP_pixelmap_draw(GX_DRAW_CONTEXT *context,
                        INT x_start, INT y_start, GX_PIXELMAP *pixelmap);
VOID            gx_display_driver_XXBPP_block_move(GX_DRAW_CONTEXT *context,
                        GX_RECTANGLE *source, INT x_shift, INT y_shift);
VOID            gx_display_driver_XXBPP_round_end_draw(GX_DRAW_CONTEXT *context,
                        INT x_start, INT y_start, INT x_end, INT y_end, INT round);
GX_COLOR        gx_display_driver_XXBPP_native_color_get(GX_COLOR raw_color);
VOID            gx_display_driver_XXBPP_one_width_line_draw(GX_DRAW_CONTEXT *context,
                        INT x_start, INT y_start, INT x_end, INT y_end);
VOID            gx_display_driver_XXBPP_polygon_draw(GX_DRAW_CONTEXT *context,
                        INT points, GX_POINT *coordinates);
```

Where the “XXBPP” is replaced by the bits-per-pixel of the driver implementation, e.g., 1, 2, 4, 8, 16, 24, or 32 bits per pixel (BPP).

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 dispaly 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_DYNAMICALLY_ALLOCATED 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_DYNAMICALLY_ALLOCATED 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
GX_TYPE_RADIO_BUTTON
GX_TYPE_CHECKBOX
GX_TYPE_PIXELMAP_BUTTON
GX_TYPE_SHADOW_BUTTON
GX_TYPE_ICON_BUTTON
GX_TYPE_ICON
GX_TYPE_SPRITE

GX_TYPE_SLIDER
GX_TYPE_PIXELMAP_SLIDER
GX_TYPE_VERTICAL_SCROLL
GX_TYPE_HORIZONTAL_SCROLL
GX_TYPE_PROGRESS_BAR

GX_TYPE_PROMPT
GX_TYPE_PIXELMAP_PROMPT

GX_TYPE_SINGLE_LINE_TEXT_INPUT
GX_TYPE_PIXELMAP_TEXT_INPUT
GX_TYPE_MULTI_LINE_TEXT_VIEW
GX_TYPE_MULTI_LINE_TEXT_INPUT

GX_TYPE_WINDOW
GX_TYPE_ROOT_WINDOW

GX_TYPE_VERTICAL_LIST
GX_TYPE_HORIZONTAL_LIST
GX_TYPE_DROPLIST
GX_TYPE_DIALOG
GX_TYPE_KEYBOARD

Styles

Widget styles consist of things like border properties (raised, recessed, thin, thick, or no boarder 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, as described in that particular widget's event notification documentation.

The following is a list of pre-defined GUIX event types:

GX_EVENT_ANIMATION_COMPLETE
GX_EVENT_CLICKED
GX_EVENT_CLOSE
GX_EVENT_DESELECTED
GX_EVENT_DESTROY
GX_EVENT_FOCUS_GAINED
GX_EVENT_FOCUS_LOST
GX_EVENT_HIDE
GX_EVENT_HORIZONTAL_FLICK
GX_EVENT_HORIZONTAL_SCROLL
GX_EVENT_KEY_DOWN
GX_EVENT_KEY_PRESS
GX_EVENT_KEY_RELEASE
GX_EVENT_KEY_UP
GX_EVENT_LANGUAGE_CHANGE
GX_EVENT_LIST_SELECT
GX_EVENT_MOVE
GX_EVENT_PARENT_SIZED
GX_EVENT_PEN_DOWN
GX_EVENT_PEN_DRAG
GX_EVENT_PEN_UP
GX_EVENT_RADIO_DESELECT
GX_EVENT_RADIO_SELECT
GX_EVENT_REDRAW
GX_EVENT_RESIZE
GX_EVENT_RESOURCE_CHANGE
GX_EVENT_SELECTED
GX_EVENT_SHOW
GX_EVENT_SLIDE
GX_EVENT_SLIDER_VALUE
GX_EVENT_SPRITE_COMPLETE
GX_EVENT_TERMINATE
GX_EVENT_TEXT_EDIT
GX_EVENT_TEXT_EDIT_COMPLETE
GX_EVENT_TEXT_EDITED
GX_EVENT_TIMER
GX_EVENT_TOGGLE_OFF
GX_EVENT_TOGGLE_ON
GX_EVENT_VERTICAL_FLICK
GX_EVENT_VERTICAL_SCROLL
GX_EVENT_ZOOM_IN
GX_EVENT_ZOOM_OUT

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.

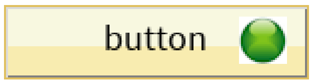
Application drawing functions can take advantage (i.e., not duplicate the coding) of the default drawing by simply calling it directly from the overridden drawing function.

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.

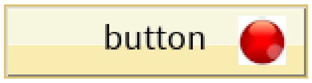
Implementing Custom Drawing (example)

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:



custom button “on”



custom button “off”

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

```
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:

```
typedef struct GX_BRUSH_STRUCT
{
    GX_PIXELMAP                  *gx_brush_pixelmap;
    GX_FONT                      *gx_brush_font;
    ULONG                        gx_brush_line_pattern;
    UINT                            gx_brush_style;
    UINT                            gx_brush_width;
    GX_COLOR                        gx_brush_fill_color;
    GX_COLOR                        gx_brush_line_color;
} GX_BRUSH;
```

The ***gx_brush_pixelmap*** defines a pixelmap to use for rectangle and polygon fills. This member is not used unless the ***gx_brush_style*** is 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
GX_BRUSH_PATTERN

The ***gx_brush_width*** member defines the line with 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 border-less 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 defalt 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 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.

To support fade in and fade out type animations, the GUIX Animation component requires that a canvas be created that is used strictly for the animation effect. 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 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_UBYTE       gx_animation_type;
    GX_UBYTE       gx_animation_start_alpha;
    GX_UBYTE       gx_animation_end_alpha;
    GX_UBYTE       gx_animation_steps;
    GX_POINT       gx_animation_start_position;
    GX_POINT       gx_animation_end_position;
    GX_VALUE       gx_animation_delay;

} GX_ANIMATION_INFO;
```

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_type** member defines the type of animation to be performed. Currently supported types include GX_ANIMATION_FADE_IN, GX_ANIMATION_FADE_OUT, GX_ANIMATION_SLIDE_IN, GX_ANIMATION_SLIDE_OUT.

The **gx_animation_start_alpha** field defines the starting canvas alpha value for fade type animations.

The **gx_animation_end_alpha** field defines the ending canvas alpha value for fade type animations.

The **gx_animation_steps** field defines how many steps the controller should execute between the start and end position or alpha value.

The **gx_animation_start_position** defines the top-left starting point for the target widget for slide type animations.

The **gx_animation_end_position** defines the top-left ending position for the target widget for slide type animations.

The **gx_animation_delay** 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.

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 target 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:

gx_pixelmap_transparent_detect
gx_utlity_ltoa
gx_utility_math_cos
gx_utility_math_sin
gx_utility_math_sqrt
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_shift

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.

GUIX Service	**Description**
gx_animation_canvas_define	Provide memory to an animation controller for a canvas to be used for subsequent animations.
gx_animation_create	Create an animation controller
gx_animation_delete	Delete an animation controller
gx_animation_start	Initiate an animation sequence
gx_animation_stop	Suspend an animation sequence
gx_brush_default	Initialize current brush to defaults
gx_brush_define	Define brush
gx_button_background_draw	Draw button background
gx_button_create	Create button
gx_button_deselect	Deselect button
gx_button_draw	Draw button
gx_button_event_process	Process button event
gx_button_select	Select button
gx_canvas_alpha_set	Set alpha-blend value for canvas
gx_canvas_arc_draw	Draw circle arc
gx_canvas_block_move	Move block
gx_canvas_circle_draw	Draw circle
gx_canvas_create	Create a canvas
gx_canvas_delete	Delete a canvas
gx_canvas_drawing_complete	Complete canvas drawing
gx_canvas_drawing_initiate	Initiate drawing on canvas
gx_canvas_line_draw	Draw line
gx_canvas_offset_set	Assign canvas x,y display offset
gx_canvas_pixel_draw	Draw a single pixel
gx_canvas_pixelmap_blend	Blend a pixelmap with background
gx_canvas_pixelmap_draw	Draw pixelmap
gx_canvas_pixelmap_tile	Tile pixelmap
gx_canvas_polygon_draw	Draw polygon
gx_canvas_rectangle_draw	Draw rectangle
gx_canvas_shift	Shift canvas by x,y
gx_canvas_text_draw	Draw text
gx_checkbox_create	Create a checkbox

gx_checkbox_draw	Draw a checkbox
gx_checkbox_event_process	Checkbox event process function
gx_checkbox_pixelmap_set	Assign checkbox pixelmap
gx_checkbox_select	Select checkbox
gx_context_brush_default	Set the brush of current context
gx_context_brush_define	Define brush of current context
gx_context_brush_get	Get brush of current context
gx_context_brush_pattern_set	Set pattern of the brush of current context
gx_context_brush_set	Set brush of current context
gx_context_brush_style_set	Set brush style of current context
gx_context_brush_width_set	Set brush width of current ontext
gx_context_color_get	Resolve a color ID to color value
gx_context_fill_color_set	Set fill color of current context
gx_context_font_get	Resolve a font ID to font pointer value
gx_context_font_set	Set font of current context
gx_context_line_color_set	Set line color of current context
gx_context_pixelmap_get	Resolve a pixelmap ID to pixelmap pointer value
gx_context_pixelmap_set	Assign brush pixelmap, used for area fills
gx_context_raw_brush_define	Define raw brush of current context
gx_context_raw_fill_color_set	Set raw fill color of current context
gx_context_raw_line_color_set	Set raw line color of current context
gx_display_color_set	Replace one color value in display color table.
gx_display_color_table_set	Assign the color table used by a display
gx_display_create	Create display
gx_display_delete	Delete display
gx_display_font_table_set	Assign the font table used by a display
gx_display_pixelmap_table_set	Assign the pixelmap table used by a display
gx_drop_list_close	Close drop list
gx_drop_list_create	Create drop list
gx_drop_list_open	Open drop list
gx_drop_list_pixelmap_set	Set pixelmap to drop list
gx_drop_list_popup_set	Set popup to drop list
gx_horizontal_list_children_position	Position children in horizontal list
gx_horizontal_list_create	Create horizontal list
gx_horizontal_list_event_process	Process event in horizontal list
gx_horizontal_list_selected_index_get	Get the selected item index
gx_horizontal_list_selected_widget_get	Get the selected item widget
gx_horizontal_list_selected_set	Set the selected item
gx_horizontal_list_total_columns_set	Change number of list columns after creation
gx_horizontal_scrollbar_create	Create horizontal scrollbar
gx_icon_button_create	Create icon button
gx_icon_button_draw	Draw an icon button

gx_icon_button_pixelmap_set	Set pixelmap in icon button
gx_icon_create	Create icon
gx_icon_draw	Draw icon
gx_icon_event_process	Icon event processing function
gx_icon_pixelmap_set	Set pixelmap for icon
gx_multi_line_text_button_create	Create multi-line text button
gx_multi_line_text_button_draw	Draw multi-line text button
gx_multi_line_text_button_event_process	Set font for multi-line text button
gx_multi_line_text_button_text_id_set	Set system string to text button
gx_multi_line_text_button_text_set	Set user-defined string to text button
gx_multi_line_text_input_create	Create multi-line text input
gx_multi_line_text_input_style_add	Add cursor style flags
gx_multi_line_text_input_style_remove	Remove cursor style flags
gx_multi_line_text_input_style_set	Assign cursor style flags
gx_multi_line_text_view_create	Create multi-line text view
gx_multi_line_text_view_event_process	Process multi-line text view event
gx_multi_line_text_view_font_set	Set font used in multi line text view
gx_multi_line_text_view_scroll	Scroll multi-line text view
gx_multi_line_text_view_text_color_set	Set text color in mulit line text view
gx_multi_line_text_view_text_id_set	Set system text string in multi line text view
gx_multi_line_text_view_text_set	Set user-defined string in multi line text view
gx_pixelmap_button_create	Create pixelmap button
gx_pixelmap_button_draw	Draw pixelmap button
gx_pixelmap_button_event_process	Pixelmap button event processing
gx_pixelmap_button_pixelmap_set	Set pixelmap in pixelmap button
gx_pixelmap_prompt_create	Create pixelmap prompt
gx_pixelmap_prompt_draw	Draw pixelmap prompt
gx_pixelmap_prompt_pixelmap_set	Set pixelmap in pixelmap prompt
gx_pixelmap_slider_create	Create pixelmap slider
gx_pixelmap_slider_draw	Draw pixelmap slider
gx_pixelmap_slider_event_process	Pixelmap slider event processing
gx_pixelmap_slider_pixelmap_set	Set pixelmap in pixelmap slider
gx_progress_bar_create	Create progress bar
gx_progress_bar_draw	Draw a progress bar
gx_progress_bar_event_process	Progress bar event processing
gx_progress_bar_font_set	Set progress bar font
gx_progress_bar_info_set	Set progress bar range and value
gx_progress_bar_pixelmap_set	Assign progress bar pixelmap
gx_progress_bar_range_set	Set progress bar range
gx_progress_bar_text_color_set	Assign progress bar text color
gx_progress_bar_value_set	Assign progress bar value
gx_prompt_create	Create prompt
gx_prompt_draw	Draw prompt

gx_prompt_font_set	Set prompt font
gx_prompt_text_color_set	Set prompt text color
gx_prompt_text_get	Get prompt text
gx_prompt_text_id_set	Set prompt with system text string
gx_prompt_text_set	Set prompt text
gx_radio_button_create	Create radio button
gx_radio_button_draw	Draw radio button
gx_radio_button_pixelmap_set	Set pixelmap in radio button
gx_scroll_thumb_create	Create scroll thumb
gx_scroll_thumb_draw	Draw scroll thumb
gx_scroll_thumb_event_process	Process scroll thumb event
gx_scrollbar_draw	Draw scrollbar
gx_scrollbar_event_process	Process scrollbar event
gx_scrollbar_limit_check	Check scrollbar limit
gx_scrollbar_reset	Reset scrollbar
gx_single_line_text_input_backspace	Handle backspace character
gx_single_line_text_input_buffer_clear	Clear the character buffer
gx_single_line_text_input_character_delete	Delete character
gx_single_line_text_input_character_insert	Insert character
gx_single_line_text_input_create	Create single-line text input
gx_single_line_text_input_draw	Draw single-line text input widget
gx_single_line_text_input_end	Move cursor to end
gx_single_line_text_input_event_process	Text input event processing
gx_single_line_text_input_home	Move cursor to home
gx_single_line_text_input_left_arrow	Handle left arrow key
gx_single_line_text_input_position_get	Get cursor position
gx_single_line_text_input_right_arrow	Handle right arrow key
gx_single_line_text_input_style_add	Add style flags
gx_single_line_text_input_style_remove	Remove style flags
gx_single_line_text_input_style_set	Assign style flags
gx_slider_create	Create slider
gx_slider_draw	Draw slider
gx_slider_event_process	Process slider event
gx_slider_info_set	Set slider information block
gx_slider_needle_draw	Draw slider needle
gx_slider_needle_position_get	Get slider needle position
gx_slider_tickmarks_draw	Draw slider tickmarks
gx_slider_travel_get	Get slider travel
gx_slider_value_calculate	Calculate slider value
gx_slider_value_set	Set slider value
gx_sprite_create	Create GX_SPRITE widget
gx_sprite_current_frame_set	Assign current display frame for sprite widget
gx_sprite_frame_list_set	Assign or modify a sprite frame list
gx_sprite_start	Start a sprite sequence
gx_sprite_stop	Stop a sprite sequence
gx_studio_widget_create	Create widget defined within Studio
gx_studio_named_widget_create	Create screen defined within Studio

gx_studio_display_configure	Create and initialize GX_DISPLAY, GX_CANVAS, and GX_WINDOW_ROOT
gx_system_active_language_set	Assign active language ID
gx_system_canvas_refresh	Force refresh (drawing) of dirty canvases
gx_system_dirty_mark	Mark area dirty
gx_system_dirty_partial_add	Mark partial area dirty
gx_system_draw_context_get	Get drawing context
gx_system_event_fold	Foldevent
gx_system_event_send	Send event
gx_system_focus_claim	Claim focus
gx_system_initialize	Initialize GUIX
gx_system_language_table_get	Retrieve language table
gx_system_language_table_set	Assign language table
gx_system_memory_allocator_set	Assign memory allocator/de-allocator
gx_system_scroll_appearance_get	Get scroll appearance
gx_system_scroll_appearance_set	Set scroll appearance
gx_system_start	Start GUIX
gx_system_string_get	Get string
gx_system_string_table_get	Get string table
gx_system_string_table_set	Set string table
gx_system_string_width_get	Get string width
gx_system_theme_install	Install font/color/pixelmap tables
gx_system_timer_start	Start timer
gx_system_timer_stop	Stop timer
gx_system_version_string_get	Retrieve GUIX library version string
gx_system_widget_find	Find widget
gx_text_button_create	Create text button
gx_text_button_draw	Draw text button
gx_text_button_font_set	Set font for text button
gx_text_button_text_color_set	Set text button color
gx_text_button_text_get	Get text used in text button
gx_text_button_text_id_set	Set system string to text button
gx_text_button_text_set	Set user-defined string to text button
gx_transition_window_create	Create a transition window
gx_utility_ltoa	Convert long integer to ASCII
gx_utility_math_cos	Compute cosine
gx_utility_math_sin	Compute sine
gx_utility_math_sqrt	Compute square root
gx_utility_rectangle_center	Center rectangle inside another rectangle
gx_utility_rectangle_center_find	Find center of rectangle
gx_utility_rectangle_combine	Combine two rectangles into first
gx_utility_rectangle_compare	Compare two rectangles
gx_utility_rectangle_define	Define rectangle
gx_utility_rectangle_resize	Resize rectangle
gx_utility_rectangle_overlap_detect	Detect overlap of rectangles
gx_utility_rectangle_point_detect	Detect if point resides in rectangle
gx_utility_rectangle_shift	Shift rectangle

gx_vertical_list_children_position	Position children in vertical list
gx_vertical_list_create	Create vertical list
gx_vertical_list_event_process	Process vertical list event
gx_vertical_list_selected_index_get	Get selected item index
gx_vertical_list_selected_widget_get	Get selected widget
gx_vertical_list_selected_set	Set entry in vertical list
gx_vertical_list_total_rows_set	Change number of list rows after creation
gx_vertical_scrollbar_create	Create vertical scrollbar
gx_widget_allocate	Dynamically allocate a widget
gx_widget_attach	Attach widget to parent
gx_widget_background_draw	Draw widget background
gx_widget_back_attach	Attach widget in back
gx_widget_back_move	Move widget to back
gx_widget_block_move	Move block of pixels
gx_widget_border_draw	Draw widget border
gx_widget_border_style_set	Set widget border style
gx_widget_border_width_get	Set widget border width
gx_widget_canvas_get	Get widget canvas
gx_widget_child_detect	Detect widget child
gx_widget_children_draw	Draw widget children
gx_widget_client_get	Get widget client area
gx_widget_color_get	Resolve color ID to color value for a visible widget
gx_widget_create	Create widget
gx_widget_created_test	Test if widget created
gx_widget_delete	Delete widget
gx_widget_detach	Detach widget from parent
gx_widget_draw	Draw widget
gx_widget_draw_set	Set draw function of widget
gx_widget_event_generate	Generate widget event
gx_widget_event_process	Process widget event
gx_widget_event_process_set	Set event processing function of widget
gx_widget_event_to_parent	Send event to widget's parent
gx_widget_fill_color_set	Assign widget fill color
gx_widget_find	Find widget
gx_widget_focus_next	Move input focus to next widget
gx_widget_focus_previous	Move input focus to previous widget
gx_widget_font_get	Resolve font ID to a font pointer for a visible widget
gx_widget_free	Free widget control block memory
gx_widget_front_move	Move widget to front
gx_widget_height_get	Get widget height
gx_widget_hide	Hide widget
gx_widget_pixelmap_get	Resolve pixelmap ID to a pixelmap pointer for a visible widget
gx_widget_resize	Resize widget
gx_widget_shift	Shift widget
gx_widget_show	Show widget
gx_widget_status_add	Add widget status
gx_widget_status_get	Retrieve widget status flags

gx_widget_status_remove	Remove widget status
gx_widget_status_test	Test widget status
gx_widget_style_add	Add widget style
gx_widget_style_get	Retrieve widget style flags
gx_widget_style_remove	Remove widget style
gx_widget_style_set	Set widget style
gx_widget_text_draw	Draw text assigned to widget
gx_widget_text_id_draw	Draw text assigned to widget
gx_widget_type_find	Find widget type
gx_widget_width_get	Get widget width
gx_window_canvas_set	Set window canvas
gx_window_client_height_get	Get window client height
gx_window_client_scroll	Scroll window clients
gx_window_client_width_get	Get window client width
gx_window_create	Create window
gx_window_draw	Draw window
gx_window_event_process	Process window event
gx_window_root_create	Create root window
gx_window_root_delete	Destroy root window
gx_window_root_event_process	Process event for root window
gx_window_root_find	Find root window
gx_window_scroll_info_get	Get window scroll info
gx_window_scrollbar_find	Find window scrollbar
gx_window_wallpaper_get	Get window wallpaper
gx_window_wallpaper_set	Set window wallpaper

gx_animation_canvas_define

Provide canvas memory to an animation controller

Prototype

```
UINT  gx_animation_canvas_define(GX_ANIMATION *animation,
      GX_VALUE width, GX_VALUE height, GX_DISPLAY *display,
      GX_COLOR *memory, ULONG memory_size);
```

Description

This service provides memory to an animation controller for creating a canvas and root window used to implement the animation sequence. This service creates a new canvas and new root window using the memory and dimensions provided.

Parameters

animation	Pointer to animation control block
width	Width of the animation canvas, in pixels
height	Height of the animation canvas, in pixels
display	The display driver to be used to paint on the animation canvas.
memory	The canvas memory area
memory_size	The size of the provide memory area, in bytes

Return Values

GX_SUCCESS	(0x00)	Successful
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

```
gx_animation_canvas_define(animation, width, height, display, memory, memsize)

/* If status is GX_SUCCESS the new canvas was successfully created and initialized. */
```

See Also

gx_animation_create, gx_animation_delete, gx_animation_start,
gx_animation_stop

gx_animation_create

Create an animation controller

Prototype

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

Description

This service creates an animation controller. The controller is initialized to the idle state.

Parameters

animation	Pointer to animation control block

Return Values

GX_SUCCESS	(0x00)	Successful
GX_ALREADY_CREATED	(0x13)	Control block already initialized
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
GX_ANIMATION animation;

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_start, gx_animation_stop

gx_animation_delete

Delete an animation controller

Prototype

```
UINT  gx_animation_delete(GX_ANIMATION *animation);
```

Description

This service deletes an animation controller. If the controller has created a canvas and root window, the canvas and root window are also deleted.

Parameters

animation	Pointer to animation control block

Return Values

GX_SUCCESS	(0x00)	Successful
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
GX_ANIMATION animation;

gx_animation_delete(&animation);

/* If status is GX_SUCCESS the new animation controller was successfully deleted. */
```

See Also

gx_animation_canvas_define, gx_animation_create, gx_animation_start, gx_animation_stop

gx_animation_start

Start an animation

Prototype

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

Description

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

Parameters

animation	Pointer to animation control block
params	Pointer to parameter structure

Return Values

GX_SUCCESS	(0x00)	Successful
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_VALUE	**(0x22)**	**Invalid parameter**

Allowed From

Initialization and threads

Example

```
GX_ANIMATION_INFO params;

/* 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_type = GX_ANIMATION_SLIDE_OUT;
params.gx_animation_target = &my_window;
params.gx_animation_parent = root_window;
params.gx_animation_start_alpha = 255;
params.gx_animation_steps = 10;
params.gx_animation_delay = 2;

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_stop

gx_animation_stop

Stop an active animation

Prototype

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

Description

Stop a previously started animation.

Parameters

animation	Pointer to animation control block

Return Values

GX_SUCCESS	(0x00)	Successful
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_STATUS	**(0x26)**	**Invalid controller status**

Allowed From

Initialization and threads

Example

```
GX_ANIMATION animation;

gx_animation_stop(&animation);

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

See Also

gx_animation_canvas_define, gx_animation_create, gx_animation_start

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

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

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

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
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

```
UINT  gx_button_background_draw(GX_BUTTON *button);
```

Description

This service draws the button background. This service 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

GX_SUCCESS	(0x00)	Successful button
background draw		
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

```
custom_button_draw()
{
        /* Draw button background. */
        status =  gx_button_background_draw(&my_stop_button);
        /* If status is GX_SUCCESS the stop button background was successfully drawn. */
        /* Add your custom drawing here */
}
```

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

button	Pointer to button control block
name	Logical name of button
parent	Pointer to parent widget of the button
style	Button style. **Appendix D** contains pre-defined general styles for all widgets as well as widget specific styles.
button_id	Application-defined ID of the button
size	Size of the button

Return Values

GX_SUCCESS	(0x00)	Successful button creation
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 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);
```

Description

This service deselects the specified button.

Parameters

button	Pointer to button control block

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

Initialization and threads

Example

```
/* Deselect button. */
status =  gx_button_deselect(&my_stop_button);

/* 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

Draw button

Prototype

```
UINT  gx_button_draw(GX_BUTTON *button);
```

Description

This service draws the specified button.

Parameters

button	Pointer to button control block

Return Values

GX_SUCCESS	(0x00)	Successful button draw
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

```
/* Draw button. */
status =  gx_button_draw(&my_stop_button);

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

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

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

Description

This service processes an event for the specified button.

Parameters

button	Pointer to button control block
event_ptr	Pointer to event to process

Return Values

GX_SUCCESS	(0x00)	Successful button 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

```
/* Process event for button. */
status =  gx_button_event_process(&my_stop_button, &my_event);

/* If status is GX_SUCCESS the stop button event was successfully processed. */
```

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.

Parameters

button	Pointer to button control block

Return Values

GX_SUCCESS	(0x00)	Successful button select
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and 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

```
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 opague). Canvas alpha values are used when an application creates a canvas with GX_CANVAS_COMPOSITE style, into which all other managed canvases are composited prior to final display. This API is only supported when using a screen driver of 16-bpp or higher color depth.

Parameters

canvas	Pointer to canvas control block
alpha	Alpha-blend value can range from 0 (transparent) to 255 (opague).

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

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_arc_draw

Draw arc

Prototype

```
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.

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_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

```
/* 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

Fixme: not sure whether the following included APIs are right.

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

Prototype

```
UINT  gx_canvas_block_move(GX_WIDGET *caller,
                           GX_RECTANGLE *block,
                           GX_VALUE x_shift, GX_VALUE y_shift,
                           GX_BOOL mark_dirty);
```

Description

This service moves a block of the canvas according to the amount specified.

Parameters

widget	Pointer to widget control block
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
mark_dirty	If set, the block is marked as dirty

Return Values

GX_SUCCESS	(0x00)	Successful block move
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_COORDINATE	(0x21)	Invalid coordinates to move to
GX_INVALID_VALUE	(0x22)	Invalid shift value(s)

Allowed From

Initialization and threads

Example

```
/* Move block of “my_radio_button”. */
status =  gx_canvas_block_move(&my_radio_button, &area, 10, 15, GX_TRUE);

/* If status is GX_SUCCESS the block in “my_radio_button” has been moved. */
```

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

```
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.

Parameters

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

Return Values

GX_SUCCESS	(0x00)	Successful circle draw
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

```
/* 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

Fixme: not sure whether the following included APIs are right.

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 canvasThe 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	Width in pixels
height	Height in pixels
memory_area	Memory area for canvas
memory_size	Size of memory area in bytes

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
GX_INVALID_WIDGET_SIZE	(0x19)	Invalid widget size

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,
                 default_canvas_memory, sizeof(default_canvas_memory));


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

See Also

gx_canvas_alpha_set, gx_canvas_delete, gx_canvas_drawing_complete, gx_canvas_drawing_initiate, gx_canvas_offset_set, gx_canvas_shift

gx_canvas_delete

Delete canvas

Prototype

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

Description

This service deletes the canvas.

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

Allowed From

Initialization and threads

Example

```
/* 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,

gx_canvas_drawing_complete

Complete canvas drawing

Prototype

```
UINT  gx_canvas_drawing_complete(GX_CANVAS *canvas,
                                 GX_BOOL flush);
```

Description

This service completes drawing on the specified canvas. The canvas tracks draw nesting levels and triggers a frame buffer swap or refresh operation when the draw nesting level returns to 0. This service should only be called by the application to close a drawing sequence begun with gx_canvas_drawing_initiate().

Parameters

canvas	Pointer to canvas control block
flush	If ***GX_TRUE***, canvas changes are flushed to the display

Return Values

GX_SUCCESS	(0x00)	Successful drawing completion
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

```
/* 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_inititate, 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

GX_SUCCESS	(0x00)	Successful drawing initiation
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_CANVAS	(0x20)	Invalid canvas
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Example

```
/* Initiate drawing on “my_canvas”. */
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.

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_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

```
/* Draw an ellipse of semi-major radius 100, semi-minor radious 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_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_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_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

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

/* If status is GX_SUCCESS the line 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_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_pie_draw

Draw pie

Prototype

```
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 brush. The pie is clipped to the canvas invalid region.

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_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

```
/* 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 on "my_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_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.

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

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_pixelmap_blend

Blend pixelmap

Prototype

```
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 opague). 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

Allowed From

Initialization and threads

Example

```
/* Draw pixelmap on active canvas */

GX_PIXELMAP *map;
gx_system_pixelmap_get(ID_MY_PIXELMAP, &map);

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

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

See Also

gx_canvas_block_move, gx_canvas_pixelmap_tile, gx_canvas_pixelmap_draw

gx_canvas_pixel_draw

Draw pixel

Prototype

```
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.

Parameters

point	x,y position of pixel to draw

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
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate

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: */
gx_canvas_pixel_draw(point);

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

See Also

gx_canvas_block_move, gx_canvas_pixelmap_tile, gx_canvas_pixelmap_blend,

gx_canvas_pixelmap_draw

Draw pixelmap

Prototype

```
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_COORDINATE	(0x21)	Invalid coordinate

Allowed From

Initialization and threads

Example

```
/* Draw pixelmap on canvas “my_canvas”. */
status =  gx_canvas_pixelmap_draw(10, 20, &my_pixelmap);

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

See Also

gx_canvas_block_move, gx_canvas_pixelmap_tile, gx_canvas_pixelmap_blend,

gx_canvas_pixelmap_tile

Tile pixelmap

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_SIZE	(0x19)	Invalid fill size
GX_INVALID_WIDGET	(0x12)	Pixelmap widget not valid

Allowed From

Initialization and threads

Example

```
/* Tile pixelmap on screen “my_canvas”. */
status =  gx_canvas_pixelmap_tile(&tile_area, &my_pixelmap);

/* If status is GX_SUCCESS the pixelmap “my_pixelmap” has been tiled on “my_canvas”. */
```

See Also

gx_canvas_block_move, gx_canvas_pixelmap_blend,
gx_canvas_pixelmap_draw,

gx_canvas_polygon_draw

Draw polygon

Prototype

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

Description

This service draws a polygon on the canvas.

Parameters

point_array	Array of points of the polygon
number_of_points	Number of points of polygon

Return Values

GX_SUCCESS	(0x00)	Successful polygon draw
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)

Allowed From

Initialization and threads

Example

```
/* Draw polygon “my_polygon” on canvas “my_canvas”. */
status =  gx_canvas_polygon_draw(&my_polygon, 4);

/* If status is GX_SUCCESS the polygon “my_polygon” 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_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_SIZE	(0x19)	Invalid rectangle size

Allowed From

Initialization and threads

Example

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

/* If status is GX_SUCCESS the rectangle “my_rectangle” 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_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_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

canvas	Pointer to canvas control block
x	Pixels to shift on the X axis
y	Pixels to shift on the Y axis

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

```
/* 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_text_draw

Draw text

Prototype

```
UINT  gx_cavas_text_draw(GX_VALUE x_start, GX_VALUE y_start,
                         GX_CHAR *string, INT length);
```

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
length	If length >= 0, limits the number of characters drawn to length. If lengh < 0, the entire string until NULL terminator is drawn.

Return Values

GX_SUCCESS	(0x00)	Successful text draw
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COORDINATE	(0x21)	Invalid coordinate(s)
GX_INVALID_SIZE	(0x19)	Invalid size of string

Allowed From

Initialization and threads

Example

```
/* 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 on “my_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_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,
                         ULONG style, USHORT checkbox_id,
                         GX_CONST GX_RECTANGLE *size);
```

Description

This service creates a checkbox widget with the specified properties.

Parameters

checkbox	Pointer to checkbox control block
name	Logical name of checkbox widget
parent	Pointer to the parent widget
text_id	Resource ID of checkbox text
style	Style of checkbox. **Appendix D** contains pre-defined general styles for all widgets as well as widget specific styles.
checkbox_id	Application-defined ID of checkbox
size	Dimensions of checkbox

Return Values

GX_SUCCESS	(0x00)	Successful checkbox 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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

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

```
UINT  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

GX_SUCCESS	(0x00)	Successful checkbox draw
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

```
/* Draw "my_checkbox". */
status =  gx_checkbox_draw(&my_checkbox);

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

See Also

gx_checkbox_create, gx_checkbox_event_process, gx_checkbox_select

gx_checkbox_event_process

Process checkbox event

Prototype

```
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

Initialization and threads

Example

```
/* Process “my_event” for the checkbox “my_checkbox”. */
status =  gx_checkbox_event_process(&my_checkbox, &my_event);

/* If status is GX_SUCCESS the event for checkbox “my_checkbox” has been processed. */
```

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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x12)	Resource ID 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_SIABLED_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

GX_SUCCESS	(0x00)	Successful checkbox select
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

```
/* 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

```
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 checkbox select
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

```
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 retireved. */
```

See Also

gx_circular_gauge_angle_set, gx_circular_gauge_animation_set, gx_circular_gauge_create

gx_circular_gauge_angle_set

Set target angle

Prototype

```
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

gauge	Pointer to circular gauge control block
angle	Target needle angle

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_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* 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_create

gx_circular_gauge_animation_set

Set animation parameters

Prototype

```
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_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* 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_create

gx_circular_gauge_create

Create circular gauge

Prototype

```
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 circular gauge info control block
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	Starting x-coordinate position
ypos	Starting 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_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Create “my_gauge”. */
status =  gx_circular_gauge_create(&my_gauge, “my_gauge”, &my_parent,
                    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_context_brush_default

Set brush of current drawing context

Prototype

UINT **gx_context_brush_default**(GX_DRAW_CONTEXT *context);

Description

This service creates a default brush within the specified drawing context.

Parameters

context	Pointer to context control block

Return Values

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

Allowed From

Initialization and threads

Example

```
/* Select "my_context" as the current context. */
status = gx_context_brush_default(&my_context);

/* If status is GX_SUCCESS the context "my_context" has been set as the 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 screen 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 The supported brush styles

include:

GX_BRUSH_OUTLINE: 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 addtional to optionally being fill. If the GX_BRUSH_OUTLINE style is set and the GX_BRUSH_SOLID_FILL flag is clear, the shape is only outlined.

GX_BRUSH_SOLID_FILL: This brush style also 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: This brush style also applies to shaped drawing functions, and indicates that the requested shape should be patten filled with the current brush pixelmap.

GX_BRUSH_ALIAS: 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: This flag applies to text drawing, and indicates that subsequent text drawn should be underlined.

GX_BRUSH_ROUND: 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.

GX_BRUSH_PATTERN: This flag applies to line drawing and outline drawing. This flag requests that lines are drawn dashed, or with an on-off pattern, rather than solid lines.

Return Values

GX_SUCCESS	(0x00)	Successful context brush define
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style

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)	Successful context brush get
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
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_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

brush	Pointer to brush to use for current context

Return Values

GX_SUCCESS	(0x00)	Successful context brush set
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* 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_pattern_set

Set brush pattern of current drawing context

Prototype

```
UINT  gx_context_brush_pattern_set(ULONG pattern);
```

Description

This service sets the brush pattern of the current screen context. The brush pattern is used for drawing dashed lines or dashed outlines.

Parameters

pattern	Pattern to be used for the brush. This is a simple hexidecimal 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
GX_CALLER_ERROR	(0x11)	Invalid caller of this function

Allowed From

Initialization and threads

Example

```
/* Set the brush of the current context. */
status =  gx_context_brush_pattern_set(my_brush);

/* If status is GX_SUCCESS the brush 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_style_set

Set brush style of current drawing context

Prototype

```
UINT  gx_context_brush_style_set(UINT  style);
```

Description

This service sets the brush style of the current screen 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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
/* Set the brush style of the current context. */
status =  gx_context_brush_style_set(GX_STYLE_BORDER_NONE);

/* 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

```
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
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDTH	(0x1E)	Invalid width

Allowed From

Initialization and threads

Example

```
/* 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

```
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.

Parameters

color_id	Resource ID of color requested.
return_color	Address of variable to hold returned color value.

Return Values

GX_SUCCESS	(0x00)	Successful context fill color set
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
GX_COLOR color_value;

/* Get the color vaue. */
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

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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

```
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)	Successful context fill color set
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
GX_FONT *my_font;

/* Get the font pointer. */
status =  gx_context_font_get(MY_MIDSIZE_FONT, &my_font);
```

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

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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

```
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)	Successful context fill color set
GX_INVALID_CONTEXT	(0x06)	No active drawing context
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
GX_PIXELMAP *map;

/* Get the pixelmap pointer. */
status =  gx_context_pixelmap_get(MY_PIXELMAP_ID, &map);
```

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 pixelmap of current draw context

Prototype

```
UINT  gx_context_pixelmap_set(GX_RESOURCE_ID  pixelmap_id);
```

Description

This service assign the pixelmap of the active brush in the current drawing context.

Parameters

pixelmap_id	Pixelmap resource ID to use for current context

Return Values

GX_SUCCESS	(0x00)	Successful context pixelmap set
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* Set pixelmap of the current context to MY_PIXELMAP_ID. */
status =  gx_context_pixelmap_set(MY_PIXELMAP_ID);

/* If status is GX_SUCCESS the pixelmap 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

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COLOR	(0x15)	Invalid color
GX_INVALID_STYLE	(0x18)	Invalid style

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

GX_SUCCESS	(0x00)	Successful context raw fill color set
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COLOR	(0x15)	Invalid color

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

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_COLOR	(0x15)	Invalid color

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_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 screen create
GX_PTR_ERROR	(0x07)	Invalid pointer

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

```
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 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 screen create
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
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 *),
                  VOID *optional_driver_info,
                  UINT color_format, GX_VALUE width,
                  GX_VALUE height);
```

Description

This service creates a screen.

Parameters

display	Pointer to display control block
name	Name of the display
display_driver_setup	Pointer to display driver setup function
optional_driver_info	Pointer to optional driver information
color_format	Color format, as defined in **Appendix C**
width	Number of pixels on the x-axis
height	Number of pixels on the y-axis

Return Values

GX_SUCCESS	(0x00)	Successful screen 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_SCREEN_SIZE	(0x23)	Invalid screen control block size
GX_INVALID_FORMAT	(0x24)	Invalid screen format

Allowed From

Initialization and threads

Example

```
/* Create screen “my_display”. */
status =  gx_display_create(&my_display, “my display”, GX_NULL,
                                                GX_COLOR_FORMAT_16BIT_RGB, 320, 480);

/* If status is GX_SUCCESS the screen “my_display” has been created. */
```

See Also

gx_display_destroy

gx_display_delete

Destroy display

Prototype

```
UINT  gx_display_delete(GX_DISPLAY *display,
                  VOID (*display_driver_cleanup)(GX_DISPLAY *))
```

Description

This service shutdown a display, and cleans up allocated resources.

Parameters

display	Pointer to display control block
display_driver_cleanup	Pointer to display driver cleanup function

Return Values

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

Allowed From

Initialization and threads

Example

```
/* Delete a display “my_display”. */
status =  gx_display_delete(&my_display, GX_NULL);

/* 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

```
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 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 screen create
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
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_pixelmap_table_set

gx_display_pixelmap_table_set

Assign display font table

Prototype

```
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 screen create
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
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_drop_list_close

Close a drop list

Prototype

```
UINT gx_drop_list_close(GX_DROP_LIST *drop_list);
```

Description

This service closes a 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

```
/* Close a drop list */
status = gx_drop_list_close(&drop_list);

/* If status is GX_SUCCESS the screen “my_display” has been destroyed. */
```

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.

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, if the total number of list rows are greater than the list visible rows. The user must provide a function to create list row widgets as the list is scrolled. Only the visible number of list widgets are required at one time, and the application can therefore re-use widget slots that have been scrolled out of view.
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 screen create
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
GX_DROP_LIST my_drop_list;
VOID create_list_widget(GX_VERTICAL_LIST *list, GX_WIDGET *row, INT index);

GX_RECTANGLE size;

gx_utility_rectangle_define(&size, 10, 10, 80, 40);

gx_drop_list_create(&my_drop_list,
        "my drop list", GX_NULL,
        10, 100, create_list_widget,
        GX_STYLE_BORDER_RECESSED | GX_STYLE_ENABLED,
        ID_DROP_LIST, &size)
```

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 horizontal list 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

```
/* Process "my_event" for drop list "my_list". */
status =  gx_drop_list_event_process(&my_list, &my_event);

/* If status is GX_SUCCESS the event for drop list "my_list" has been processed. */
```

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

```
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 screen create
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
GX_DROP_LIST mylist;
gx_drop_list_open(&mylist);
```

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

Set pixelmap to the drop list

Prototype

```
UINT gx_drop_list_pixelmap_set(GX_DROP_LIST *drop_list,
                                          GX_RESOURCE_ID id);
```

Description

Assign a pixelmap to the icon button displayed on a previously created drop list.

Parameters

drop_list	Pointer to the drop list control block
id	Resource ID to the pixlemap

Return Values

GX_SUCCESS	(0x00)	Successful screen create
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid pixlemap ID

Allowed From

Initialization and threads

Example

```
GX_DROP_LIST mylist;

/* create the drop list here */

/* Assign a pixelmap id, which will turn on the list button */
gx_drop_list_pixelmap_set(&mylist, GX_PIXELMAP_ID_DOWN_ARROW);
```

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

Destroy display

Prototype

```
UINT gx_drop_list_popup_get(GX_DROP_LIST *drop_list,
                            GX_VERTICAL_LIST **return_list)
```

Description

This service retrieves the popup list from the drop list.

Parameters

drop_list	Pointer to the drop list control block
return_list	Pointer to the vertical list stored in the drop list

Return Values

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

Allowed From

Initialization and threads

Example

```
GX_DROP_LIST drop_list;
GX_VERTICAL_LIST *vertical_list;

gx_drop_list_popup_get(&drop_list, &vertical_list)
```

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

```
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

```
/* 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_horinzontal_list_selected_widget_get, gx_horizontal_list_selected_set, gx_horizontal_list_total_columns_set

gx_horizontal_list_create

Create horizontal list

Prototype

```
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

horizontal_list	Horizontal list widget control block
name	Name of horizontal list
parent	Pointer to parent widget
total_columns	Total number of comumns in horizontal list
callback	**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.**
style	Style of scrollbar widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
horizontal_list_id	Application-defined ID of horizontal list
size	Dimensions of the horitzonal list

Return Values

GX_SUCCESS	(0x00)	Successfully created the horizontal 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_WIDGET	(0x14)	Invalid widget control block size
GX_INVALID_VALUE	(0x22)	Number of columns not valid

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_horinzontal_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

```
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
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Process “my_event” for horizontal list “my_list”. */
status =  gx_horizontal_list_event_process(&my_list, &my_event);

/* If status is GX_SUCCESS the event for horizontal list “my_list” has been processed. */
```

See Also

gx_horizontal_list_children_position, gx_horizontal_list_create, gx_horizontal_list_page_index_set, gx_horizontal_list_selected_index_get, gx_horinzontal_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

```
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
event	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
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Sets the starting page index of horizontal list “my_list” as “top_index”. */
status =  gx_horizontal_list_page_index_set(&my_list, &top_index);

/* If status is GX_SUCCESS the starting page index of “m_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_horinzontal_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_horizobntal_LIST
                                            *horizontal_list,
                                      INT *return_index);
```

Description

This service returns the selected list entry index of the horizontal list.

Parameters

horizontal_list	Horizontal list widget control block
return_index	Destination for return list index

Return Values

GX_SUCCESS	(0x00)	Successful obtained the horizontal list entry
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

```
/* 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” 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_horinzontal_list_selected_widget_get, gx_horizontal_list_selected_set, gx_horizontal_list_total_columns_set

gx_horizontal_list_selected_widget_get

Get selected entry from horizontal list

Prototype

```
UINT gx_horizontal_list_selected_widget_get(GX_horizobntal_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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_FAILURE	(0x10)	Selected widget has been scrolled from view in a list with more rows than client children.

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_selected_set

Set entry in horizontal list

Prototype

```
UINT gx_horizontal_list_selected_set(GX_HORIZONATAL_LIST
                                     *horizontal_list,
                                        GX_WIDGET *list_entry);
```

Description

This service sets the list entry in the horizontal list.

Parameters

horizontal_list	Horizontal list widget control block
list_entry	Pointer to new list entry

Return Values

GX_SUCCESS	(0x00)	Successfully set the horizontal list entry
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

```
/* Set the list entry at the current index of horizontal list “my_list”. */
status =  gx_horizontal_list_selected_set(&my_list, &my_entry);

/* If status is GX_SUCCESS, the current index points to the widget “my_entry”. */
```

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_horinzontal_list_selected_widget_get, 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_horizobntal_LIST
                                           *horizontal_list,
                                          INT count);
```

Description

This service assigns the total number of columns to be displayed by the horizontal list.

Parameters

horizontal_list	Horizontal list widget control block
count	Number of columns to display

Return Values

GX_SUCCESS	(0x00)	Successful assigned column count
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* Tell my list to display 20 total columns. */
status =  gx_horizontal_list_total_columns_set(&my_list, 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_horinzontal_list_selected_widget_get, gx_horizontal_list_selected_set

gx_horizontal_scrollbar_create

Create horizontal scrollbar

Prototype

```
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

scrollbar	Scrollbar widget control block
name	Name of scrollbar
parent	Pointer to parent widget
appearance	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. The scrollbar appearance structure holds the following fields:

gx_scroll_width:	The total width, in pixels, of the scrollbar widget.
gx_scroll_thumb_width:	The width, in pixels, of the thumb button which slides on the scrollbar. This value is usually some number of pixels less than the total scrollbar width.
gx_scroll_thumb_travel_min:	An offset, in pixels, between the scrollbar end position and the minimuim limit to which the thumb button is allowed to travel. This limit

	can be use to prevent the thumb button from travelling to the very end of the scrollbar.
gx_scroll_thumb_travel_max:	An offset value, in pixels, between the scrollbar end position and the maximum limit to which the scrollbar thumb is allowed to travel.
gx_scroll_fill_pixelmap:	An optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar uses this pixelmap to draw the scrollb ar background.
gx_scroll_thumb pixelmap:	An optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar thumb button uses this pixelmap to draw itself.
gx_scroll_up_pixelmap:	An optional pixelmap ID. If this pixelmap ID is not zero, the scrollbar end button uses this pixelmap ID to draw the scrollbar left end button.
gx_scroll_down_pixelmap:	An optional pixelmap ID. If this pixelmap ID is not zero. the scrollbar end button used this pixelmap ID to draw the scrollbar right end button.
gx_scroll_fill_color:	The fill color used to draw the background of the scrollbar widget.
gx_scroll_button_color:	The fill color used to draw the scrollbar thumbbutton and end buttons.

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 horizontal 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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
/* Create horizontal scrollbar “my_scrollbar”. */
status =  gx_horizontal_scrollbar_create(&my_scrollbar, “my_horizontal_scrollbar”, &my_parent,
                                             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

Create icon button

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.

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
size	Dimensions of icon button

Return Values

GX_SUCCESS	(0x00)	Successfully created icon button
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 “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

```
UINT  gx_icon_button_create(GX_ICON_BUTTON *button);
```

Description

This service draws the specified icon button widget.

Parameters

button	Pointer to icon button control block

Return Values

GX_SUCCESS	(0x00)	Successfully drawn icon button
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

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_pixelmap_set

Create icon button

Prototype

```
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	Pointer to icon button control block
icon_id	Resource ID of pixelmap

Return Values

GX_SUCCESS	(0x00)	Successfully created icon button
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

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_create

Create icon

Prototype

```
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_WIDGET_SIZE	(0x14)	Invalid widget control block
size		
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_COORDINATE	(0x21)	Invalid coordinates

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

```
UINT  gx_icon_draw(GX_ICON *icon);
```

Description

This service draws the specified icon widget.

Parameters

icon	Pointer to icon widget control block

Return Values

GX_SUCCESS	(0x00)	Successful icon drawing
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

```
/* Draw “my_icon”. */
status =  gx_icon_draw(&my_icon);

/* If status is GX_SUCCESS the icon “my_icon” has been drawn. */
```

See Also

gx_icon_button_create, gx_icon_create, gx_icon_pixelmap_set

gx_icon_event_process

Icon widget event processing

Prototype

```
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 icon drawing
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

```
switch(event_ptr->gx_event_type)
{
case GX_EVENT_SHOW:
        /* Do default handling. */
        status =  gx_icon_event_process(icon, event_ptr);

        /* add my 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

```
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
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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_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_BUTTONS is derived from GX_TEXT_BUTTON, and all GX_TEXT_BUTTON services can therefore be used with a multi-line text button.

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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* Create multi-line 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 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

gx_multi_line_text_button_draw

Draw multi-line text button

Prototype

```
UINT  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 internaly 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

button	Pointer to text button control block

Return Values

GX_SUCCESS	(0x00)	Successful text button draw
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

```
/* Draw the text button “my_text_button”. */
void MyButtonDraw(GX_MULTI_LINE_TEXT_BUTTON *button)
{
        /* do the normal drawing first */
        status =  gx_multi_line_text_button_draw(&my_text_button);

        /* If status is GX_SUCCESS, the text button “my_text_button” was drawn. */
        /* Add custom drawing here */
}
```

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_event_process

Default event handling for multi-line text button

Prototype

```
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

```
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_proces(button, event_ptr);
            /* add custom actions here */
            break;
   default:
            gx_multi_line_text_button_event_proces(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_id_set

Set text resource ID to the text button

Prototype

```
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

```
/* 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

Prototype

```
UINT  gx_mult_line_text_button_text_set(GX_MULTI_LINE_TEXT_BUTTON
                              *text_button, GX_CHAR *text)
```

Description

This service assigns the specified string to the text button. 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

Allowed From

Initialization and threads

Example

```
/* Set the string “my string” to the text button “my_text_button”. */
status =  gx_multi_line_text_button_text_set(&my_text_button, “my\rstring”);

/* If status is GX_SUCCESS, the string “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_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.

Parameters

text_input	Multi-line text input widget control block
name	Name of text input widget
parent	Pointer to parent widget
input_buffer	Pointer to text input buffer
buffer_size	Size of text input buffer
style	Style of text input widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
text_input_id	Application-defined ID of text input
size	Dimensions of text input widget

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
GX_ALREADY_CREATED	(0x13)	Widget already created
GX_INVALID_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* 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, sizeof(my_buffer),  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_single_line_text_input_buffer_get, gx_single_line_text_input_create, 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_scroll, 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_input_buffer_get

Retrieves buffer information of text input widget

Prototype

```
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 the text input widget.

Parameters

text_input	Multi-line text input widget control block
buffer_address	The address of the input buffer
content_size	The count of the input data
buffer_size	The size of the input buffer

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

Allowed From

Initialization and threads

Example

```
/* 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_create, gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, 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_scroll, 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_input_style_add

Add styles

Prototype

```
UINT gx_multi_line_text_input_style_add(GX_MULTI_LINE_TEXT_INPUT
                                                  *text_input,
                              ULONG style);
```

Description

This service adds styles to multi line text input widgtet.

Parameters

text_input	Multi-line text input widget control block
style	Styles to add. **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

```
/* 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 text input style has been added */
```

See Also

gx_multi_line_text_input_buffer_get, gx_multi_line_text_input_create, gx_multi_line_text_input_style_remove, gx_multi_line_text_input_style_set, 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_scroll, 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_input_style_remove

Remove styles

Prototype

```
UINT gx_multi_line_text_input_remove(GX_MULTI_LINE_TEXT_INPUT
                                              *text_input,
                        ULONG style);
```

Description

This service removes styles from 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_add(&my_text_input,   GX_STYLE_CURSOR_ALWAYS);

/* If status is GX_SUCCESS the text input style has been removed */
```

See Also

gx_multi_line_text_input_buffer_get, gx_single_line_text_input_create, gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_set, 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_scroll, 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_input_style_set

Set styles

Prototype

```
UINT gx_multi_line_text_input_set(GX_MULTI_LINE_TEXT_INPUT
                                              *text_input,
                         ULONG style);
```

Description

This service sets styles for 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 create
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_buffer_get, gx_multi_line_text_input_create, gx_multi_line_text_input_style_add, gx_multi_line_text_input_style_remove, 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_scroll, 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_char_index_set

Assign a character index buffer to the multi line text view widget

Prototype

```
UINT  gx_multi_line_text_view_char_index_set(
      GX_MULTI_LINE_TEXT_VIEW *text_view,
      int *index_buffer_ptr);
```

Description

This service assigns a buffer to the mult-line text view for .

Parameters

text_view	Multi-line text view widget control block
name	Name of text view widget
parent	Pointer to parent widget
input_buffer	Pointer to text view buffer
style	Style of text view widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
text_input_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_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Create multi-line text view widget  “my_text_view”. */
status =  gx_multi_line_text_view_create(&my_text_view, “my_text_view”, &my_parent,
                          my_buffer, GX_STYLE_BORDER_RAISED,
                          MY_TEXT_VIEW_ID, &size);

/* If status is GX_SUCCESS the text view “my_text_view” has been created. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, gx_multi_line_text_view_scroll, 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_create

Create multi-line text view

Prototype

```
UINT  gx_multi_line_text_view_create(GX_MULTI_LINE_TEXT_VIEW
                                           *text_view,
                                     GX_CONST GX_CHAR *name,
                                     GX_WINDOW *parent,
                                     GX_CHAR *input_buffer,
                                     ULONG style,
                                     USHORT text_view_id,
                                     GX_CONST GX_RECTANGLE *size);
```

Description

This service creates a multi-line text view widget.

Parameters

text_view	Multi-line text view widget control block
name	Name of text view widget
parent	Pointer to parent widget
input_buffer	Pointer to text view buffer
style	Style of text view widget. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
text_input_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_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Create multi-line text view widget  “my_text_view”. */
status =  gx_multi_line_text_view_create(&my_text_view, “my_text_view”, &my_parent,
                                         my_buffer, GX_STYLE_BORDER_RAISED,
                                         MY_TEXT_VIEW_ID, &size);

/* If status is GX_SUCCESS the text view “my_text_view” has been created. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_font_set, gx_multi_line_text_view_scroll, 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_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 the multi-line text view widget.

Parameters

text_view	Multi-line text view widget control block
event	Pointer to event to process

Return Values

GX_SUCCESS	(0x00)	Successful multi-line text view 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

```
/* Process event for multi-line text view widget  “my_text_view”. */
status =  gx_multi_line_text_view_event_processe(&my_text_view, &my_event);

/* If status is GX_SUCCESS the event “my_event” for text view “my_text_view” has been processed. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_font_set, gx_multi_line_text_view_scroll, 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_font_set

Set the font for the multi line text view

Prototype

```
UINT gx_multi_line_text_view_text_id_set(GX_MULTI_LINE_TEXT_VIEW
                                          *text_view,
                                      GX_RESOURCE_ID font_id);
```

Description

This service assigns the resource ID of a font to the multi-line text view widget.

Parameters

text_view	Multi-line text view widget control block
font_id	Resource ID for the text string

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
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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_single_line_text_input_create, gx_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll, 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_scroll

Scroll multi-line text view

Prototype

```
UINT  gx_multi_line_text_view_scroll(GX_MULTI_LINE_TEXT_VIEW
                                         *text_view,
                                 GX_VALUE amount_to_scroll);
```

Description

This service scrolls a multi-line text view widget.

Parameters

text_view	Multi-line text view widget control block
amount_to_scroll	Number of rows to scroll

Return Values

GX_SUCCESS	(0x00)	Successful multi-line text view 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 not valid

Allowed From

Initialization and threads

Example

```
/* Scroll multi-line text view widget  "my_text_view" by 10 rows. */
status =  gx_multi_line_text_view_scroll(&my_text_view, 10);

/* If status is GX_SUCCESS the text view "my_text_view" has been scrolled by 10 rows. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, 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_text_color_set, gx_multi_line_text_view_text_id_set, gx_multi_line_text_view_text_set

gx_multi_line_text_view_text_color_set

Set the text color for the multi line text view

Prototype

```
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);
```

Description

This service assigns text color to the multi-line text view widget.

Parameters

text_view	Multi-line text view widget control block
normal_text_color_id	Resource ID of the color for normal text
selected_text_color_id	Resource ID of the color selected text

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
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* Set normal text color and selected text color to the multi-line text view widget  "my_text_view". */
status =  gx_multi_line_text_view_text_color_set(&my_text_view, NORMAL_TEXT_COLOR,
                                                  SELECTED_TEXT_COLOR);

/* If status is GX_SUCCESS the text view "my_text_view" will display the text using specified colors. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, 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_scroll, gx_multi_line_text_view_text_id_set, gx_multi_line_text_view_text_set

gx_multi_line_text_view_text_id_set

Set the string ID for the multi line text view

Prototype

```
UINT gx_multi_line_text_view_text_id_set(GX_MULTI_LINE_TEXT_VIEW
                                           *text_view,
                                        GX_RESOURCE_ID text_id);
```

Description

This service assigns 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 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	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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 view “my_text_view” will display the string
  resource STRING_ID. */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, 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_scroll, gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_set

gx_multi_line_text_view_text_set

Set the string for the multi line text view

Prototype

```
UINT  gx_multi_line_text_view_text_set(GX_MULTI_LINE_TEXT_VIEW
                                          *text_view,
                                          GX_CHAR *text);
```

Description

This service assigns a text string to the multi-line text view widget.

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_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 view "my_text_view" will display the string "my string". */
```

See Also

gx_single_line_text_input_create, gx_multi_line_text_input_create, 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_scroll, gx_multi_line_text_view_text_color_set, gx_multi_line_text_view_text_id_set

gx_pixelmap_button_create

Create pixelmap button

Prototype

```
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.

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_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* Create “my_pixelmap_button”. */
status =  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_pxielmap_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

```
UINT  gx_pixelmap_button_draw(GX_PIXELMAP_BUTTON *button);
```

Description

This service draws a pixelmap button widget.

Parameters

button	Pointer to pixelmap button control block

Return Values

GX_SUCCESS	(0x00)	Successful pixelmap button draw
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

```
/* Draw "my_pixelmap_button". */
status =  gx_pixelmap_button_draw(&my_pixelmap_button);

/* If status is GX_SUCCESS the pixelmap button "my_pixelmap_button" has been drawn. */
```

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_pxielmap_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_event_process

Pixelmap button event processing

Prototype

```
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
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
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_pxielmap_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_pixelmap_set

Assign pixelmaps to button

Prototype

```
UINT  gx_pixelmap_button_pixelmap_set(GX_PIXELMAP_BUTTON *button,
                                      GX_RESOURCE_ID normal_id,
                                      GX_RESOURCE_ID selected_id,
                                      GX_RESOURCE_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
GX_INVALID_RESOURCE_ID	(0x22)	Resource ID not valid

Allowed From

Initialization and threads

Example

```
/* Draw "my_pixelmap_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 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

```
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_ALREADY_CREATED	(0x13)	Widget already created
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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

```
UINT  gx_pixelmap_prompt_draw(GX_PIXELMAP_PROMPT *prompt);
```

Description

This service draws a pixelmap prompt widget.

Parameters

prompt	Pointer to pixelmap prompt control block

Return Values

GX_SUCCESS	(0x00)	Successful pixelmap prompt draw
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

```
/* Draw “my_pixelmap_prompt”. */
status =  gx_pixelmap_prompt_draw(&my_pixelmap_prompt);

/* If status is GX_SUCCESS the pixelmap prompt “my_pixelmap_prompt” has been drawn. */
```

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. Up to size pixelmaps may be assigned. 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	(0x22)	Resource ID not valid

Allowed From

Initialization and threads

Example

```
/* Assign pixelmap 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 pixelmap 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 pixelmap slider

Prototype

```
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 pixelmap slider widget.

Parameters

slider	Pointer to pixelmap slider control block
name	Logical name of pixelmap 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. The GX_SLIDER_INFO structure has the following integer members:
	
	gx_slider_info_min_value;
	gx_slider_info_max_value;
	gx_slider_info_current_value;
	gx_slider_info_increment;
	gx_slider_info_min_travel;
	gx_slider_info_max_travel;
	gx_slider_info_needle_pos;
	gx_slider_info_needle_width;
	gx_slider_info_needle_height;
	gx_slider_info_needle_inset;
	gx_slider_info_needle_hotspot_offset;

pixelmap_info	Pointer to a GX_PIXELMAP_SLIDER_INFO structure which defines the pixelmaps used the draw the slider background and needle. 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 pre-defined general styles for all widgets as well as widget-specific styles.
pixelmap_slider_id	Application-defined ID of pixelmap slider
size	Dimensions of pixelmap prompt

Return Values

GX_SUCCESS	(0x00)	Successfully created pixelmap slider
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)	Parent widget not valid

Allowed From

Initialization and threads

Example

```
/* 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

```
UINT  gx_pixelmap_slider_draw(GX_PIXELMAP_SLIDER *slider);
```

Description

This service draws a pixelmap slider widget.

Parameters

slider	Pointer to pixelmap slider control block

Return Values

GX_SUCCESS	(0x00)	Successful pixelmap slider draw
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

```
/* Draw “my_pixelmap_slider”. */
status =  gx_pixelmap_slider_draw(&my_pixelmap_slider);

/* If status is GX_SUCCESS the pixelmap slider “my_pixelmap_slider” has been drawn. */
```

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

```
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

```
/* Process event for the pixelmap slider “my_pixelmap_slider”. */
status =  gx_pixelmap_slider_event_process(&my_pixelmap_slider, &my_event);

/* If status is GX_SUCCESS the event has been processed for pixelmap slider “my_pixelmap_slider”.
*/
```

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

slider	Pointer to pixelmap slider control block
pixinfo	Slider information block

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

```
/* Draw “my_pixelmap_button”. */
status =  gx_pixelmap_slider _pixelmap_set (&my_pixelmap_slider, &pixinfo);

/* If status is GX_SUCCESS the pixelmap slider 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_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_create

Create progress bar

Prototype

```
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.

Parameters

progress_bar	Progress bar control block
name	Logical name
parent	Pointer to the parent widget
progress_bar_info	Pointer to progress bar information 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 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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
            /* Create progress bar */

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;


status = gx_progress_bar_create(&progress_bar, GX_NULL, GX_NULL,
                              &info, GX_STYLE_BORDER_THIN,
                              0, &size);
```

See Also
gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_draw

Draw a progress bar

Prototype

```
UINT  gx_progress_bar_draw(GX_PROGRESS_BAR *progress_bar);
```

Description

This service draws a progress bar.

Parameters

progress_bar	Progress bar control block

Return Values

GX_SUCCESS	(0x00)	Successful prompt create
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_draw(progress_bar);

/* if status == GX_SUCCESS the bar was successfully drawn. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_event_process

Progress bar event process function

Prototype

```
UINT  gx_progress_bar_event_process(GX_PROGRESS_BAR *progress_bar,
GX_EVENT *event_ptr);
```

Description

This service handles progress bar events.

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

Allowed From

Initialization and threads

Example

```
UINT status = gx_progress_bar_event_process(progress_bar, event_ptr);

/* if status == GX_SUCCESS the event was successfully processed. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_font_set

Assign font used by progress bar

Prototype

```
UINT  gx_progress_bar_font_set(GX_PROGRESS_BAR *progress_bar,
GX_RESOURCE_ID font_id);
```

Description

This service assigns the font used by a progress bar.

Parameters

progress_bar	Progress bar control block
font_id	Font resource id

Return Values

GX_SUCCESS	(0x00)	Successful prompt create
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_font_set(progress_bar, GX_FONT_ID_MEDIUM);

/* if status == GX_SUCCESS the font was successfully assigned. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_info_set

Redefine the progress bar information structure

Prototype

```
UINT  gx_progress_bar_info_set(GX_PROGRESS_BAR *progress_bar,
GX_PROGRESS_BAR_INFO *info);
```

Description

Re-assign all progress bar drawing information.

Parameters

progress_bar	Progress bar control block
info	Pointer to GX_PROGRESS_BAR_INFO structure

Return Values

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

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;


UINT 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_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_pixelmap_set

Assign pixelmap used to draw progress bar

Prototype

```
UINT  gx_progress_bar_pixelmap_set(GX_PROGRESS_BAR *progress_bar,
GX_RESOURCE_ID pixelmap_id);
```

Description

This service assigns the pixelmap used to fill the progress bar.

Parameters

progress_bar	Progress bar control block
pixelmap_id	Pixelmap resource id

Return Values

GX_SUCCESS	(0x00)	Successful prompt create
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_pixelmap_set(progress_bar,
GX_PIXELMAP_ID_PROGRESS_FILL);

/* if status == GX_SUCCESS the pixelmap was successfully assigned. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_range_set

Assign 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 assigns 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 prompt create
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 == GX_SUCCESS the progress bar range was successfully assigned. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_text_color_set

Assign the text color of a progress bar

Prototype

```
UINT  gx_progress_bar_text_color_set(GX_PROGRESS_BAR *progress_bar,
GX_RESOURCE_ID normal_text_color, GX_RESOURCE_ID selected_text_color);
```

Description

This service assigns the colors used to draw the progress bar text.

Parameters

progress_bar	Progress bar control block
normal_text_color	Resource ID of normal text color
selected_text_color	Resource ID of selected text color

Return Values

GX_SUCCESS	(0x00)	Successful prompt create
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_text_color_set(progress_bar, GX_COLOR_ID_WHITE,
GX_COLOR_ID_BLUE);

/* if status == GX_SUCCESS the progress bar text colors were successfully assigned. */
```

See Also

gx_progress_bar_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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

Assign 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	Progress bar control block
value	Progress bar current value

Return Values

GX_SUCCESS	(0x00)	Successful prompt create
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_draw, gx_progress_bar_event_process, gx_progress_bar_fill_color_set, 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_prompt_create

Create prompt

Prototype

```
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.

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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid 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

```
UINT  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

GX_SUCCESS	(0x00)	Successful prompt draw
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

```
/* Draw “my_prompt”. */
status =  gx_prompt_draw(&my_prompt);

/* If status is GX_SUCCESS the prompt “my_prompt” has been drawn. */
```

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

prompt	Pointer to prompt widget control block
font_id	Resource ID of font

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
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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);
```

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 B** 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. **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 prompt 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_RESOURCE_ID	(0x22)	Invalid resource ID

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);

/* 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_get

Get prompt text

Prototype

```
UINT  gx_prompt_text_get(GX_PROMPT *prompt,
                         GX_CHAR **return_text);
```

Description

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_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

```
/* Get the text of “my_prompt”. */
GX_CHAR *my_prompt_text;

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_id_set

Set prompt text ID

Prototype

```
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 color set
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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

Prototype

```
UINT  gx_prompt_text_set(GX_PROMPT *prompt, GX_CHAR *text);
```

Description

This service sets the text of a prompt widget.

Parameters

prompt	Pointer to prompt widget control block
text	Pointer to text

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_INVALID_WIDGET	(0x12)	Widget not valid

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_radio_button_create

Create radio button

Prototype

```
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.

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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_SIZE	(0x19)	Invalid size

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

```
UINT  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

button	Pointer to radio button widget control block

Return Values

GX_SUCCESS	(0x00)	Successful radio button draw
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

```
/* Draw “my_radio_button”. */
status =  gx_radio_button_draw(&my_radio_button);

/* If status is GX_SUCCESS the radio button “my_radio_button” has been drawn. */
```

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_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x12)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* Sets pixelmaps 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_scroll_thumb_create

Create scroll thumb

Prototype

```
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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
/* Create scroll thumb “my_scroll_thumb”. */
status = gx_scroll_thumb_create(&my_scroll_thumb, “my_scroll_thumb”,
&my_scrollbarGX_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

```
UINT  gx_scroll_thumb_draw(GX_SCROLL_THUMB *scroll_thumb);
```

Description

This service draws a scroll thumbwheel.

Parameters

scroll_thumb	Scroll thumb widget control block

Return Values

GX_SUCCESS	(0x00)	Successful scroll thumb draw
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

```
/* Draw scroll thumb “my_scroll_thumb”. */
status =  gx_scroll_thumb_draw(&my_scroll_thumb);

/* If status is GX_SUCCESS the scroll thumb “my_scroll_thumb” has been drawn. */
```

See Also

gx_scroll_thumb_create, gx_scroll_thumb_event_process

gx_scroll_thumb_event_process

Process scroll thumb event

Prototype

```
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

```
/* Process event for scroll thumb “my_scroll_thumb”. */
status =  gx_scroll_thumb_event_process(&my_scroll_thumb, &my_event);

/* If status is GX_SUCCESS the event for scroll thumb “my_scroll_thumb” has been processed. */
```

See Also

gx_scroll_thumb_create, gx_scroll_thumb_draw

gx_scrollbar_draw

Draw scrollbar

Prototype

```
UINT  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.

Parameters

scrollbar	Scrollbar widget to draw

Return Values

GX_SUCCESS	(0x00)	Successful scrollbar draw
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

```
/* Draw scrollbar “my_scrollbar”. */
status =  gx_scrollbar_draw(&my_scrollbar);

/* If status is GX_SUCCESS the scrollbar “my_scrollbar” has been drawn. */
```

See Also

gx_ horizontal_scrollbar_create, gx_scrollbar_event_process, gx_scrollbar_limit_check, gx_scrollbar_reset, gx_ vertical_scrollbar_create

gx_scrollbar_event_process

Process scrollbar event

Prototype

```
UINT  gx_scrollbar_event_process(GX_SCROLLBAR *scrollbar,
                                 GX_EVENT *event);
```

Description

This service processes a scrollbar event. A common event handling function is 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

Initialization and threads

Example

```
/* Process event for scrollbar “my_scrollbar”. */
status =  gx_scrollbar_event_process(&my_scrollbar, &my_event);

/* If status is GX_SUCCESS the event “my_event” for scrollbar “my_scrollbar” has been processed. */
```

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

GX_SUCCESS	(0x00)	Successful scrollbar limit check
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

```
/* 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_horizontal_scrollbar_create, gx_scrollbar_draw, gx_scrollbar_event_process, gx_scrollbar_reset, gx_vertical_scrollbar_create

gx_scrollbar_reset

Reset scrollbar

Prototype

```
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. The GX_SCROLL_INFO structure has the following integer members:

```
GX_VALUE gx_scroll_value      current scroll position
GX_VALUE gx_scroll_minimum    minimum reported position
GX_VALUE gx_scroll_maximum    maximum reported position
GX_VALUE gx_scroll_visible    parent window visible range
GX_VALUE gx_scroll_increment  scrollbar minimum delta value
```

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_vertical_scrollbar_create

gx_single_line_text_input_backspace

Process a backspace character in text input widget

Prototype

```
UINT  gx_single_line_text_input_backspace(
                     GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

Description

This service processes a backspace character. This service is called internally when a backspace character 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

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status =  gx_single_line_text_input_backspace(&my_text_input);

/* If status is GX_SUCCESS the text input widget has processed a backspace input character. */
```

See Also

gx_single_line_text_input_buffer_clear, gx_single_line_text_input_buffer_get, gx_single_line_text_input_create, 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_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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

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 delets all characters from the text input buffer.

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

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_backspace, gx_single_line_text_input_buffer, gx_single_line_text_input_create, 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_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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

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

text_input	Single-line text input widget control block
buffer_address	The address of the input buffer
content_size	The count of the input data
buffer_size	The size of the input buffer

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

Allowed From

Initialization and threads

Example

```
/* Retrieves 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_create, 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_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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_character_delete

Delete the character at the current cursor position

Prototype

```
UINT  gx_single_line_text_input_character_delete(
                  GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

Description

This service delete the character after the text input cursor position. This service is called internally when a delete character 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

Allowed From

Initialization and threads

Example

```
/* delete character. */
status =  gx_single_line_text_input_character_delete(&my_text_input);

/* If status is GX_SUCCESS the text input widget has processed a backspace input character. */
```

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_create, gx_single_line_text_input_character_insert, gx_single_line_text_input_draw, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_character_insert

Insert a character at current cursor position

Prototype

```
UINT  gx_single_line_text_input_character_insert(
                   GX_SINGLE_LINE_TEXT_INPUT *text_input,
                   GX_CHAR character);
```

Description

This service inserts a character into the text input string at the current cursor position.

Parameters

text_input	Single-line text input widget control block
character	Character value to be inserted

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

Allowed From

Initialization and threads

Example

```
/* Insert character at current cursor position. */
status =  gx_single_line_text_input_character_insert(&my_text_input, ‘A’);

/* If status is GX_SUCCESS the text input widget has inserted the characterr. */
```

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_create, gx_single_line_text_input_character_delete, gx_single_line_text_input_draw, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_create

Create a text input widget

Prototype

```
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.

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

Allowed From

Initialization and threads

Example

```
/* Create single-line text input widget  "my_text_input". */
static GX_CHAR input_string[20];
GX_RECTANGLE size;

gx_utility_rectangle_define(&size, 10, 10, 110, 40);
```

```
status =  gx_single_line_text_input_create(&my_text_input, “text_input”, GX_NULL, input_string, 20,
GX_STYLE_BORDER_RECESSED, 0, &size);

/* If status is GX_SUCCESS the text input widget has processed a backspace input character. */
```

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_draw, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_draw

Draw a text input widget

Prototype

```
UINT  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

GX_SUCCESS	(0x00)	Successful single-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

```
/* invoke backspace handler */
status =  gx_single_line_text_input_draw(&my_text_input);

/* If status is GX_SUCCESS the text input widget was drawnr. */
```

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_create, gx_single_line_text_input_character_delete, gx_single_line_text_input_character_insert, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_end

Move the text input cursor to the string end

Prototype

```
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.

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

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status =  gx_single_line_text_input_end(&my_text_input);

/* If status is GX_SUCCESS the cursor has been moved to the end of the text input 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_create, 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_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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

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 is the default event processing function of a gx_single_line_text_input widget.

Parameters

text_input	Single-line text input widget control block
event_ptr	Pointer to GX_EVENT structure

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

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status = gx_single_line_text_input_event_process(&my_text_input, &event);

/* If status is GX_SUCCESS the text input widget has processed the event */
```

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_create, 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_end, 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_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_home

Move the text input cursor to the home position

Prototype

```
UINT  gx_single_line_text_input_home(
      GX_SINGLE_LINE_TEXT_INPUT *text_input);
```

Description

Move the text input cursor position to the start of the input string.

Parameters

text_input	Single-line text input widget control block

Return Values

GX_SUCCESS	(0x00)	Success
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
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_create, 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_end, 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_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_left_arrow

Move input cursor one character to the left

Prototype

```
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 position to the left.

Parameters

text_input	Single-line text input widget control block

Return Values

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

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status =  gx_single_line_text_input_left_arrow(&my_text_input);

/* If status is GX_SUCCESS the text input cursor has been moved */
```

See Also

gx_single_line_text_input_backspace, gx_single_line_text_input_buffer_clear, gx_single_line_text_input_buffer_remove, gx_single_line_text_input_create, 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_end, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_position_get

Move cursor to pixel position

Prototype

```
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.

Parameters

text_input	Single-line text input widget control block
pixel_position	X value of pixel position

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

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
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_create, 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_end, 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_multi_line_text_input_create, gx_multi_line_text_view_create,
gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

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 position to the right.

Parameters

text_input	Single-line text input widget control block

Return Values

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

Allowed From

Initialization and threads

Example

```
/* move cursor 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 */
```

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_create, 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_end, 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_cursor_flag_set, gx_single_line_text_input_cursor_flag_clear

gx_single_line_text_input_style_add

Add styles

Prototype

```
UINT  gx_single_line_text_input_style_add(
      GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);
```

Description

This service adds styles to a 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)	Successful
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status =  gx_single_line_text_input_style_add(&my_text_input, GX_STYLE_CURSOR_SHOW);

/* If status is GX_SUCCESS the text input style has been 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_create, 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_end, 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_remove, gx_single_line_text_input_style_set, gx_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_style_remove

Remove styles

Prototype

```
UINT  gx_single_line_text_input_style_remove(
      GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);
```

Description

This service removes styles from single line text input widget.

Parameters

text_input	Single-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
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* invoke backspace handler */
status = gx_single_line_text_input_style_remove(&my_text_input, GX_STYLE_CURSOR_BLINK);

/* If status is GX_SUCCESS the text input style has been 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_create, 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_end, 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_multi_line_text_input_create, gx_multi_line_text_view_create, gx_multi_line_text_view_event_process, gx_multi_line_text_view_scroll

gx_single_line_text_input_style_set

Set styles

Prototype

```
UINT  gx_single_line_text_input_style_set(
      GX_SINGLE_LINE_TEXT_INPUT *text_input, ULONG style);
```

Description

This service sets styles for 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)	Successful
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_BLINK for single line text input widget "my_text_input" */
status = gx_single_line_text_input_style_set(&my_text_input, GX_STYLE_CURSOR_BLINK);

/* If status is GX_SUCCESS the text input styles has been 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_create, 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_end, 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_multi_line_text_input_create,

gx_multi_line_text_view_create, gx_multi_line_text_view_event_process,
gx_multi_line_text_view_scroll

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.

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. The GX_SLIDER_INFO structure has the following integer members:

```
INT gx_slider_info_min_value                  minimum reported value
INT gx_slider_info_max_value                  maximum reported value
INT gx_slider_info_current_value              current value
INT gx_slider_info_increment                  value delta
GX_VALUE gx_slider_info_min_travel            needle travel limit
GX_VALUE gx_slider_info_max_travel            needle travel limit
GX_VALUE gx_slider_info_needle_pos            current need position
GX_VALUE gx_slider_info_needle_width          needle width in pixels
GX_VALUE gx_slider_info_needle_height         needle height in pixels
GX_VALUE gx_slider_info_needle_inset          needle drawing position
GX_VALUE gx_slider_info_needle_hotspot_offset  needle hotspot offset
```

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
GX_INVALID_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* Create 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_pos = 0;
info.gx_slider_info_needle_width = 10;
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

```
UINT  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

GX_SUCCESS	(0x00)	Successful slider draw
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

```
/* Draw slider “my_slider”. */
status =  gx_slider_draw(&my_slider);

/* If status is GX_SUCCESS the slider “my_slider” has been drawn. */
```

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

```
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

Initialization and threads

Example

```
/* Process event for slider "my_slider". */
status =  gx_slider_event_process(&my_slider, &my_event);

/* If status is GX_SUCCESS the event for slider "my_slider" has been processed. */
```

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 incidated slider. The slider will update the needle position and redraw based on the new slider information.

Parameters

slider	Slider widget control block
info	Pointer to the slider information structure.

The slider information structure contains the following parameters:

```
INT gx_slider_info_min_value                 minimum reported value
INT gx_slider_info_max_value                 maximum reported value
INT gx_slider_info_current_value             current value
INT gx_slider_info_increment                 value delta
GX_VALUE gx_slider_info_min_travel           needle travel limit
GX_VALUE gx_slider_info_max_travel           needle travel limit
GX_VALUE gx_slider_info_needle_pos           current need position
GX_VALUE gx_slider_info_needle_width         needle width in pixels
GX_VALUE gx_slider_info_needle_height        needle height in pixels
GX_VALUE gx_slider_info_needle_inset         needle drawing position
GX_VALUE gx_slider_info_needle_hotspot_offset  needle hotspot offset
```

Return Values

GX_SUCCESS	(0x00)	Successfully set slider information
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* Process event 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

```
UINT  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

GX_SUCCESS	(0x00)	Successful slider needle draw
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

```
/* Draw needle for slider “my_slider”. */
status =  gx_slider_needle_draw(&my_slider);

/* If status is GX_SUCCESS the needle for slider “my_slider” has been drawn. */
```

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

```
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. The GX_SLIDER_INFO structure has the following integer members:
return_position	Pointer to destination for needle postion

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

```
/* Get the needle position for slider "my_slider". */
status =  gx_slider_needle_posistion_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

```
UINT  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

GX_SUCCESS	(0x00)	Successful slider tickmark draw
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

```
/* Draw tickmarks for slider "my_slider". */
status =  gx_slider_tickmarks_draw(&my_slider);

/* If status is GX_SUCCESS the tickmarks for slider "my_slider" have been drawn. */
```

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

slider	Slider widget control block
info	Pointer to slider info structure.
return_min_travel	Pointer to destination for minimum travel value
return_max_travel	Pointer to destination for maximum travel value

Return Values

GX_SUCCESS	(0x00)	Successful slider travel 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

```
/* Get travel information for 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

Calculate slider value

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. The GX_SLIDER_INFO structure has the following integer members:
	
	gx_slider_info_min_value;
	gx_slider_info_max_value;
	gx_slider_info_current_value;
	gx_slider_info_increment;
	gx_slider_info_min_travel;
	gx_slider_info_max_travel;
	gx_slider_info_needle_pos;
	gx_slider_info_needle_width;
	gx_slider_info_needle_height;
	gx_slider_info_needle_inset;
	gx_slider_info_needle_hotspot_offset;

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, &info, new_value);

/* If status is GX_SUCCESS the 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_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	Slider widget control block
info	Pointer to slider info structure. The GX_SLIDER_INFO structure has the following integer members:

```
INT gx_slider_info_min_value                 minimum reported value
INT gx_slider_info_max_value                 maximum reported value
INT gx_slider_info_current_value             current value
INT gx_slider_info_increment                 value delta
GX_VALUE gx_slider_info_min_travel           needle travel limit
GX_VALUE gx_slider_info_max_travel           needle travel limit
GX_VALUE gx_slider_info_needle_pos           current need position
GX_VALUE gx_slider_info_needle_width         needle width in pixels
GX_VALUE gx_slider_info_needle_height        needle height in pixels
GX_VALUE gx_slider_info_needle_inset         needle drawing position
GX_VALUE gx_slider_info_needle_hotspot_offset  needle hotspot offset
```

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
GX_INVALID_VALUE	(0x22)	Slider info or new value not valid

Allowed From

Initialization and threads

Example

```
/* Set new value for slider “my_slider”. */
status =  gx_slider_value_set(&my_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 sprite

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.

The GX_SPRITE widget requires an array of GX_SPRITE_FRAME structures to define the sprite animation. The GX_SPRITE_FRAME structure is defined as:

```
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;
```

The gx_sprite_frame_pixelmap is the ID of the pixelmap to be displayed for this frame. The ID can be 0.

The x and y offset fields specify an offset, if desired, from the top-left corner of the sprite widget to display the pixelmap.

The frame_delay field specifies the delay value, in GUIX timer ticks, after displaying this frame before advancing to the next sprite frame.

The background operation defines 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 */
```

The gx_sprite_frame_alpha value defines an 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.

Parameters

sprite	Sprite widget control block
name	Optional sprite name
parent	Pointer to parent widget
frame_list	An array of GX_SPRITE_FRAME structures
frame_count	specifies the number of entries in the frame list array
style	Style of sprite. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
sprite_id	Application-defined ID of sprite
size	Dimensions of sprite

Return Values

GX_SUCCESS	(0x00)	Successful sprite 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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* Create spriter "my_sprite". */
status =  gx_slider_create(&my_sprite, "my_spriter", &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 spriter "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

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” has 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
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

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

```
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)	Successful
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* 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

```
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)	Successful
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* 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_studio_widget_create

Create widget defined in Studio generated specifications file

Prototype

```
GX_WIDGET *gx_studio_widget_create(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.

The definition structure is always named according to this convention: <screen_name>_<widget_name>_define, where the <widget_name> field may be repeated multiple times if the widget is child of a child widget.

Parameters

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 named “child_popup”, which is a child of the top-level screen “main_screen”. */
widget =  gx_studio_widget_create(&main_screen_child_popup_define, &main_screen);

/* If widget != GX_NULL the widget was created. */
```

See Also

gx_studio_named_widget_create

gx_studio_named_widget_create

Create widget defined in Studio generated specifications file

Prototype

```
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

```
UINT *gx_studio_display_configure(USHORT 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

GX_SUCCESS	(0x00)	Successful
GX_FAILURE	(0x11)	Display could not be initialized

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_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

```
UINT  gx_system_active_language_set(UINT language);
```

Description

This service set the current language. The language index must be less than the number of columns in the application string table.

Parameters

language	Language index, defined in resource header file.

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

```
/* Mark widget “my_widget” 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_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

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

None

Allowed From

Initialization and threads

Example

```
/* Force immediate redraw operation. */
 gx_system_canvas_refresh();
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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

GX_SUCCESS	(0x00)	Successful dirty mark
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

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_dirty_partial_add

Mark partial area dirty

Prototype

```
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 add

Return Values

GX_SUCCESS	(0x00)	Successful partial dirty area mark
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_SIZE	(0x19)	Invalid size of dirty area

Allowed From

Initialization and threads

Example

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_draw_context_get

Get drawing context

Prototype

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_event_fold

Send event

Prototype

```
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

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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 “my_event” for processing. */
status =  gx_system_event_send(&my_event);

/* If status is GX_SUCCESS the event has been sent for processing. */
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_focus_claim

Claim focus

Prototype

```
UINT  gx_system_focus_claim(GX_WIDGET *widget);
```

Description

This service claims the focus for the specified widget. If the widget did no 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_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

```
/* Claim focus for widget “my_widget”. */
status =  gx_system_claim_focus(&my_widget);

/* If status is GX_SUCCESS the focus has been claimed for “my_widget”. */
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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_CALLER_ERROR	(0x11)	Invalid caller of this function

Allowed From

Initialization and threads

Example

```
/* Initialize GUIX. */
status =  gx_system_initialize();

/* If status is GX_SUCCESS, GUIX has been initialized. */
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_language_table_get

Retrieve active language table

Prototype

```
UINT  gx_system_language_table_get(GX_CHAR ****language_table, UINT
*languages_count, UINT *string_count);
```

Description

This service retrieves the active language table.

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)	Successful
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;
UINT       language_count;
UINT       string_count;

/* Retrieve the language table */
status =  gx_system_language_table_get(&language_table, &language_count, &string_count);
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_language_table_set

Assign active language table

Prototype

```
UINT  gx_system_language_table_set(GX_CHAR ***language_table, UINT
languages_count, UINT string_count);
```

Description

This service retrieves the active language table.

Parameters

language_table	Pointer to language table.
languages_count	Number of languages in table.
string_count	Number of string table rows.

Return Values

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

Allowed From

Initialization and threads

Example

```
/* Retrieve the language table */
status =  gx_system_language_table_set(language_table, language_count, string_count);
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_memory_allocator_set

Assign functions for memory allocation, de-allocation

Prototype

```
UINT  gx_system_memory_allocator_set(VOID *(allocate)(ULONG size),
VOID(*release)(VOID *));
```

Description

This service assigns the application supplied function for dynamic memory allocation. This is only required if widget control blocks will be dynamically allocated at runtime, rather then control blocks being statically defined.

Parameters

allocator	Memory allocator function
release	Memory free function

Return Values

GX_SUCCESS	(0x00)	Successful
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;
UINT      language_count;
UINT      string_count;

/* Retrieve the language table */
status =  gx_system_language_table_get(&language_table, &language_count, &string_count);
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_scroll_appearance_get

Get scroll appearance

Prototype

```
UINT  gx_display_scroll_appearance_get(GX_DISPLAY *display,
                       ULONG style,
                      GX_SCROLLBAR_APPEARANCE *return_appearance);
```

Description

This service gets the scrollbar appearance.

Parameters

style	Scrollbar style GX_SCROLLBAR_HORIZONTAL or GX_SCROLLBAR_VERTICAL
return_appearance	Pointer to destination for appearance

Return Values

GX_SUCCESS	(0x00)	Successfull get scrollbar appearance
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
GX_SCROLLBAR_APPEARACE my_appearance;

/* Get the scrollbar appearance. */
status =  gx_display_scroll_appearance_get(display, style, &my_appearance);

/* If status is GX_SUCCESS “my_appearance” now contains the scroll appearance. */
```

See Also

gx_display_scroll_appearance_set

gx_system_scroll_appearance_set

Set scroll appearance

Prototype

```
UINT  gx_display_scroll_appearance_set(GX_DISPLAY *display,
                        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
or	
	GX_SCROLLBAR_VERTICAL
appearance	Pointer to appearance structure initialized with various scrollbar appearance attributes.

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
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
/* Set the scroll appearance. */
status =  gx_display_scroll_appearance_set(display, style, &my_appearance);

/* If status is GX_SUCCESS the scroll appearance has been set. */
```

See Also

gx_display_scroll_appearance_get

gx_system_start

Start GUIX

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

GX_SUCCESS	(0x00)	Successful system start
GX_CALLER_ERROR	(0x11)	Invalid caller of this function

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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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.

Parameters

string_id	String resource ID
return_string	Pointer to string destination pointer

Return Values

GX_SUCCESS	(0x00)	Successful string get
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_string_table_get

Retrieves the string table

Prototype

```
UINT  gx_system_string_table_get(INT language,
                                 GX_CHAR ***string_table,
                                 UINT *get_size);
```

Description

This service retrieves the system string table.

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_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_string_width_get

Get string width

Prototype

```
UINT  gx_system_string_width_get(GX_FONT *font, GX_CHAR *string,
                                 INT string_length,
                                 GX_VALUE *return_width);
```

Description

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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_FONT	(0x16)	Invalid font
GX_INVALID_SIZE	(0x19)	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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get,

gx_system_string_table_set, gx_system_string_width_get,
gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_timer_start

Start timer

Prototype

```
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.

Parameters

owner	Pointer to widget control block
timer_id	ID of timer
initial_ticks	Initial expiration ticks
reschedule_ticks	Periodic expiration ticks

Return Values

GX_SUCCESS	(0x00)	Successful timer start
GX_OUT_OF_TIMERS	(0x04)	No more timers
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)	Timer value(s) not valid

Allowed From

Initialization and threads

Example

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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 for the specified widget.

Parameters

owner	Pointer to widget control block
timer_id	ID of timer

Return Values

GX_SUCCESS	(0x00)	Successful timer stop
GX_NOT_FOUND	(0x09)	Timer ID 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

```
/* 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_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_system_version_string_get

Retrieve GUIX library version string

Prototype

```
UINT  gx_system_version_string_get(GX_CHAR **version);
```

Description

This service retrieves the GUIX library version string.

Parameters

version	Pointer to return string value.

Return Values

GX_SUCCESS	(0x00)	Successful

Allowed From

Initialization and threads

Example

```
GX_CHAR *version;

/* get the library version string. */
status =  gx_system_verrsion_string_get(&version);
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, 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 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 reqested widget ID.
return_search_result	Pointer to destination for widget found

Return Values

GX_SUCCESS	(0x00)	Successful widget search
GX_NOT_FOUND	(0x09)	Widget ID not found
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

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_search(MY_WIDGET_ID, GX_TRUE, NULL, &my_widget);

/* If status is GX_SUCCESS . the search was successful and "my_widget" contains the pointer to the
widget. */
```

See Also

gx_system_dirty_partial_add, gx_system_draw_context_get, gx_system_event_send, gx_system_focus_claim, gx_system_initialize, gx_system_start, gx_system_string_get, gx_system_string_table_get, gx_system_string_table_set, gx_system_string_width_get, gx_system_timer_start, gx_system_timer_stop, gx_system_widget_find

gx_text_button_create

Create text button

Prototype

```
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.

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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* 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

```
UINT  gx_text_button_draw(GX_TEXT_BUTTON *button);
```

Description

This service draws the text button.

Parameters

button	Pointer to text button control block

Return Values

GX_SUCCESS	(0x00)	Successful text button draw
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

```
/* Draw the text button “my_text_button”. */
status =  gx_text_button_draw(&my_text_button);

/* If status is GX_SUCCESS, the text button “my_text_button” was drawn. */
```

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

button	Pointer to text button control block
font_id	Resource ID fo the font

Return Values

GX_SUCCESS	(0x00)	Successfully set the font
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid font id

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 to 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);
```

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 B** 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 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 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
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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_YELLOW_ID,
                                                          GX_COLOR_BLUE_ID);

/* If status is GX_SUCCESS, the text color "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_get

Get text from the thext button

Prototype

```
UINT  gx_text_button_text_get(GX_TEXT_BUTTON *text_button,
                              GX_CHAR **return_text)
```

Description

This service sets the specified string to 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

Allowed From

Initialization and threads

Example

```
/* Get the string from the text button "my_text_button". */
status =  gx_text_button_text_color_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_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 thext button

Prototype

```
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

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

```
/* 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

Set text to the thext button

Prototype

```
UINT  gx_text_button_text_set(GX_TEXT_BUTTON *text_button,
                              GX_CHAR *text)
```

Description

This service sets the specified string to the text button.

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

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_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_utility_ltoa

Convert long integer to ASCII

Prototype

```
VOID  gx_utility_itoa(LONG value, GX_CHAR *return_buffer,
                      UINT seturn_buffer_size);
```

Description

This service converts a long integer value into an ASCII string.

Parameters

value	Long integer value to convert
return_buffer	Destination buffer for ASCII string
return_buffer_size	Size of destination buffer

Return Values

None

Allowed From

All

Example

```
/* Convert “my_value” into an ASCII string. */
gx_utility_ltoa(my_value, my_value_string, sizeof(my_value_string));

/* “my_value_string” 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_cos

Compute cosine

Prototype

```
INT  gx_utility_math_cos(INT angle);
```

Description

This service computes the cosine of the supplied angle.

Parameters

angle	Angle to compute cosine of

Return Values

cosine	Cosine of supplied angle

Allowed From

All

Example

```
/* Compute cosine of "my_angle". */
my_angle_cosine =  gx_utility_math_cos(my_angle);

/* "my_angle_cosine" contains the cosine of  "my_angle". */
```

See Also

gx_utility_ltoa, 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

```
INT  gx_utility_math_sin(INT angle);
```

Description

This service computes the sine of the supplied angle.

Parameters

angle	Angle to compute sine of

Return Values

sine	Sine of supplied angle

Allowed From

All

Example

```
/* Compute sine of "my_angle". */
my_angle_sine =  gx_utility_math_sin(my_angle);

/* "my_angle_sine" contains the sine of  "my_angle". */
```

See Also

gx_utility_ltoa, 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

Compute square root

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_rectangle_grow, gx_utility_rectangle_overlap_detect, gx_utility_rectangle_point_detect, gx_utility_rectangle_shift

gx_utility_pixelmap_rotate

Rotate pixelmap

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.

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

None

Allowed From

All

Example

```
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_utility_pixelmap_simple_rotate

Rotate pixelmap

Prototype

```
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

None

Allowed From

All

Example

```
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_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

```
VOID  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

None

Allowed From

All

Example

```
/* Center "my_inner_rectangle" inside of "my_outer_rectangle". */
gx_utility_rectangle_center(&my_inner_rectangle, &my_outer_rectangle);

/* "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

```
VOID  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 destination to store found center point

Return Values

None

Allowed From

All

Example

```
/* Find center of "my_rectangle"". */
gx_utility_rectangle_center_find(&my_rectangle, &my_center_point);

/* "my_center_point" contains the center point of "my_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_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

```
VOID 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

None

Allowed From

All

Example

```
/* Combine "my_rectangle_a" to "my_rectangle_b". */
gx_utility_rectangle_combine(&my_rectangle_a, &my_rectangle_b);

/* "my_rectangle_a" is the merger of the original "my_rectangle_a" and "my_rectangle_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

```
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

All

Example

```
/* 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

```
VOID  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 coordiante

Return Values

None

Allowed From

All

Example

```
/* Define “my_rectangle”. */
gx_utility_rectangle_define(&my_rectangle, 10, 5, 200, 100);

/* “my_rectangle” 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

```
UINT  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

first_rectangle	First rectangle
second_rectangle	Second rectangle
return_overlap_area	Overlapping rectangle area

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

```
UINT 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

```
/* 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

```
VOID gx_utility_rectangle_resize(GX_RECTANGLE *rectangle,
                                 GX_VALUE adjust);
```

Description

This service increases the size of the rectangle as specified.

Parameters

adjust	Amount to adjust the rectangle

Return Values

None

Allowed From

All

Example

```
/* Adjust "my_rectangle" by increasing 20 pixels on four sides */
gx_utility_rectangle_adjust(&my_rectangle, 20);

/* "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

gx_utility_rectangle_shift

Shift rectangle

Prototype

```
VOID  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

None

Allowed From

All

Example

```
/* Shift  "my_rectangle". */
gx_utility_rectangle_shift(&my_rectangle, 10, 20);

/* At this point "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_vertical_list_children_position

Position children for the vertical list

Prototype

```
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

Allowed From

Initialization and threads

Example

```
/* 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_selecgted_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.

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_WIDGET_SIZE	(0x14)	Invalid widget control block size

GX_INVALID_VALUE	(0x22)	Number of rows not valid

Allowed From

Initialization and threads

Example

```
/* 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

```
UINT  gx_vertical_list_event_process(GX_VERTICAL_LIST *list,
                                     GX_EVENT *event);
```

Description

This service processes an event for the vertical list.

Parameters

list	Vertical list widget control block
event	Pointer to event to process

Return Values

GX_SUCCESS	(0x00)	Successfully processed the vertical list 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

```
/* 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_selected_set

gx_vertical_list_page_index_set

Set starting page index

Prototype

```
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
event	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 pointer
GX_INVALID_VALUE	(0x22)	Invalid value
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* Sets the starting page index of vertical list “my_list” as “top_index”. */
status =  gx_vertical_list_page_index_set(&my_list, &top_index);

/* If status is GX_SUCCESS the starting page index of “m_list” has been set. */
```

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 entry from vertical list

Prototype

```
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_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

```
/* Get the list entry at the current index of vertical list “my_list”. */
status =  gx_vertical_list_selected_index_get(&my_list, &current_list_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_widget_get

Get entry from vertical list

Prototype

```
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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_FAILURE	(0x10)	The selected widget has been scrolled from view.

Allowed From

Initialization and threads

Example

```
/* Get the list entry at the current index of vertical list "my_list". */
status =  gx_vertical_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_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_selected_set

Set entry in vertical list

Prototype

```
UINT  gx_vertical_list_selected_set
                              (GX_VERTICAL_LIST *vertical_list,
                               GX_WIDGET *list_entry);
```

Description

This service sets the list entry at the current list index.

Parameters

vertical_list	Vertical list widget control block
list_entry	Pointer to new list entry

Return Values

GX_SUCCESS	(0x00)	Successfully set the vertical list entry
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

```
/* Set the list entry at the current index of vertical list "my_list". */
status =  gx_vertical_list_selected_set(&my_list, &my_entry);

/* If status is GX_SUCCESS, the current index points to the widget "my_entry". */
```

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_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

GX_SUCCESS	(0x00)	Successfully set the vertical list row count
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* Set the list row count to 20 items. */
status =  gx_vertical_list_total_rows_set(&my_list, 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

```
UINT  gx_vertical_scrollbar_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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_APPEARANCE	(0x25)	Invalid appearance

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);

/* 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

```
UINT  gx_widget_allocate(GX_WIDGET **control_block,
                         ULONG memsize);
```

Description

This service allocates a widget control block, by calling the application defined memory allocation function.

Parameters

control_block	Pointer to returned control block pointer
memsize	Control block size, in bytes

Return Values

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

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_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_attach

Attach widget to its parent

Prototype

```
UINT  gx_widget_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 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_back_attach 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

```
UINT  gx_widget_background_draw(GX_WIDGET *widget);
```

Description

This service performs a solid color fill of a widget background.

Parameters

widget	Pointer to widget to be drawn

Return Values

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

Allowed From

Initialization and threads

Example

```
/* Draw the widget background */
status =  gx_widget_background_draw(widget);

/* If status is GX_SUCCESS the widget background has been drawn. */
```

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

```
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

```
/* 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_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

```
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

```
/* 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_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 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

```
UINT  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

GX_SUCCESS	(0x00)	Successful widget border draw
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_COLOR	(0x15)	Invalid color

Allowed From

Initialization and threads

Example

```
/* Draw border of “my_widget”. */
status =  gx_widget_border_draw(&my_widget, GX_COLOR_BLACK, GX_COLOR_GREEN,
                                             GX_COLOR_BLUE, GX_TRUE);

/* If status is GX_SUCCESS the widget “my_widget” border has been drawn. */
```

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

widget	Pointer to widget
style	Style of border. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.

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
GX_INVALID_STYLE	(0x18)	Invalid style

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_background_set, gx_widget_border_draw, 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_width_get

Get widget border width

Prototype

```
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)	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

```
/* 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

```
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_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

```
/* 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

parent	Pointer to parent widget
child	Pointer to child widget
return_detect	Pointer to destination for detection

Return Values

GX_SUCCESS	(0x00)	Successful widget child detection
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Parent or child widget not valid

Allowed From

Initialization and threads

Example

```
/* 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

Draw widget children

Prototype

```
UINT  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

GX_SUCCESS	(0x00)	Successful widget children draw
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

```
/* Draw children of “my_widget”. */
status =  gx_widget_children_draw(&my_widget);

/* If status is GX_SUCCESS, the child widgets of “my_widget” have been drawn. */
```

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

```
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_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)	Widget border not valid

Allowed From

Initialization and threads

Example

```
/* 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

GX_SUCCESS	(0x00)	Successful color get
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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

```
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_WIDGET_SIZE	(0x14)	Invalid widget control block size
GX_INVALID_WIDGET	(0x12)	Parent widget not valid
GX_INVALID_STYLE	(0x18)	Invalid style
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* 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.

Parameters

widget	Pointer to widget
return_test	Destination for test result

Return Values

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

Allowed From

Initialization and threads

Example

```
/* Test to see if widget "my_widget" is created. */
status =  gx_widget_created_test(&my_widget, &was_created);

/* If status is GX_SUCCESS and "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

```
UINT  gx_widget_delete(GX_WIDGET *widget);
```

Description

This service deletes the widget.

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

Allowed From

Initialization and threads

Example

```
/* 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

```
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

```
/* 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

Draw widget

Prototype

```
UINT  gx_widget_draw(GX_WIDGET *widget);
```

Description

This service draws the widget.

Parameters

widget	Pointer to widget

Return Values

GX_SUCCESS	(0x00)	Successful widget draw
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

```
/* Draw widget “my_widget”. */
status =  gx_widget_draw(&my_widget);

/* If status is GX_SUCCESS the widget “my_widget” has been drawn. */
```

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

```
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

```
/* 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 drawning 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

```
UINT  gx_widget_event_generate(GX_WIDGET *widget, ULONG
                               event_type, LONG value);
```

Description

This service generates an event and sends it to the parent widget The child widgets of a window send events to their parent window.

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

Initialization and 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_widget_event_process

Process widget event

Prototype

```
UINT  gx_widget_event_process(GX_WIDGET *widget, GX_EVENT *event);
```

Description

This 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

widget	Pointer to widget
event	Pointer to event to process

Return Values

GX_SUCCESS	(0x00)	Successful widget event processing
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

```
/* 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

```
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_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

```
/* Use "my_event_process" to process events for widget "my_widget". */
status =  gx_widget_event_process_set(&my_widget, my_event_process);

/* If status is GX_SUCCESS all event processing for widget "my_widget" 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

widget	Pointer to widget
event	Pointer to the event

Return Values

GX_SUCCESS	(0x00)	Successful widget event processing override
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and 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

```
UINT  gx_widget_fill_color_set(GX_WIDGET *widget,
                               GX_RESOURCE_ID normal_color_id,
                               GX_RESOURCE_ID selected_color_id);
```

Description

This service sets the widget background colors.

Parameters

widget	Pointer to widget
normal_color_id	Resource ID of the normal color. **Appendix B** 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 selected color. **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 widget background 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_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* Set background of “my_widget”. */
status =  gx_widget_fill_color_set(&my_widget, GX_COLOR_BLACK_ID, GX_COLOR_GREEN_ID);

/* 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 widget

Prototype

```
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 fount
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

```
/* 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_focus_next

Move focus to next widget in navigation order

Prototype

```
UINT  gx_widget_focus_next(GX_WIDGETG *widget);
```

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

Return Values

GX_SUCCESS	**(**0x00)	focus was moved
GX_FAILURE	**(**0x00)	focus was not moved

Allowed From

Initialization and threads

Example

```
/* Get font for MY_FONT_ID. */
status =  gx_widget_focus_next(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

```
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

```
/* Get font for MY_FONT_ID. */
status =  gx_widget_focus_previous(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

Prototype

```
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)	Successful font get
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

Allowed From

Initialization and threads

Example

```
/* 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_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)	No changes applied
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_INVALID_WIDGET	(0x12)	Widget not valid

Allowed From

Initialization and threads

Example

```
/* 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

```
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

```
/* 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

```
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

```
/* 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_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 by visible widgets.

Parameters

widget	Pointer to widget control block
pixelmap_id	Pixelmap resource ID
return_pixelmap	Pointer to pixelmap destination pointer

Return Values

GX_SUCCESS	(0x00)	Successful pixelmap get
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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 pixemap pointer. */
```

See Also

gx_widget_color_get, gx_widget_font_get

gx_widget_resize

Resize widget

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
GX_INVALID_SIZE	(0x19)	Invalid size

Allowed From

Initialization and threads

Example

```
/* 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
GX_INVALID_COORDINATE	(0x21)	Invalid x or y coordinate

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

```
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

```
/* 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

```
UINT  gx_widget_status_add(GX_WIDGET *widget, ULONG status)
```

Description

This service adds any combination of status flags to the specified widget.

Parameters

widget	Pointer to widget
status	Status to add

Return Values

GX_SUCCESS	(0x00)	Successful widget status 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

```
/* 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

widget	Pointer to widget
return_status	Pointer to the status being returned

Return Values

GX_SUCCESS	(0x00)	Successful widget status 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

```
/* Retreive status flag from widget "my_widget". */
status =  gx_widget_status_get(&my_widget, &status);

/* If status is GX_SUCCESS the status from widget "my_widget" is saved to "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

```
UINT  gx_widget_status_remove(GX_WIDGET *widget, ULONG status)
```

Description

This service removes the specified status flags from the widgets internal status variable.

Parameters

widget	Pointer to widget
status	Status to remove

Return Values

GX_SUCCESS	(0x00)	Successful widget status removal
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_STATUS	(0x26)	Status not valid

Allowed From

Initialization and threads

Example

```
/* Remove status of widget "my_widget". */
status =  gx_widget_status_remove(&my_widget, status_to_remove);

/* 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_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.

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_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer
GX_INVALID_WIDGET	(0x12)	Widget not valid
GX_INVALID_STATUS	(0x26)	Status not valid

Allowed From

Initialization and threads

Example

```
/* 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_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_style_add, gx_widget_style_get, gx_widget_style_remove, gx_widget_style_set, gx_widget_width_get

gx_widget_style_add

Add widget style

Prototype

```
UINT  gx_widget_style_add(GX_WIDGET *widget, ULONG style)
```

Description

This service adds a style to the widget.

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
GX_INVALID_STYLE	(0x18)	Invalid style

Allowed From

Initialization and threads

Example

```
/* Add style to widget “my_widget”. */
status = gx_widget_style_add(&my_widget, GX_STYLE_BORDER_RAISED);

/* If status is GX_SUCCESS the widget “my_widget” style 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_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

```
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)	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

```
/* 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.

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 widget “my_widget” style was removed.*/
```

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

```
UINT  gx_widget_style_remove(GX_WIDGET *widget, ULONG style)
```

Description

This service sets a style to the widget.

Parameters

widget	Pointer to widget
style	Style to set. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.

Return Values

GX_SUCCESS	(0x00)	Successful widget 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_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_width_get

Get widget width

Prototype

```
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

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

Allowed From

Initialization and threads

Example

```
/* 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

```
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_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

```
/* 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

```
/* 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

```
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

```
/* 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_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.

Parameters

window	Pointer to window control block
name	Logical name of window
parent	Pointer to parent widget
style	Window style. **Appendix D** contains pre-defined general styles for all widgets as well as widget-specific styles.
window_id	Application-defined ID of the window
size	Size of the window

Return Values

GX_SUCCESS	(0x00)	Successful window 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 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

```
UINT  gx_window_draw(GX_WINDOW *widget);
```

Description

This service draws a window.

Parameters

widget	Pointer to Window control block

Return Values

GX_SUCCESS	(0x00)	Successful window draw
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

```
/* Draw window "my_window". */
status =  gx_window_draw(&my_window);

/* If status is GX_SUCCESS window "my_window" has been drawn. */
```

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 *widget, GX_EVENT *event);
```

Description

This service processes an event for this window.

Parameters

widget	Pointer to Window control block
event	Pointer to event to process

Return Values

GX_SUCCESS	(0x00)	Successful window event processing
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

```
/* Process “my_event” for window “my_window”. */
status =  gx_window_event_process(&my_window, &my_event);

/* If status is GX_SUCCESS the event for window “my_window” has been processed. */
```

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_root_create

Create 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 and attaches the root window to a canvas. The root window is the parent of all visible windows on a given canvas, and is responsible for redrawing the background when no application window covers some portion of the canvas. Changing the background canvas color is therefore actually done by changing the fill color of the root window associated with that canvas. One root window must be created for each canvas defined by the system.

When the GUIX Studio function gx_studio_display_configure is invoked, the canvas and root window for the requested display are automatically created by the gx_studio_display_configure function. However, the application software is free to create other canvases and other root windows for those canvases.

Parameters

root_window	Pointer to root window control block
name	Name of the root window
canvas	Canvas this root window belongs to
style	Style of the root window
id	User-specified root window ID
size	Window size

Return Values

GX_SUCCESS	(0x00)	Successful root window find
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* Create a root window */
GX_WINDOW_ROOT my_root;
GX_RECTANGLE size;

/* create a VGA size root window on main canvas */
gx_utility_rectangle_define(&size, 0, 0, 639, 479);
status =  gx_window_root_create(&my_root, "main root window", main_canvas, GX_STYLE_NONE,
0, &size);

/* 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_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_delete

Destroy a root window

Prototype

```
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)	Successful root window find
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
/* 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

```
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)	Successful root window find
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
GX_PTR_ERROR	(0x07)	Invalid pointer

Allowed From

Initialization and threads

Example

```
status =  gx_window_root_event_process(&root_window, &my_event);

/* If status is GX_SUCCESS the event "my_event" is processed by the roont 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_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

widget	Pointer to widget control block
return_root_window	Pointer to destination for found root window

Return Values

GX_SUCCESS	(0x00)	Successful root window find
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

```
/* 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 type,
                               GX_SCROLL_INFO *return_scroll_info);
```

Description

This service gets the window scroll information.

Parameters

window	Pointer to window
type	GX_SCROLLBAR_HORIZONTAL or GX_SCROLLBAR_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. The GX_SCROLL_INFO structure has the following members:

GX_VALUE gx_scroll_value	current scroll value
GX_VALUE gx_scroll_mimimum	minimum scroll value
GX_VALUE gx_scroll_maximum	maximum scroll value
GX_VALUE gx_scroll_visible	window visible area
GX_VALUE gx_scroll_increment	scroll delta value

Return Values

GX_SUCCESS	(0x00)	Successful window scroll info 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_TYPE	(0x1B)	Invalid type

Allowed From

Initialization and threads

Example

```
/* 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

Find window scrollbar

Prototype

```
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_SCROLLBAR_HORIZONTAL or GX_SCROLLBAR_VERTICAL
return_scrollbar	Pointer to destination for scrollbar

Return Values

GX_SUCCESS	(0x00)	Successful window scrollbar find
GX_NOT_FOUND	(0x09)	Scrollbar not found
GX_CALLER_ERROR	(0x11)	Invalid caller of this function
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

```
/* 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

```
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_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

```
/* 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
GX_INVALID_RESOURCE_ID	(0x22)	Invalid resource ID

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 physcial 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 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 implementaton 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:

```
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 addtional 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_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, INTy1, 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, INTy1, 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, INTy1, 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, INTy1, INT x2, INT y2): This is a pointer to a function to implement generic anti-aliased 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_horizonal_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_vertical_line_draw)(GX_DRAW_CONTECT *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_horizonal_pattern_line_draw)(GX_DRAW_CONTECT *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_CONTECT *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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

VOID (*gx_display_driver_circle_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r): Pointer to a function to draw a circle.

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.

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.

VOID (*gx_display_driver_anti_aliased_arc_draw)(GX_DRAW_CONTEXT *context, INT xcenter, INT ycenter, UINT r, INT start_angle, INTend_angle) : Pointer to a function to draw an anti-aliased arc.

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.

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.

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.

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.

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.

VOID (*gx_display_driver_8bit_glyph_draw)(GX_DRAW_CONTEXT *context, GX_RECTANGLE *draw_area, GX_POINT *map_offset, const GX_GLYPH *glyph): Pointer to function to draw one 8-bit aliased text glyph to the canvas using the brush of the current drawing context.

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.

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.

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 dimentions specific to this implemenation.  */
#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 displayl. */
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[] = {
    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 Pre-Defined Colors

Color	Value
GX_COLOR_BLACK	**0x00000000**
GX_COLOR_RED	**0x00b80000**
GX_COLOR_GREEN	**0x0000bc00**
GX_COLOR_BROWN	**0x00b8bc00**
GX_COLOR_BLUE	**0x000000b8**
GX_COLOR_MAGENTA	**0x00b800b8**
GX_COLOR_CYAN	**0x0000bcb8**
GX_COLOR_LIGHTGRAY	**0x00c0c0c0**
GX_COLOR_DARKGRAY	**0x00808080**
GX_COLOR_LIGHTRED	**0x00ff0000**
GX_COLOR_LIGHTGREEN	**0x0000ff00**
GX_COLOR_YELLOW	0**x00ffff00**
GX_COLOR_LIGHTBLUE	**0x000000ff**
GX_COLOR_LIGHTMAGENTA	0**x00ff00ff**
GX_COLOR_LIGHTCYAN	**0x0000ffff**
GX_COLOR_WHITE	**0x00ffffff**

Appendix B: GUIX Pre-Defined Color Resource IDs

Color	Value
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

Appendix C: GUIX Color Formats

Color	Value
GX_COLOR_FORMAT_MONOCHROME	1
GX_COLOR_FORMAT_MONOCHROME_INVERTED	2
GX_COLOR_FORMAT_2BIT_4GRAY	3
GX_COLOR_FORMAT_2BIT_GRAY_INVERTED	4
GX_COLOR_FORMAT_4BIT_GRAY	5
GX_COLOR_FORMAT_4BIT_GRAY_INVERTED	6
GX_COLOR_FORMAT_4BIT_VGA	7
GX_COLOR_FORMAT_8BIT_GRAY	8
GX_COLOR_FORMAT_8BIT_GRAY_INVERTED	9
GX_COLOR_FORMAT_8BIT_PALETTE	10
GX_COLOR_FORMAT_8BIT_PACKED_PIXEL	11
GX_COLOR_FORMAT_15BIT_BGR	12
GX_COLOR_FORMAT_15BIT_RGB	13
GX_COLOR_FORMAT_16BIT_RGB	14
GX_COLOR_FORMAT_16BIT_ARGB	15
GX_COLOR_FORMAT_16BIT_BGRA	16
GX_COLOR_FORMAT_16BIT_BGR	17
GX_COLOR_FORMAT_24BIT_RGB	18
GX_COLOR_FORMAT_24BIT_BGR	19
GX_COLOR_FORMAT_24BIT_XRGB	20
GX_COLOR_FORMAT_24BIT_BGRX	21
GX_COLOR_FORMAT_32BIT_ARGB	22
GX_COLOR_FORMAT_32BIT_RGBA	23
GX_COLOR_FORMAT_32BIT_ABGR	24
GX_COLOR_FORMAT_32BIT_BGRA	25

Appendix D: GUIX Widget Styles

Style	Value
General Styles:	
GX_STYLE_BORDER_NONE	`0x00000000`
GX_STYLE_BORDER_RAISED	`0x00000001`
GX_STYLE_BORDER_RECESSED	`0x00000002`
GX_STYLE_BORDER_THIN	`0x00000004`
GX_STYLE_BORDER_THICK	`0x00000008`
GX_STYLE_BORDER_MASK	`0x0000000f`
GX_STYLE_TRANSPARENT	`0x10000000`
GX_STYLE_DRAW_SELECTED	`0x20000000`
GX_STYLE_ENABLED	`0x40000000`
Additional Button Styles:	
GX_STYLE_BUTTON_PUSHED	`0x00000010`
GX_STYLE_BUTTON_TOGGLE	`0x00000020`
GX_STYLE_BUTTON_RADIO	`0x00000040`
GX_STYLE_BUTTON_EVENT_ON_PUSH	`0x00000080`
GX_STYLE_BUTTON_REPEAT	`0x00000100`
Additional List Styles:	
GX_STYLE_CENTER_SELECTED	`0x00000010`
GX_STYLE_WRAP	`0x00000020`
GX_STYLE_FLICKABLE	`0x00000040`
Additional Pixelmap Button and Icon Button Styles:	
GX_STYLE_HALIGN_CENTER	`0x00010000`
GX_STYLE_HALIGN_LEFT	`0x00020000`
GX_STYLE_HALIGN_RIGHT	`0x00040000`
GX_STYLE_VALIGN_CENTER	`0x00080000`
GX_STYLE_VALIGN_TOP	`0x00100000`
GX_STYLE_VALIGN_BOTTOM	`0x00200000`
GX_PIXELMAP_HALIGN_MASK	`0x00070000`
GX_PIXELMAP_VALIGN_MASK	`0x00380000`

Additional Slider Styles:

GX_STYLE_SHOW_NEEDLE	0x00000200
GX_STYLE_SHOW_TICKMARKS	0x00000400
GX_STYLE_SLIDER_VERTICAL	0x00000800

Additional Pixelmap Slider Styles:

GX_STYLE_TILE_BACKGROUND	0x00001000

Additional Progress Bar Styles:

GX_STYLE_SHOW_PERCENT	0x00000010

Additional Text Alignment Styles:

GX_STYLE_TEXT_LEFT	0x00001000
GX_STYLE_TEXT_RIGHT	0x00002000
GX_STYLE_TEXT_CENTER	0x00004000
GX_STYLE_TEXT_ALIGNMENT_MASK	0x00007000
GX_STYLE_TEXT_COPY	0x00008000

Additional Cursor Styles:

GX_STYLE_ CURSOR_BLINK	0x00000040
GX_STYLE_ CURSOR_ALWAYS	0x00000080

Additional Text Input Styles:

GX_STYLE_TEXT_INPUT_NOTIFY_ALL	0x00000100

Additional Window Styles:

GX_STYLE_TILE_WALLPAPER	0x00040000
GX_STYLE_AUTO_HSCROLL	0x00100000
GX_STYLE_AUTO_VSCROLL	0x00200000

Additional Scrollbar Styles:

GX_SCROLLBAR_BACKGROUND_TILE	0x00010000
GX_SCROLLBAR_RELATIVE_THUMB	0x00020000
GX_SCROLLBAR_END_BUTTONS	0x00040000
GX_SCROLLBAR_VERTICAL	0x01000000

GX_SCROLLBAR_HORIZONTAL	**0x02000000**

Appendix E: GUIX Events

Event	Value
GX_EVENT_TERMINATE	1
GX_EVENT_REDRAW	2
GX_EVENT_SHOW	3
GX_EVENT_HIDE	4
GX_EVENT_RESIZE	5
GX_EVENT_SLIDE	6
GX_EVENT_FOCUS_GAINED	7
GX_EVENT_FOCUS_LOST	8
GX_EVENT_HORIZONTAL_SCROLL	9
GX_EVENT_VERTICAL_SCROLL	10
GX_EVENT_TIMER	11
GX_EVENT_PEN_DOWN	12
GX_EVENT_PEN_UP	13
GX_EVENT_PEN_DRAG	14
GX_EVENT_KEY_DOWN	15
GX_EVENT_KEY_UP	16
GX_EVENT_CLOSE	17
GX_EVENT_DESTROY	18
GX_EVENT_SLIDER_VALUE	19
GX_EVENT_TOGGLE_ON	20
GX_EVENT_TOGGLE_OFF	21
GX_EVENT_RADIO_SELECT	22
GX_EVENT_RADIO_DESELECT	23
GX_EVENT_CLICKED	24
GX_EVENT_LIST_SELECT	25
GX_EVENT_VERTICAL_FLICK	26
GX_EVENT_HORIZONTAL_FLICK	28
GX_EVENT_MOVE	29
GX_EVENT_PARENT_SIZED	30
GX_EVENT_CLOSE_POPUP	31
GX_EVENT_ZOOM_IN	32
GX_EVENT_ZOOM_OUT	33
GX_EVENT_LANGUAGE_CHANGE	34
GX_EVENT_RESOURCE_CHANGE	35
GX_EVENT_ANIMATION_COMPLETE	36
GX_EVENT_SPRITE_COMPLETE	37
GX_EVENT_TEXT_EDITED	40
GX_EVENT_TX_TIMER	41
GX_EVENT_FOCUS_NEXT	42
GX_EVENT_FOCUS_PREVIOUS	43
GX_EVENT_FOCUS_GAIN_NOTIFY	44
GX_EVENT_SELECT	45
GX_EVENT_DESELECT	46

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:

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 GUX.

A complete example of a generic RTOS binding is implemented in the filed 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:

```
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:

```
typedef struct GX_GLYPH_STRUCT
{
    GX_CONST 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 opague). 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.

Index

alpha channel............... 18, 27, 29
ANSI C 1, 2, 3, 5, 39, 49
anti-aliasing............ 2, 11, 36, 110
API
 call 9, 39, 49
 drawing 35
 GUIX API function.... 13, 14, 17, 25
 object creation 14
 service 42, 286
ASCII...................... 7, 24, 58, 313
blend pixel 35
block move 35
buffer
 composite 21
 frame 18, 19, 20, 22, 26, 27, 37, 50, 81, 84, 89, 98, 424, 426, 428, 431
 local frame 19, 20
 ping-pong........................ 20, 21
buffer toggle 35
canvas
 alpha channel 11, 27, 28
 blend.................................... 28
 control block 11, 27, 78, 81, 82, 83, 84, 85, 89, 98
 creation........................... 11, 27
 drawing .. 27, 28, 54, 81, 84, 85, 87, 89
 GUIX canvas component 11, 26
 managed............ 26, 27, 78, 81
 memory...... 22, 35, 37, 82, 424, 427, 428
 object 27
 overlay 27
 simple 26
 Z-order.................................. 27
color depth .. 2, 18, 28, 35, 36, 78, 110, 424, 426, 427, 428
color format 18, 23, 128, 424, 426, 427, 428
compiler............... xvii, 3, 9, 21, 22
configuration....................... 9, 426
data type .... xvii, 9, 42
demo thread 431, 432
dirty list 15
display driver xv, 2, 11, 18, 25, 27, 29, 37, 135, 137, 424, 428, 430
display memory architecture 19
draw basic line.......................... 35
draw horizontal line 35
draw pixelmap . 35, 54, 56, 90, 92, 94, 188, 189, 195, 201
draw polygon 35
draw quadragon 35
draw rectangle.......................... 35
draw text.................................. 35
draw vertical line....................... 35
event notification 12, 42, 43, 51
event processing ... 11, 14, 15, 17, 44, 45, 51, 59, 104, 255, 280, 281, 371, 373, 374, 375, 413
event queue 13, 14, 15, 17, 24, 25, 42, 284, 293
global7, 10, 22, 23, 24, 27, 29, 40, 49, 82
GRAM 19, 21
GUIX components.................... 39
GUIX objects 14
GUIX system mutex 24
GUIX thread .. 7, 9, 13, 14, 17, 21, 22, 25, 37, 44, 46, 51
GUIX widget 13, 14, 23, 33, 37, 39, 49
hardware initialization............... 35
input drivers........................ 13, 17
LCD display............................ 424
memory
 architecture 19
 buffer 18
 canvas 22, 35, 37, 82, 424, 427, 428
 constraints 19
 dynamic 22
 frame buffer 18

object. 3, 7, 14, 23, 27, 29, 39, 48, 49
overlay................................ 21, 27
periodic processing 7
pixelmaps.... 22, 29, 30, 105, 106, 191, 192, 196, 198, 199, 203, 204, 227
queue, event ... 13, 14, 15, 17, 24, 25, 42, 284, 293
runtime library 8
screen control block 11, 29
screen driver 28, 36, 47, 78
screen refresh 14, 21, 25, 47
screens........................... 2, 13, 14
scrolling........ 49, 50, 51, 419, 428
setup . 6, 133, 134, 135, 138, 139, 424
skinning.................... 2, 11, 32, 33
stack size 9, 14, 22
static text.................................. 31
string table.......... 30, 58, 295, 432
system error handling......... 11, 26
ThreadX ii, xv, xvii, xviii, 1, 2, 3, 4, 5, 7, 8, 9, 13, 18, 25, 431
ThreadX timer 7, 13, 18, 25
tile pixelmap 35, 54, 95
timer tick................................... 18
user interface ... 3, 4, 6, 11, 13, 14
utility component 12, 51, 53
version_id......................... xviii, 10
widget component 11, 39
widget control block...... 12, 39, 40
widget creation 12, 39
widget defaults 11, 30
window
  background........................... 50
  border 49
  children 37, 49, 50
  component............................ 48
  control block........................ 410
  event handler........................ 51
  GUIX....... 14, 17, 25, 48, 49, 50
  object.................................... 49
  processing 48
  root 8, 15, 36, 37, 49, 50, 60, 302, 414, 415, 416, 417, 418, 431, 432, 433

Renesas Synergy^{TM} Platform

User’s Manual: Software

Publication Date: Rev.5 October, 2015

Renesas Synergy^{TM} Platform

User’s Manual: Software

Renesas Electronics Corporation

R11UM0003EU0500