This chapter describes the errors, warnings, and notices that can be posted by QuickDraw GX functions, and how you can manipulate them. In addition, this chapter describes how you can use application-defined handlers to provide alternative or complementary processing of errors, warnings, and notices. The reference sections of the *Inside Macintosh: QuickDraw GX* books list the errors, warnings, and notices for each function that they describe.

Before reading this chapter, you should be familiar with the information in the chapter "Introduction to QuickDraw GX" in *Inside Macintosh: QuickDraw GX Objects*.

The errors, warnings, and notices and their related functions that are discussed in this chapter pertain to the graphic and typography parts of QuickDraw GX and do not, in general, apply to printing. For more information on printing errors, see *Inside Macintosh: QuickDraw GX Printing* and *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

This chapter starts by introducing you to the errors, warnings, and notices provided in the debugging and non-debugging versions of QuickDraw GX. It then shows you how to use their related functions to

- obtain the QuickDraw GX errors, warnings, and notices posted
- change the QuickDraw GX errors, warnings, and notices posted
- ignore QuickDraw GX warnings and notices
- install application-defined error, warning, and notice handlers

This chapter also contains reference information for all data types, application-defined handlers, and functions associated with QuickDraw GX errors, warnings, and notices.



# About QuickDraw GX Errors, Warnings, and Notices

QuickDraw GX posts **errors**, **warnings**, or **notices**, depending upon the severity of the problem that was detected when your application was running. The three types of QuickDraw GX execution problems are

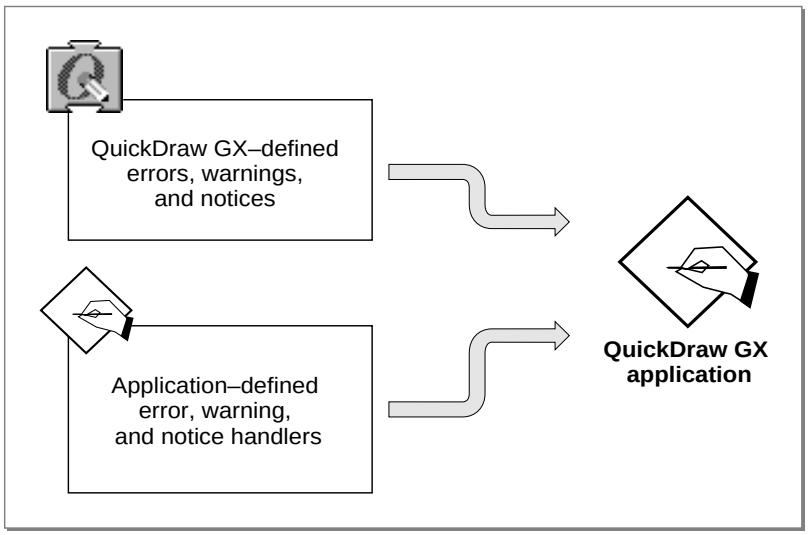
- **Errors.** QuickDraw GX posts errors whenever a function in your application is unable to execute. An error indicates that an operation cannot continue. Execution terminates at the nonexecutable function. When an error is posted inside a QuickDraw GX function, the function returns immediately with a function result (if any) of 0 or `nil`.
- **Warnings.** QuickDraw GX posts warnings whenever your application executes a function that doesn't provide the result that you expect. Execution continues internally, as if the warning had not been posted.
- **Notices.** QuickDraw GX posts notices to alert you to the fact that it has performed an unnecessary or redundant function. Execution continues as if the notice had not been posted. Graphics notices are posted only in the debugging version of QuickDraw GX.

In addition to the **posting** of errors, warnings, and notices, QuickDraw GX supports application-defined error, warning, and notice **handlers.** You can use your own handlers or QuickDraw GX's errors, warnings, and notices either separately or together.

To obtain errors, warnings, and notices, either check for QuickDraw GX errors, warnings, and notices or install your application's error, warning, and notice handlers. The use of error, warning, and notice handlers is a simple and efficient method of managing errors, warnings, and notices. Error handlers are described in the section "Installing an Error, Warning, or Notice Handler" beginning on page 3-40.

Figure 3-1 shows the relationship of the two problem-management approaches.

**Figure 3-1** QuickDraw GX and application-defined error, warning, and notice management




There are two versions of QuickDraw GX.

- **Non-debugging version.** This version of QuickDraw GX is intended for debugged applications used by the end user. The number of QuickDraw GX errors and warnings is limited. Notices are not posted. This version of QuickDraw GX is smaller and faster than the debugging version.
- **Debugging version.** This version of QuickDraw GX is intended for developers that are writing and debugging new applications. This version provides an extensive set of QuickDraw GX errors, warnings, and notices to assist in debugging and optimizing the performance of your application. Special functions are provided to assist in the posting, utilization, and control of debugging errors.

To determine if the debugging or non-debugging version is installed, use the `Gestalt` function described in the chapter "QuickDraw GX and the Macintosh Environment" in this book.

QuickDraw GX posts most errors and warnings only in the debugging version. The non-debugging version posts errors and warnings if the error could not be anticipated at compile time—for example, running out of memory or disk space. You should correct application problems that result in errors and warnings while developing your application. The non-debugging version does not include most of the errors and warnings that the debugging version provides.

QuickDraw GX non-debugging and debugging errors are defined by the `gxGraphicErrors` enumeration given in the section "Errors" beginning on page 3-42. QuickDraw GX non-debugging and debugging warnings are defined by the `gxGraphicWarnings` enumeration given in the section "Warnings" beginning on page 3-50. QuickDraw GX debugging notices are defined by the `gxGraphicNotices` enumeration given in the section "Notices" beginning on page 3-53.

## Non-Debugging Version

When you install the non-debugging version, QuickDraw GX provides a reduced set of errors and warnings. Since the amount of testing is less, the non-debugging version of QuickDraw GX runs significantly faster than the debugging version. Use the non-debugging version for debugged applications that you have extensively tested using the debugging version of QuickDraw GX.

When the non-debugging version is installed and corrupt data is used, drawings may execute with undesirable results, including crashes, without the posting of errors and warnings. With other execution problems, the application may not crash, but the drawing may not yield the expected result.

In the non-debugging version, typical problem messages indicate that there is insufficient memory, insufficient storage space, or that the required fonts are not installed. If problems persist, you can always install the debugging version to assist in the analysis of the errors that are occurring.

For a complete list of errors, please see the graphics `errors.h` interface file. The many notices, warnings, and errors defined between `#ifdef debugging` and `#endif` in that file are available only with the debugging version.

A debugged application should encounter only errors like

```
out_of_memory
not_enough_memory_for_graphics_client_heap
graphics_client_too_small
could_not_create_backing_store
```

A debugged application should encounter warnings like

```
character_substitution_occurred
map_shape_out_of_range
move_shape_out_of_range
scale_shape_out_of_range
```

```
rotate_shape_out_of_range
skew_shape_out_of_range
map_transform_out_of_range
move_transform_out_of_range
scale_transform_out_of_range
rotate_transform_out_of_range
skew_transform_out_of_range
```

Both the debugging and non-debugging versions of QuickDraw GX provide a debugging utility called GraphicsBug. This versatile utility allows you to examine the details of each graphics object. GraphicsBug is described in the chapter "QuickDraw GX Debugging" in this book.

## Errors

This section describes the errors that may be posted by both the debugging and non-debugging versions of QuickDraw GX. These errors can be grouped into the following categories:

- fatal errors
- internal errors
- recoverable errors
- font management errors
- bad parameter errors
- implementation limit errors
- font scaler errors

Each QuickDraw GX error has an error number and an error name. Table 3-1 gives the non-debugging error number ranges.

**Table 3-1** Non-debugging error number ranges

| Number | Name |
|---|---|
| –27999 | `gxFirstSystemError` |
| –27999 | `gxFirstFatalError` |
| –27951 | `gxLastFatalError` |
| –27950 | `gxFirstNonFatalError` |
| –27900 | `gxFirstFontScalerError` |
| –27851 | `gxLastFontScalerError` |
| –27850 | `gxFirstParameterError` |
| –27800 | `gxFirstImplementationLimitError` |

QuickDraw GX **fatal errors** terminate operation and automatically call the `GXExitGraphics` function. Control returns to the calling application after the error is posted. If the function that caused the error returns a function result, its value is either 0 or `nil`. Table 3-2 lists fatal errors. They are included in both the debugging and non-debugging versions of QuickDraw GX.

**Table 3-2** Fatal errors

| Number | Name |
|---|---|
| –27999 | `out_of_memory` |
| –27998 | `internal_fatal_error` |
| –27997 | `no_outline_font_found` |
| –27996 | `not_enough_memory_for_graphics_client_heap` |
| –27995 | `could_not_create_backing_store` |

QuickDraw GX nonfatal **internal errors** indicate damaged files, memory problems, or incorrect implementation of QuickDraw GX. Table 3-3 lists the internal errors.

**Table 3-3** Internal errors

| Number | Name |
|---|---|
| –27950 | `internal_error` |
| –27949 | `internal_font_error` |
| –27948 | `internal_layout_error` |

Table 3-4 lists the QuickDraw GX recoverable errors.

**Table 3-4** Recoverable errors

| Number | Name |
|---|---|
| –27946 | `could_not_dispose_backing_store` |
| –27945 | `unflattening_interrupted_by_client` |

Table 3-5 lists the QuickDraw GX **font management errors.**

**Table 3-5** Font management errors

| Number | Name |
|---|---|
| –27944 | font_cannot_be_changed |
| –27943 | illegal_font_parameter |

Table 3-6 lists the QuickDraw GX **font scaler errors.**

**Table 3-6** Font scaler errors

| Number | Name |
|---|---|
| –27900 | null_font_scaler_context |
| –27899 | null_font_scaler_input |
| –27988 | invalid_font_scaler_context |
| –27897 | invalid_font_scaler_input |
| –27896 | invalid_font_scaler_font_data |
| –27895 | font_scaler_newblock_failed |
| –27894 | font_scaler_getfonttable_failed |
| –27893 | font_scaler_bitmap_allocation_failed |
| –27892 | font_scaler_outline_allocation_failed |
| –27891 | required_font_scaler_table_missing |
| –27890 | unsupported_font_scaler_outline_format |
| –27889 | unsupported_font_scaler_stream_format |
| –27888 | unsupported_font_scaler_font_format |
| –27887 | font_scaler_hinting_error |
| –27886 | font_scaler_rasterizer_error |
| –27885 | font_scaler_internal_error |
| –27884 | font_scaler_invalid_matrix |
| –27883 | font_scaler_fixed_overflow |
| –27882 | font_scaler_api_version_mismatch |
| –27881 | font_scaler_streaming_aborted |
| –27880 | unknown_font_scaler_error |

QuickDraw GX posts **bad parameter errors** when a required parameter is out of range, invalid, or is passed with the value of `nil`. Table 3-7 lists bad parameter errors.

**Table 3-7** Bad parameter errors

| Number | Name |
|---|---|
| –27850 | `parameter_is_nil` |
| –27849 | `shape_is_nil` |
| –27848 | `style_is_nil` |
| –27847 | `transform_is_nil` |
| –27846 | `ink_is_nil` |
| –27845 | `transferMode_is_nil` |
| –27844 | `color_is_nil` |
| –27843 | `colorProfile_is_nil` |
| –27842 | `colorSet_is_nil` |
| –27841 | `spoolProcedure_is_nil` |
| –27840 | `tag_is_nil` |
| –27839 | `type_is_nil` |
| –27838 | `mapping_is_nil` |
| –27837 | `invalid_viewDevice_reference` |
| –27836 | `invalid_viewGroup_reference` |
| –27835 | `invalid_viewPort_reference` |

QuickDraw GX posts **implementation limit errors** to indicate that the size or number exceeds the size or number supported by the current version of QuickDraw GX. Table 3-8 lists the implementation limit errors.

**Table 3-8** Implementation limit errors

| Number | Name |
|---|---|
| –27800 | `number_of_contours_exceeds_implementation_limit` |
| –27799 | `number_of_points_exceeds_implementation_limit` |
| –27798 | `size_of_polygon_exceeds_implementation_limit` |
| –27797 | `size_of_path_exceeds_implementation_limit` |
| –27796 | `size_of_text_exceeds_implementation_limit` |
| –27795 | `size_of_bitmap_exceeds_implementation_limit` |
| –27794 | `number_of_colors_exceeds_implementation_limit` |
| –27793 | `procedure_not_reentrant` |

## Warnings

This section describes the warnings that the debugging and non-debugging versions of QuickDraw GX may post. These errors can be grouped into the following categories:

- stack, heap, and object warnings
- result is out of range warnings
- parameter is out of range warnings
- font scaler warnings
- unexpected result warnings
- storage warnings

Each QuickDraw GX warning has a unique **warning number** and **warning name.** Table 3-9 gives the non-debugging warning number ranges.

**Table 3-9** Non-debugging warning number ranges

| Number | Description |
|---|---|
| –26999 | `gxFirstSystemWarning` |
| –26950 | `gxFirstResultOutOfRangeWarning` |
| –26900 | `gxFirstParameterOutOfRangeWarning` |
| –26850 | `gxFirstFontScalerWarning` |

QuickDraw GX **overflow warnings** occur when the number of warnings that have been added to the warning or notice stack exceeds the current implementation limit. An **underflow warning** occurs when the `GXPopGraphicsNotice` or `GXPopGraphicsWarning` function attempts to remove an error or warning on its ignore stack and there is no error or warning to remove. This topic is discussed in the section "Ignoring Warnings and Notices" beginning on page 3-37.

Table 3-10 lists QuickDraw GX stack, heap, and object warnings.

**Table 3-10** Stack, heap, and object warnings

| Number | Name |
|---|---|
| –26999 | `warning_stack_underflow` |
| –26998 | `warning_stack_overflow` |
| –26997 | `notice_stack_underflow` |
| –26996 | `notice_stack_overflow` |
| –26995 | `about_to_grow_heap` |
| –26994 | `about_to_unload_objects` |

QuickDraw GX **result out of range warnings** occur when a function result is out of the usable or defined QuickDraw boundaries. Table 3-11 lists result out of range warnings.

**Table 3-11** Result out of range warnings

| Number | Name |
|---|---|
| –26950 | `map_shape_out_of_range` |
| –26949 | `move_shape_out_of_range` |
| –26948 | `scale_shape_out_of_range` |
| –26947 | `rotate_shape_out_of_range` |
| –26946 | `skew_shape_out_of_range` |
| –26945 | `map_transform_out_of_range` |
| –26944 | `move_transform_out_of_range` |
| –26943 | `scale_transform_out_of_range` |
| –26942 | `rotate_transform_out_of_range` |
| –26941 | `skew_transform_out_of_range` |
| –26940 | `map_points_out_of_range` |

QuickDraw GX **parameter out of range warnings** occur when a function parameter is out of the usable range. Table 3-12 lists parameter out of range warnings.

**Table 3-12** Parameter out of range warnings

| Number | Name |
|---|---|
| –26900 | `contour_out_of_range` |
| –26899 | `index_out_of_range_in_contour` |
| –26898 | `picture_index_out_of_range` |
| –26897 | `color_index_requested_not_found` |
| –26896 | `colorSet_index_out_of_range` |
| –26895 | `index_out_of_range` |
| –26894 | `count_out_of_range` |
| –26893 | `length_out_of_range` |
| –26892 | `font_table_index_out_of_range` |
| –26891 | `font_glyph_index_out_of_range` |
| –26890 | `point_out_of_range` |
| –26889 | `profile_response_out_of_range` |

Table 3-13 lists QuickDraw GX **font scaler warnings.**

**Table 3-13** Font scaler warnings

| Number | Name |
|---|---|
| –26850 | `font_scaler_no_output` |
| –26849 | `font_scaler_fake_metrics` |
| –26848 | `font_scaler_fake_linespacing` |
| –26847 | `font_scaler_glyph_substitution` |
| –26846 | `font_scaler_no_kerning_applied` |

Table 3-14 lists QuickDraw GX **unexpected result warnings.**

**Table 3-14** Unexpected result warnings

| Warning number | Warning name |
|---|---|
| –26845 | `character_substitution_took_place` |
| –26844 | `unable_to_bounds_on_multiple_devices` |
| –26843 | `font_language_not_found` |
| –26842 | `font_not_found_during_unflattening` |

Table 3-15 lists QuickDraw GX data stream **storage warnings.**

**Table 3-15** Storage warnings

| Number | Name |
|---|---|
| –26841 | `unrecognized_stream_version` |
| –26840 | `bad_data_in_stream` |

## Debugging Version

When you install the debugging version, QuickDraw GX posts errors, warnings, and notices in addition to those posted by the non-debugging version. The debugging analysis and resulting number of errors, warnings, and notices posted is far more extensive than can be provided by the non-debugging version of QuickDraw GX. As a result, the debugging version executes significantly slower than the non-debugging version.

The errors and warnings posted by both the debugging and non-debugging versions of QuickDraw GX are listed in the sections “Errors” beginning on page 3-6 and “Warnings” beginning on page 3-10. The errors, warnings, and notices described in the following sections are posted only in the debugging version of QuickDraw GX.

The debugging version also provides a number of useful functions that you can use to analyze your code and that assist in determining the cause of a wide variety of problems. These are described in the section “Using Errors, Warnings, and Notices” beginning on page 3-30.

The debugging version of QuickDraw GX also provides validation functions and GraphicsBug so that you can examine the details of each graphics object. These are described in the chapter “QuickDraw GX Debugging” in this book.

## Errors

This section describes the errors that the debugging version of QuickDraw GX may post. QuickDraw GX debugging errors can be grouped into the following categories:

- internal errors
- font parameter errors
- bad parameter errors
- restricted access errors
- wrong type or bad reference errors
- validation errors

Table 3-16 gives the debugging error number range.

**Table 3-16** Debugging error number range

| Number | Name |
|---|---|
| –27700 | `gxFirstSystemDebuggingError` |
| –27000 | `gxLastSystemError` |

Table 3-17 lists the internal debugging errors.

**Table 3-17** Internal debugging errors

| Number | Name |
|---|---|
| –27700 | `functionality_unimplemented` |
| –27699 | `clip_to_frame_shape_unimplemented` |

Table 3-18 lists the font parameter debugging errors.

**Table 3-18** Font parameter debugging errors

| Number | Name |
|---|---|
| –27698 | `illegal_font_storage_type` |
| –27697 | `illegal_font_storage_reference` |
| –27696 | `illegal_font_attributes` |

QuickDraw GX bad parameter errors are posted when a required parameter is out of range, invalid, or is passed with the value of `nil`. Table 3-19 lists the bad parameter debugging errors.

**Table 3-19** Bad parameter debugging errors

| Number | Name |
|---|---|
| –27695 | `parameter_out_of_range` |
| –27694 | `inconsistent_parameters` |
| –27693 | `index_is_less_than_zero` |
| –27692 | `index_is_less_than_one` |
| –27691 | `count_is_less_than_zero` |
| –27690 | `count_is_less_than_one` |
| –27689 | `contour_is_less_than_zero` |
| –27688 | `length_is_less_than_zero` |
| –27687 | `invalid_client_reference` |
| –27686 | `invalid_graphics_heap_start_pointer` |
| –27685 | `invalid_nongraphic_globals_pointer` |
| –27684 | `colorSpace_out_of_range` |
| –27683 | `pattern_lattice_out_of_range` |
| –27682 | `frequency_parameter_out_of_range` |
| –27681 | `tinting_parameter_out_of_range` |
| –27680 | `method_parameter_out_of_range` |
| –27679 | `space_may_not_be_indexed` |
| –27678 | `glyph_index_too_small` |
| –27677 | `no_glyphs_added_to_font` |
| –27676 | `glyph_not_added_to_font` |
| –27675 | `point_does_not_intersect_bitmap` |
| –27674 | `required_font_table_not_present` |
| –27673 | `unknown_font_table_format` |
| –27672 | `shapeFill_not_allowed` |
| –27671 | `inverseFill_face_must_set_clipLayer_flag` |
| –27670 | `invalid_transferMode_colorSpace` |
| –27669 | `colorProfile_must_be_nil` |

*continued*

**Table 3-19** Bad parameter debugging errors (continued)

| Number | Name |
|---|---|
| –27668 | bitmap_pixel_size_must_be_1 |
| –27667 | empty_shape_not_allowed |
| –27666 | ignorePlatformShape_not_allowed |
| –27665 | nil_style_in_glyph_not_allowed |
| –27664 | complex_glyph_style_not_allowed |
| –27663 | invalid_mapping |
| –27662 | cannot_set_item_shapes_to_nil |
| –27661 | cannot_use_original_item_shapes_when_growing_picture |
| –27660 | cannot_add_unspecified_new_glyphs |
| –27659 | cannot_dispose_locked_tag |
| –27658 | cannot_dispose_locked_shape |

Table 3-20 lists the QuickDraw GX **restricted access** debugging errors.

**Table 3-20** Restricted access debugging errors

| Number | Name |
|---|---|
| –27657 | shape_access_not_allowed |
| –27656 | colorSet_access_restricted |
| –27655 | colorProfile_access_restricted |
| –27654 | tag_access_restricted |
| –27653 | viewDevice_access_restricted |
| –27652 | graphic_type_does_not_have_a_structure |
| –27651 | style_run_array_does_not_match_number_of_characters |
| –27650 | rectangles_cannot_be_inserted_into |
| –27649 | unknown_graphics_heap |
| –27648 | graphics_routine_selector_is_obsolete |
| –27647 | cannot_set_graphics_client_memory_without_setting_size |
| –27646 | graphics_client_memory_too_small |
| –27645 | graphics_client_memory_is_already_allocated |
| –27644 | viewPort_is_a_window |

Table 3-21 lists the QuickDraw GX **wrong type** and **bad reference** debugging errors.

**Table 3-21** Wrong type and bad reference debugging errors

| Number | Name |
|---|---|
| –27643 | illegal_type_for_shape |
| –27642 | shape_does_not_contain_a_bitmap |
| –27641 | shape_does_not_contain_text |
| –27640 | picture_expected |
| –27639 | bitmap_is_not_resizable |
| –27638 | shape_may_not_be_a_bitmap |
| –27637 | shape__may_not_be_a_picture |
| –27636 | graphic_type_does_not_contain_points |
| –27635 | graphic_type_does_not_have_multiple_contours |
| –27634 | graphic_type_cannot_be_mapped |
| –27633 | graphic_type_cannot_be_moved |
| –27632 | graphic_type_cannot_be_scaled |
| –27631 | graphic_type_cannot_be_rotated |
| –27630 | graphic_type_cannot_be_skewed |
| –27629 | graphic_type_cannot_be_reset |
| –27628 | graphic_type_cannot_be_dashed |
| –27627 | graphic_type_cannot_be_reduced |
| –27626 | graphic_type_cannot_be_inset |
| –27625 | shape_cannot_be_inverted |
| –27624 | shape_does_not_have_area |
| –27623 | shape_does not_have_length |
| –27622 | first_glyph_advance_must_be_absolute |
| –27621 | picture_cannot_contain_itself |
| –27620 | viewPort_cannot_contain_itself |
| –27619 | cannot_set_unique_items_attribute_when_picture_contains_items |
| –27618 | layer_style_cannot_contain_a_face |
| –27617 | layer_glyph_shape_cannot_contain_nil_styles |

QuickDraw GX posts validation errors only when QuickDraw GX validation error functions activate validation error checking. Validation error checking is discussed the chapter “QuickDraw GX Debugging” in this book. Table 3-22 lists the type validation debugging errors.

**Table 3-22** Type validation debugging errors

| Number | Name |
|---|---|
| –27616 | `object_wrong_type` |
| –27615 | `shape_wrong_type` |
| –27614 | `style_wrong_type` |
| –27613 | `ink_wrong_type` |
| –27612 | `transform_wrong_type` |
| –27611 | `device_wrong_type` |
| –27610 | `port_wrong_type` |

Table 3-23 lists the QuickDraw GX **cache validation** debugging errors.

**Table 3-23** Cache validation debugging errors

| Number | Name |
|---|---|
| –27609 | `shape_cache_wrong_type` |
| –27608 | `style_cache_wrong_type` |
| –27607 | `ink_cache_wrong_type` |
| –27606 | `transform_cache_wrong_type` |
| –27605 | `port_cache_wrong_type` |
| –27604 | `shape_cache_parent_mismatch` |
| –27603 | `style_cache_parent_mismatch` |
| –27602 | `ink_cache_parent_mismatch` |
| –27601 | `transform_cache_parent_mismatch` |
| –27600 | `port_cache_parent_mismatch` |
| –27599 | `invalid_shape_cache_port` |
| –27598 | `invalid_shape_cache_device` |
| –27597 | `invalid_ink_cache_port` |
| –27596 | `invalid_ink_cache_device` |
| –27595 | `invalid_style_cache_port` |

**Table 3-23** Cache validation debugging errors (continued)

| Number | Name |
|---|---|
| –27594 | `invalid_style_cache_device` |
| –27593 | `invalid_transform_cache_port` |
| –27592 | `invalid_transform_cache_device` |
| –27591 | `recursive_caches` |

Table 3-24 lists the QuickDraw GX **shape cache** validation debugging errors.

**Table 3-24** Shape cache validation shape debugging errors

| Number | Name |
|---|---|
| –27590 | `invalid_fillShape_ownerCount` |
| –27589 | `recursive_fillShapes` |

Table 3-25 lists the QuickDraw GX memory block validation debugging errors.

**Table 3-25** Memory block validation debugging errors

| Number | Name |
|---|---|
| –27588 | `indirect_memory_block_too_small` |
| –27587 | `indirect_memory_block_too_large` |
| –27586 | `unexpected_nil_pointer` |
| –27585 | `bad_address` |

Table 3-26 lists the QuickDraw GX object validation debugging errors.

**Table 3-26** Object validation debugging errors

| Number | Name |
|---|---|
| –27584 | no_owners |
| –27583 | invalid_pointer |
| –27582 | invalid_seed |
| –27581 | invalid_frame_seed |
| –27580 | invalid_text_seed |
| –27579 | invalid_draw_seed |
| –27578 | bad_printer_flags |

Table 3-27 lists the QuickDraw GX path and polygon validation debugging errors.

**Table 3-27** Path and polygon validation debugging errors

| Number | Name |
|---|---|
| –27577 | invalid_vector_count |
| –27576 | invalid_contour_count |

Table 3-28 lists the QuickDraw GX bitmap validation debugging errors.

**Table 3-28** Bitmap validation debugging errors

| Number | Name |
|---|---|
| –27575 | bitmap_ptr_too_small |
| –27574 | bitmap_ptr_not_aligned |
| –27573 | bitmap_rowBytes_negative |
| –27572 | bitmap_width_negative |
| –27571 | bitmap_height_negative |
| –27570 | invalid_pixelSize |
| –27569 | bitmap_rowBytes_too_small |
| –27568 | bitmap_rowBytes_not_aligned |
| –27567 | bitmap_rowBytes_must_be_specified_for_user_image_buffer |

Table 3-29 lists the QuickDraw GX bitmap image validation debugging errors.

**Table 3-29** Bitmap image validation debugging errors

| Number | Name |
|---|---|
| –27566 | `invalid_bitImage_fileOffset` |
| –27565 | `invalid_bitImage_owners` |
| –27564 | `invalid_bitImage_rowBytes` |
| –27563 | `invalid_bitImage_internal_flag` |

Table 3-30 lists the QuickDraw GX text validation debugging errors.

**Table 3-30** Text validation debugging errors

| Number | Name |
|---|---|
| –27562 | `text_bounds_cache_wrong_size` |
| –27561 | `text_metrics_cache_wrong_size` |
| –27560 | `text_index_cache_wrong_size` |

Table 3-31 lists the QuickDraw GX glyph validation debugging errors.

**Table 3-31** Glyph validation debugging errors

| Number | Name |
|---|---|
| –27559 | `glyph_run_count_negative` |
| –27558 | `glyph_run_count_zero` |
| –27557 | `glyph_run_counts_do_not_sum_to_character_count` |
| –27556 | `glyph_first_advance_bit_set_not_allowed` |
| –27555 | `glyph_tangent_vectors_both_zero` |

Table 3-32 lists the QuickDraw GX layout validation debugging errors.

**Table 3-32** Layout validation debugging errors

| Number | Name |
|---|---|
| –27554 | `layout_run_length_negative` |
| –27553 | `layout_run_length_zero` |
| –27552 | `layout_run_level_negative` |
| –27551 | `layout_run_lengths_do_not_sum_to_text_length` |

Table 3-33 lists the QuickDraw GX picture validation debugging errors.

**Table 3-33** Picture validation debugging errors

| Number | Name |
|---|---|
| –27550 | `bad_shape_in_picture` |
| –27549 | `bad_style_in_picture` |
| –27548 | `bad_ink_in_picture` |
| –27547 | `bad_transform_in_picture` |
| –27546 | `bad_shape_cache_in_picture` |
| –27545 | `bad_seed_in_picture` |
| –27544 | `invalid_picture_count` |

Table 3-34 lists the QuickDraw GX text face validation debugging errors.

**Table 3-34** Text face validation debugging errors

| Number | Name |
|---|---|
| –27543 | `bad_textLayer_count` |
| –27542 | `bad_fillType_in_textFace` |
| –27541 | `bad_style_in_textFace` |
| –27540 | `bad_transform_in_textFace` |

Table 3-35 lists the QuickDraw GX transform validation debugging errors.

**Table 3-35** Transform validation debugging errors

| Number | Name |
|---|---|
| –27539 | `invalid_matrix_flag` |
| –27538 | `transform_clip_missing` |

Table 3-36 lists the QuickDraw GX font cache validation debugging errors.

**Table 3-36** Font cache validation debugging errors

| Number | Name |
|---|---|
| –27537 | `metrics_wrong_type` |
| –27536 | `metrics_point_size_probably_bad` |
| –27535 | `scalar_block_wrong_type` |
| –27534 | `scalar_block_parent_mismatch` |
| –27533 | `scalar_block_too_small` |
| –27532 | `scalar_block_too_large` |
| –27531 | `invalid_metrics_range` |
| –27530 | `invalid_metrics_flags` |
| –27529 | `metrics_maxWidth_probably_bad` |
| –27528 | `font_wrong_type` |
| –27527 | `font_wrong_size` |
| –27526 | `invalid_font_platform` |
| –27525 | `invalid_lookup_range` |
| –27524 | `invalid_lookup_platform` |
| –27523 | `font_not_in_font_list` |
| –27522 | `metrics_not_in_metrics_list` |

Table 3-37 lists the QuickDraw GX view device validation debugging errors.

**Table 3-37** View device validation debugging errors

| Number | Name |
|---|---|
| –27521 | `bad_device_private_flags` |
| –27520 | `bad_device_attributes` |
| –27519 | `invalid_device_number` |
| –27518 | `invalid_device_viewGroup` |
| –27517 | `invalid_device_bounds` |
| –27516 | `invalid_bitmap_in_device` |

Table 3-38 lists the QuickDraw GX color set validation debugging errors.

**Table 3-38** Color set validation debugging errors

| Number | Name |
|---|---|
| –27515 | `colorSet_wrong_type` |
| –27514 | `invalid_colorSet_viewDevice_owners` |
| –27513 | `invalid_colorSet_colorSpace` |
| –27512 | `invalid_colorSet_count` |

Table 3-39 lists the QuickDraw GX color profile validation debugging errors.

**Table 3-39** Color profile validation debugging errors

| Number | Name |
|---|---|
| –27511 | `colorProfile_wrong_type` |
| –27510 | `invalid_colorProfile_flags` |
| –27509 | `invalid_colorProfile_response_count` |

Table 3-40 lists the QuickDraw GX internal backing store validation debugging errors.

**Table 3-40** Internal backing store validation debugging errors

| Number | Name |
|---|---|
| –27508 | `backing_free_parent_mismatch` |
| –27507 | `backing_store_parent_mismatch` |

## Warnings

This section describes the warnings that the debugging version of QuickDraw GX may post. QuickDraw GX debugging warnings can be grouped into the following categories:

- invalid data warnings
- can’t find warnings
- other warnings

Table 3-41 gives the range of debugging warning numbers.

**Table 3-41** Debugging warning number range

| Number | Description |
|---|---|
| –26700 | `gxFirstSystemDebuggingWarning` |
| –26000 | `gxLastSystemWarning` |

Table 3-42 lists the QuickDraw GX **invalid data** debugging warnings.

**Table 3-42** Invalid data debugging warnings

| Number | Name |
|---|---|
| –26700 | new_shape_contains_invalid_data |
| –26699 | new_tag_contains_invalid_data |
| –26698 | extra_data_passed_was_ignored |
| –26697 | font_table_not_found |
| –26696 | font_name_not_found |
| –26695 | unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve |
| –26694 | unable_to_draw_open_contour_that_starts_or_ends_off_the_curve |
| –26693 | cannot_dispose_default_shape |
| –26692 | cannot_dispose_default_style |
| –26691 | cannot_dispose_default_ink |
| –26690 | cannot_dispose_default_transform |
| –26689 | cannot_dispose_default_colorProfile |
| –26688 | cannot_dispose_default_colorSet |
| –26687 | shape_direct_attribute_not_set |

Table 3-43 lists the QuickDraw GX can't find debugging warnings.

**Table 3-43** Can't find debugging warnings

| Number | Name |
|---|---|
| –26686 | point_does_not_intersect_port |
| –26685 | cannot_dispose_non_font |
| –26684 | face_override_style_font_must_match_style |
| –26683 | union_of_area_and_and_length_returns_area_only |
| –26682 | insufficient_coordinate_space_for_new_device |

Table 3-44 lists the QuickDraw GX other debugging warnings.

**Table 3-44** Other debugging warnings

| Number | Name |
|---|---|
| –26681 | `shape_passed_has_no_bounds` |
| –26680 | `tags_of_type_flst_removed` |
| –26679 | `translator_not_installed_on_this_grafport` |

## Notices

QuickDraw GX provides notices only in the debugging version. This section describes the notices that the debugging version of QuickDraw GX may post. Each QuickDraw notice has a unique **notice number** and a **notice name.** Table 3-45 gives the debugging notice number range.

**Table 3-45** Debugging version notice number summary

| Number | Description |
|---|---|
| –25999 | `gxFirstSystemNotice` |
| –25500 | `gxLastSystemNotice` |

Table 3-46 lists the QuickDraw GX debugging notices.

**Table 3-46** Debugging notices

| Number | Name |
|---|---|
| –25999 | `parameters_have_no_effect` |
| –25998 | `attributes_already_set` |
| –25997 | `caps_already_set` |
| –25996 | `clip_already_set` |
| –25995 | `color_already_set` |
| –25994 | `curve_error_already_set` |
| –25993 | `dash_already_set` |
| –25992 | `default_colorProfile_already_set` |
| –25991 | `default_ink_already_set` |
| –25990 | `default_transform_already_set` |

*continued*

**Table 3-46** Debugging notices (continued)

| Number | Name |
|---|---|
| –25989 | default_shape_already_set |
| –25988 | default_style_already_set |
| –25987 | dither_already_set |
| –25986 | encoding_already_set |
| –25985 | face_already_set |
| –25984 | fill_already_set |
| –25983 | font_already_set |
| –25982 | font_variations_already_set |
| –25981 | glyph_positions_are_already_set |
| –25980 | glyph_tangents_are_already_set |
| –25979 | halftone_already_set |
| –25978 | hit_test_already_set |
| –25977 | ink_already_set |
| –25976 | join_already_set |
| –25975 | justification_already_set |
| –25974 | mapping_already_set |
| –25973 | pattern_already_set |
| –25972 | pen_already_set |
| –25971 | style_already_set |
| –25970 | tag_already_set |
| –25969 | text_attributes_already_set |
| –25968 | text_size_already_set |
| –25967 | transfer_already_set |
| –25966 | translator_already_installed_on_this_grafport |
| –25965 | transform_already_set |
| –25964 | type_already_set |
| –25963 | validation_level_already_set |
| –25962 | viewPorts_already_set |
| –25961 | viewPorts_already_in_viewGroup |
| –25960 | viewDevice_already_in_viewGroup |
| –25959 | geometry_unaffected |
| –25958 | mapping_unaffected |

**Table 3-46** Debugging notices (continued)

| Number | Name |
|---|---|
| –25957 | tags_in_shape_ignored |
| –25956 | shape_already_in_primitive_form |
| –25955 | shape_already_in_simple_form |
| –25954 | shape_already_broken |
| –25953 | shape_already_joined |
| –25952 | cache_already_cleared |
| –25951 | shape_not_disposed |
| –25950 | style_not_disposed |
| –25949 | ink_not_disposed |
| –25948 | transform_not_disposed |
| –25947 | colorSet_not_disposed |
| –25946 | colorProfile_not_disposed |
| –25945 | font_not_disposed |
| –25944 | glyph_tangents_have_no_effect |
| –25943 | glyph_positions_determined_by_advance |
| –25942 | transform_viewPorts_already_set |
| –25941 | directShape_attribute_set_as_side_effect |
| –25940 | lockShape_called_as_side_effect |
| –25939 | lockTag_called_as_side_effect |
| –25938 | shapes_unlocked_as_side_effect |
| –25937 | shape_not_locked |
| –25936 | tag_not_locked |
| –25935 | disposed_dead_caches |
| –25934 | disposed_live_caches |
| –25933 | low_on_memory |
| –25932 | very_low_on_memory |
| –25931 | transform_references_disposed_viewPort |

# Using Errors, Warnings, and Notices

This section describes how to control and utilize QuickDraw GX errors, warnings, and notices and how to include an application-defined function to provide complementary or alternative error, warning, and notice processing. This section describes how you can

- obtain the QuickDraw GX errors, warnings, and notices posted
- change the QuickDraw GX errors, warnings, and notices posted
- ignore QuickDraw GX warnings and notices
- install application-defined error, warning, and notice handlers

## Obtaining Errors, Warnings, and Notices

You can use the `GXGetGraphicsError`, `GXGetGraphicsWarning`, and `GXGetGraphicsNotice` functions to obtain QuickDraw GX error, warning, and notice messages describing problems that occur during the execution of your application. These three functions return the last problem encountered during execution. If no problem has been posted, the function returns 0 until a problem message is posted.

The `stickyError`, `stickyWarning`, or `stickyNotice` parameters of the respective function, if not `nil`, are pointers to the first execution problem that QuickDraw GX encountered after the last time that the `GXGetGraphicsError`, `GXGetGraphicsWarning`, or `GXGetGraphicsNotice` function was called. These functions thereby allow you to determine both the original problem and the final problem that was detected by QuickDraw GX during execution of your application.

**Note**
Notices are posted only in the debugging version of QuickDraw GX. ◆

Figure 3-2 shows the use of these polling functions to obtain the errors, warnings, and notices of selected blocks of your code.

**Figure 3-2** Polling for errors, warnings, and notices

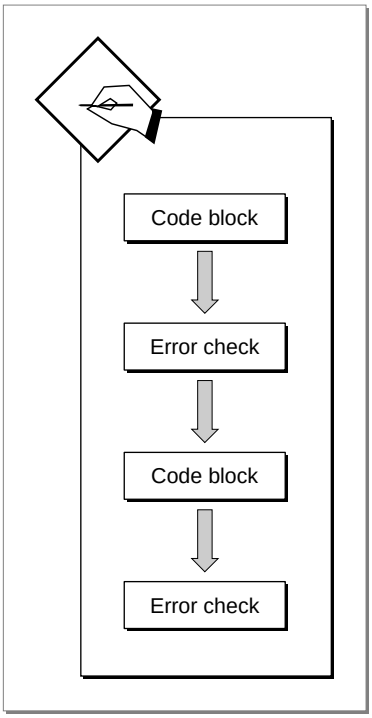
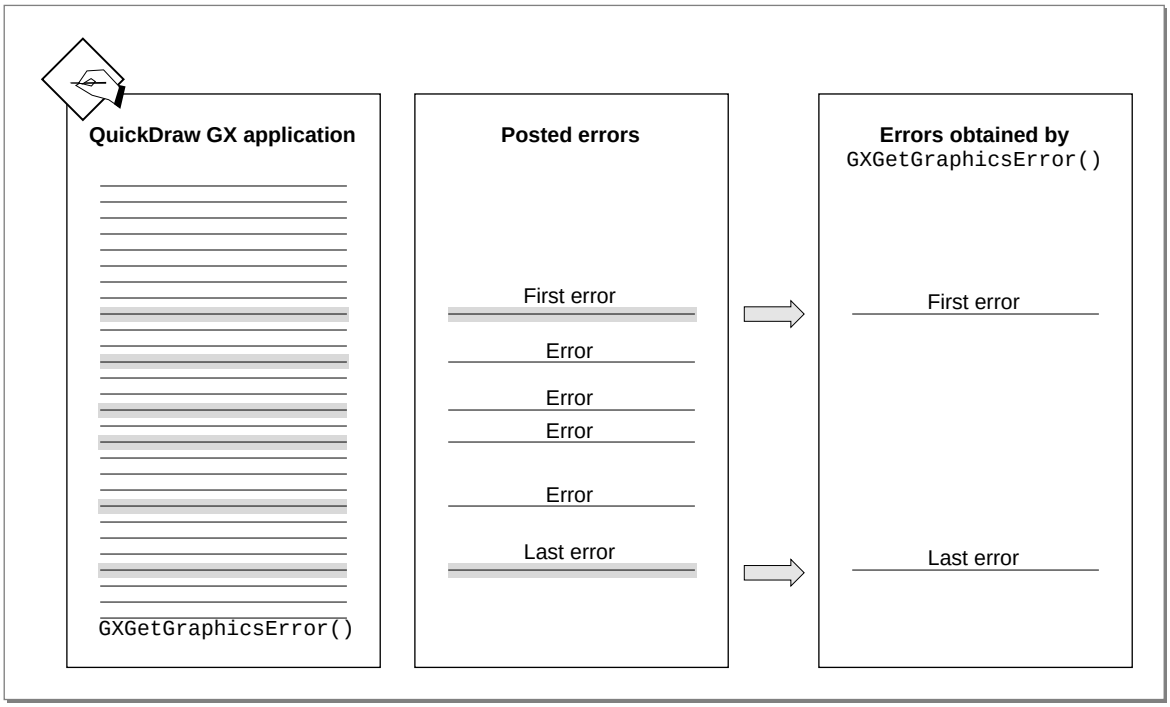


Figure 3-3 shows the use of the `GXGetGraphicsError` function to obtain the first and last errors posted when you test your QuickDraw GX application.

**Figure 3-3** Obtaining the first and last posted QuickDraw GX error



Listing 3-1 shows the use of the `GXGetGraphicsError` function to obtain the first error posted after the execution of a block of code.

**Listing 3-1** Obtaining the first posted error

```
static void ObtainOriginalError(void)
{

/* block of application code */

/*
If an error occurred, then see if the orginal error was
out_of_memory. Note that you need to look at the original error,
not the last error returned, since if the NewLine fails, then the
next two functions (DrawShape and DisposeShape) will generate a
shape_is_nil error.
*/

   {  graphicsError myError, originalError;
      if( myError = GetGraphicsError(&originalError) ) {
         if( originalError == out_of_memory ) {
            /* post out of memory dialog box */
         } else {
/* post generic error dialog box */
         }
      }
   }
}
```

Listing 3-2 shows the use of the `GXGetGraphicsWarning` function to obtain the first and last warning posted after the execution of a block of code.

**Listing 3-2** Obtaining the first and last QuickDraw GX warning

```
static void ObtainFirstLastWarning(void)
{

/* block of application code */

/*
It might be valuable to look at both myWarning (last warning
posted) and originalWarning (first warning posted), although the
last warning is usually the most important warning posted.
*/

   {  graphicsWarning myWarning, originalWarning;
      if( myWarning = GXGetGraphicsWarning(&originalWarning) ) {
         DebugStr("\pa warning occurred");
      }
   }
}
```

Listing 3-3 shows the use of the `GXGetGraphicsNotice` function to obtain the first and last notices posted after the execution of a block of code.

**Listing 3-3** Obtaining the first and last posted notices

```
static void ObtainFirstLastNotice(void)
{

/* block of application code */

/*
It might be useful to look at both myNotice (last notice
posted)and originalNotice (first notice posted), although the
last notice is usually the most important notice posted.
*/
```

```
   {  graphicsNotice myNotice, originalNotice;
      if( myNotice = GXGetGraphicsNotice(&originalNotice) ) {
         DebugStr("\pa notice occurred");
      }
   }
}
```

The `GXGetGraphicsError` function is described on page 3-56. The QuickDraw GX errors that may be posted are listed in the section "Errors" beginning on page 3-6. QuickDraw GX allows you to ignore warnings and notices, but does not provide a function that will ignore errors.

The `GXGetGraphicsWarning` function is described on page 3-60. The QuickDraw GX warnings that may be posted are listed in the section "Warnings" beginning on page 3-10. QuickDraw GX allows you to ignore warnings that would otherwise be posted. How to ignore warnings is discussed in the section "Ignoring Warnings and Notices" beginning on page 3-37. The `GXIgnoreGraphicsWarning` function is discussed on page 3-64.

The `GXGetGraphicsNotice` function is described on page 3-66. The QuickDraw GX notices that may be posted are listed in the section "Notices" beginning on page 3-27. QuickDraw GX allows you to ignore notices that would otherwise be posted. How to ignore notices is discussed in the section "Ignoring Warnings and Notices" beginning on page 3-37. The `GXIgnoreGraphicsNotice` function is discussed on page 3-70.

**Note**
An alternative or complementary approach to the use of the `GXGetGraphicsError`, `GXGetGraphicsWarning`, and `GXGetGraphicsNotice` functions is to include an application-defined error, warning, or notice handler. This topic is discussed in the section "Installing an Error, Warning, or Notice Handler" beginning on page 3-40. ◆

## Changing the Error, Warning, or Notice Posted

You can use the `GXPostGraphicsError`, `GXPostGraphicsWarning`, and `GXPostGraphicsNotice` functions to post your own errors, warnings, and notices from inside your application.

**Note**
Notices are posted only in the debugging version of QuickDraw GX. ◆

The `GXPostGraphicsError` function replaces the current QuickDraw GX error with any error message you provide as the `error` parameter. The error you substitute may be one of the QuickDraw GX errors or your own error message. This function stores the new error message so that subsequent calls to `GXGetGraphicsError` return the error substituted by this function.

Listing 3-4 shows the use of the `GXPostGraphicsError` function to change the posted error to an error having the name `special_user_error` and the error number 2097152. This is the `gxFirstAppError` constant.

**Listing 3-4** Changing the error posted

```
static long SampleCode4(void)
{
   #define special_user_error 2097152L
   #define end_of_file -1L
   long myFilePosition = 0;

/* block of application code */

   if( myFilePosition == end_of_file ) {

/* indicate that an error occurred */

      PostGraphicsError(special_user_error);
   }

   /* block of application code */

/*
You need to check for errors only once; this will catch errors
generated by QuickDraw GX and any user-defined errors that were
posted.
*/


   {  graphicsError myError;
      if( myError = GXGetGraphicsError(nil) )
         return myError;
   }

   /* block of application code */

}
```

The `GXPostGraphicsError` function is described on page 3-57. The QuickDraw GX errors are listed in the section “Errors” beginning on page 3-6.

The `GXPostGraphicsWarning` function replaces the current QuickDraw GX warning with any warning message you provide as the `warning` parameter. The warning you substitute may be one of the QuickDraw GX warnings or your own warning message. This function stores the new warning message so that subsequent calls to `GXGetGraphicsWarning` return the warning substituted by this function.

The `GXPostGraphicsWarning` function is described on page 3-61. The QuickDraw GX warnings are listed in the section "Warnings" beginning on page 3-10.

The `GXPostGraphicsNotice` function replaces the current QuickDraw GX notice with any notice message you provide as the `notice` parameter. The notice you substitute may be one of the QuickDraw GX notices or your own notice message. This function stores the new notice message so that subsequent calls to `GXGetGraphicsNotice` return the notice substituted by this function.

The `GXPostGraphicsNotice` function is described on page 3-67. The QuickDraw GX notices are listed in the section "Notices" beginning on page 3-27.



## Ignoring Warnings and Notices

You can use the `GXIgnoreGraphicsWarning` and `GXIgnoreGraphicsNotice` functions to selectively ignore, and thereby suppress, the posting of specific QuickDraw GX warnings and notices in parts of your application. There is no analogous function to ignore errors.

**Note**

Notices are posted only in the debugging version of QuickDraw GX. ◆

The `GXIgnoreGraphicsWarning` function places the warning to be ignored on the **ignore warning stack.** The posting of all QuickDraw GX warnings that are on the ignore warning stack is suppressed, just as if the problem that resulted in the warning message never occurred.

When a QuickDraw GX warning is about to be posted, QuickDraw GX determines if the specific warning is on the ignore warning stack. If the warning to be posted is on the stack, QuickDraw GX does not post this warning. If the warning to be posted is not on the ignore warning stack, QuickDraw GX does post the warning. QuickDraw GX does not change the stack when it checks for the presence or absence of a warning.

The `GXPopGraphicsWarning` function removes warnings from the ignore warning stack in the reverse order that they are placed on the stack by the `GXIgnoreGraphicsWarning` function. You don't need to specify which warning to remove. You remove one ignored warning code from the top of the ignore warning stack each time that you call the `GXPopGraphicsWarning` function.

**Note**
There is an implementation limit on the number of times that you can use the `GXIgnoreGraphicsWarning` and `GXPopGraphicsWarning` functions. When the implementation limit is exceeded, QuickDraw GX posts a `warning_stack_overflow` warning message. If there are no warnings on the ignore warning stack and the `GXPopGraphicsWarning` function is called, QuickDraw GX posts a `warning_stack_underflow` warning message. ◆

Since there is an implementation limit on the number of warnings and notices that you can ignore, you should use the `GXIgnoreGraphicsWarning` and `GXPopGraphicsWarning` functions only when you need to debug specific parts of your application code.

The `GXIgnoreGraphicsNotice` function provides the same feature for notices that the `GXIgnoreGraphicsWarning` function provides for warnings.

The `GXIgnoreGraphicsNotice` function places the notice to be ignored on the **ignore notice stack.** The posting of all QuickDraw GX notices on the ignore notice stack is suppressed, just as if the problem that resulted in the notice message never occurred.

When a QuickDraw GX notice is about to be posted, QuickDraw GX determines if the specific notice is on the ignore notice stack. If the notice to be posted is on the stack, QuickDraw GX does not post this notice. If the notice to be posted is not on the ignore notice stack, QuickDraw GX does post it. QuickDraw GX does not change the stack when it checks for the presence or absence of a notice.

The `GXPopGraphicsNotice` function removes notices from the ignore notice stack in the reverse order that they are placed on the stack by the `GXIgnoreGraphicsNotice` function. You don't need to specify which notice to remove. You remove one ignored notice code from the top of the ignore notice stack each time you call the `GXPopGraphicsNotice` function.

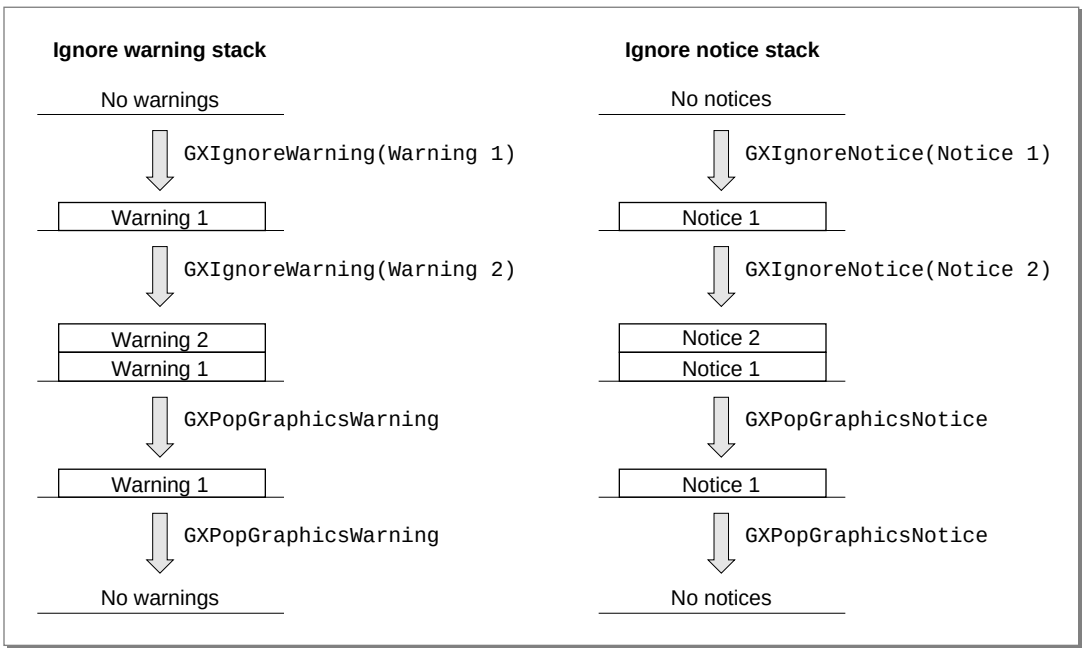
**Note**
There is an implementation limit on the number of times that you can use the `GXIgnoreGraphicsNotice` and `GXPopGraphicsNotice` functions. When the implementation limit is exceeded, QuickDraw GX will post a `notice_stack_overflow` warning message. If there are no notices on the notice warning stack and the `GXPopGraphicsNotice` function is called, QuickDraw GX posts a `notice_stack_underflow` warning message. ◆

For example, if you wanted to suppress the `attributes_already_set` notice posted by QuickDraw GX, you could use the `GXIgnoreGraphicsNotice` function to push its notice number, –25998, onto the ignore notice stack. When QuickDraw GX is about to post a notice, it looks on the ignore notice stack to determine if its notice number is on the ignore notice stack. If the notice to be posted is `attributes_already_set`, then the notice is not posted. QuickDraw GX posts any notice that is not on the ignore notice stack.

If you also wanted to ignore the `color_already_set` notice, then you could use the `GXIgnoreGraphicsNotice` function to push its notice number, –25995, onto the ignore notice stack. QuickDraw GX would then ignore, and therefore not post, the `attributes_already_set` and `color_already_set` notices. Since you added the notices to the ignore notice stack in the order `attributes_already_set` and then `color_already_set`, the `color_already_set` notice would be on top of the ignore notice stack. When you use the `GXPopGraphicsNotice` function to remove a notice from the stack, the first notice to be removed is `color_already_set`, the one on top of the ignore notice stack. To remove the `attributes_already_set` notice, you need to call the `GXPopGraphicsNotice` function a second time. After the second call to the `GXPopGraphicsNotice` function, no notices are on the ignore notice stack. As a result, QuickDraw GX resumes posting all notices.

Figure 3-4 illustrates how warnings and notices are added to and removed from the ignore warning stack and the ignore notice stack.

**Figure 3-4** Adding and removing warnings and notices from the ignore warning and ignore notice stacks




You should ignore warnings and notices only if you are confident that you understand why they are being issued and the consequences of ignoring these warnings and notices.

For example, if your program asks for 100 points in a polygon and there are fewer points available, QuickDraw GX posts a warning and returns all of the points that are available. You can add the `GXIgnoreGraphicsNotice` function to your code to suppress this warning, but your application needs to be smart enough to accommodate the fact that less than the requested number of points may be returned.

The GXIgnoreGraphicsWarning function is discussed on page 3-64. The GXPopGraphicsWarning function is discussed on page 3-65. The QuickDraw warning names and numbers that may be ignored are listed in the section “Warnings” beginning on page 3-10.

The GXIgnoreGraphicsNotice function is discussed on page 3-70. The GXPopGraphicsNotice function is discussed on page 3-71. The QuickDraw GX warning names and numbers that can be ignored are listed in the section “Notices” beginning on page 3-27.

## Installing an Error, Warning, or Notice Handler

You can use the GXSetUserGraphicsError, GXSetUserGraphicsWarning, and GXSetUserGraphicsNotice functions to install an application-defined function that you want to call whenever an error, warning, or notice occurs. QuickDraw GX will pass this function the error, warning, or notice when it is generated. Your function can then respond accordingly. You may use the reference argument to pass an associated long value parameter to your function. If you want to disable your handler, you just pass nil.

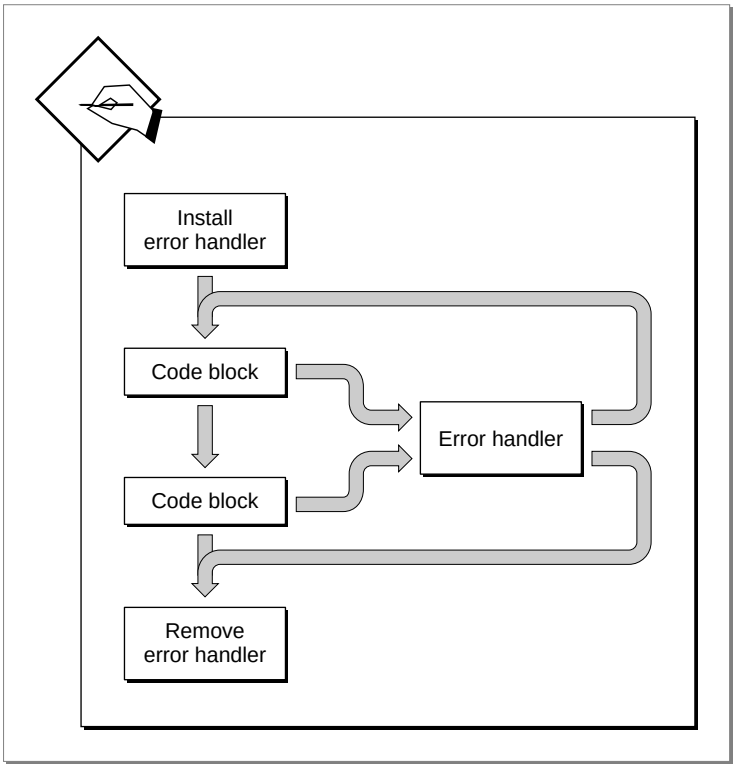
Your application can then take advantage of these errors. For example, QuickDraw GX may post an error indicating that your application has run out of memory or has tried to use a font that is not installed. As a result, your application may be able to recommend corrective action via the application-defined error handling function and the application’s human interface.

You can use the GXGetUserGraphicsError, GXGetUserGraphicsWarning, and GXGetUserGraphicsNotice functions to obtain the application-defined handler functions that have been previously installed by GXSetUserGraphicsError, GXSetUserGraphicsWarning, and SetUserGraphicsNotices. These functions return nil if no function has been installed.

You usually install handlers at the beginning of your application code. You can install error, warning, and notice handlers before any graphics operations have occurred and before the GXEnterGraphics function has been called. If you do, QuickDraw GX will call the GXEnterGraphics function for you. In contrast, you can’t install error, warning, and notice handlers before calling the GXNewGraphicsClient function.

Alternatively, you may selectively enable and disable error, warning, and notice handlers at different sections of the application code. Figure 3-5 shows how an error handler can be enabled and disabled within the application. This is an effective method for ignoring errors, warnings, and notices, analogous to the use of the GXIgnoreGraphicsError, GXIgnoreGraphicsWarning, and GXIgnoreGraphicsNotice functions.

**Figure 3-5** Enabling and disabling an error handler




The handler should respond to the problems that occur during typical application scenarios. A friendly application should let the user know when it is taking action in response to errors and warnings that have occurred. For example, if an application runs out of memory, it may let the user know that it is out of memory and that it is responding in a particular manner to alleviate the problem. If it cannot solve the problem, it may need to notify the user that it needs to abort processing. Such an application would need to install an error handler that looks for `out_of_memory` errors.

In general, in the non-debugging version of your application, the handler might be relatively simple. If the handler doesn't have a response to an error or warning, it should just return and continue execution.

In contrast, the debugging version of the handler may be relatively complex to accommodate special error, warning, and notice conditions. In general, you should stop and print the errors, warnings, and notices whenever a problem occurs.

An application can have more than one error handler. A simple application might have just one error handler to handle specific problems. However, a more complicated application may have multiple error handlers. For example, an application might have one error handler that takes care of memory problems and another error handler for other types of errors. The special error handler may be installed only when a particular type of processing is to occur, like animation or QuickTime movies.

# Errors, Warnings, and Notices Reference

This section provides reference information related to the data types and functions that allow you to control the generation of errors, warnings, and notices.

## Constants and Data Types

This section describes the error, warning, and notice data types that you may use in your application.

### Errors

QuickDraw GX provides you with an extended set of errors in the debugging version and a reduced set of errors in the non-debugging version. Each QuickDraw GX error constant has an error number described by the `gxGraphicsError` type definition and the `gxGraphicErrors` enumeration:

```
typedef long gxGraphicsError;

enum gxGraphicErrors {

   /* truly fatal errors */
   out_of_memory = -27999,
   internal_fatal_error,
   no_outline_font_found,
   not_enough_memory_for_graphics_client_heap,
   could_not_create_backing_store,

   /* internal errors */
   internal_error = -27950,
   internal_font_error,
   internal_layout_error,

   /* recoverable errors */
   could_not_dispose_backing_store = internal_layout_error + 2,
   unflattening_interrupted_by_client,

   /* font manager errors */
   font_cannot_be_changed,
   illegal_font_parameter,
```

```
/* gxFont scaler errors */
null_font_scaler_context = -27900,
null_font_scaler_input,
invalid_font_scaler_context,
invalid_font_scaler_input,
invalid_font_scaler_font_data,
font_scaler_newblock_failed,
font_scaler_getfonttable_failed,
font_scaler_bitmap_allocation_failed,
font_scaler_outline_allocation_failed,
required_font_scaler_table_missing,
unsupported_font_scaler_outline_format,
unsupported_font_scaler_stream_format,
unsupported_font_scaler_font_format,
font_scaler_hinting_error,
font_scaler_rasterizer_error,
font_scaler_internal_error,
font_scaler_invalid_matrix,
font_scaler_fixed_overflow,
font_scaler_api_version_mismatch,
font_scaler_streaming_aborted,
unknown_font_scaler_error,

/* bad parameters */
parameter_is_nil = -27850,
shape_is_nil,
style_is_nil,
transform_is_nil,
ink_is_nil,
transferMode_is_nil,
color_is_nil,
colorProfile_is_nil,
colorSet_is_nil,
spoolProcedure_is_nil,
tag_is_nil,
type_is_nil,
mapping_is_nil,
invalid_viewDevice_reference,
invalid_viewGroup_reference,
invalid_viewPort_reference,

/* implementation limits */
number_of_contours_exceeds_implementation_limit = -27800,
```

```
    number_of_points_exceeds_implementation_limit,
    size_of_polygon_exceeds_implementation_limit,
    size_of_path_exceeds_implementation_limit,
    size_of_text_exceeds_implementation_limit,
    size_of_bitmap_exceeds_implementation_limit,
    number_of_colors_exceeds_implementation_limit,
    procedure_not_reentrant

    #ifdef debugging
    ,
    /* internal debugging errors: following available only in */
    /* the debugging init */
    functionality_unimplemented = -27700,
    clip_to_frame_shape_unimplemented,

    /* font parameter debugging errors */
    illegal_font_storage_type,
    illegal_font_storage_reference,
    illegal_font_attributes,

    /* parameter debugging errors */
    parameter_out_of_range,
    inconsistent_parameters,
    index_is_less_than_zero,
    index_is_less_than_one,
    count_is_less_than_zero,
    count_is_less_than_one,
    contour_is_less_than_zero,
    length_is_less_than_zero,

    invalid_client_reference,
    invalid_graphics_heap_start_pointer,
    invalid_nongraphic_globals_pointer,

    colorSpace_out_of_range,

    pattern_lattice_out_of_range,
    frequency_parameter_out_of_range,
    tinting_parameter_out_of_range,
    method_parameter_out_of_range,
    space_may_not_be_indexed,
```

```
    glyph_index_too_small,
    no_glyphs_added_to_font,
    glyph_not_added_to_font,
    point_does_not_intersect_bitmap,

    required_font_table_not_present,
    unknown_font_table_format,

/* the styles encoding is not present in the font */
    shapeFill_not_allowed,
    inverseFill_face_must_set_clipLayer_flag,
    invalid_transferMode_colorSpace,
    colorProfile_must_be_nil,
    bitmap_pixel_size_must_be_1,
    empty_shape_not_allowed,
    ignorePlatformShape_not_allowed,
    nil_style_in_glyph_not_allowed,
    complex_glyph_style_not_allowed,
    invalid_mapping,

    cannot_set_item_shapes_to_nil,
    cannot_use_original_item_shapes_when_growing_picture,
    cannot_add_unspecified_new_glyphs,
    cannot_dispose_locked_tag,
    cannot_dispose_locked_shape,

    /* restricted access */
    shape_access_not_allowed,
    colorSet_access_restricted,
    colorProfile_access_restricted,
    tag_access_restricted,
    viewDevice_access_restricted,

    graphic_type_does_not_have_a_structure,
    style_run_array_does_not_match_number_of_characters,
    rectangles_cannot_be_inserted_into,

    unknown_graphics_heap,
    graphics_routine_selector_is_obsolete,
    cannot_set_graphics_client_memory_without_setting_size,
    graphics_client_memory_too_small,
    graphics_client_memory_is_already_allocated,
```

```
viewPort_is_a_window,

/* wrong type/bad reference */
illegal_type_for_shape,
shape_does_not_contain_a_bitmap,
shape_does_not_contain_text,
picture_expected,
bitmap_is_not_resizable,
shape_may_not_be_a_bitmap,
shape_may_not_be_a_picture,
graphic_type_does_not_contain_points,
graphic_type_does_not_have_multiple_contours,
graphic_type_cannot_be_mapped,
graphic_type_cannot_be_moved,
graphic_type_cannot_be_scaled,
graphic_type_cannot_be_rotated,
graphic_type_cannot_be_skewed,
graphic_type_cannot_be_reset,
graphic_type_cannot_be_dashed,
graphic_type_cannot_be_reduced,
graphic_type_cannot_be_inset,
shape_cannot_be_inverted,
shape_does_not_have_area,
shape_does_not_have_length,
first_glyph_advance_must_be_absolute,
picture_cannot_contain_itself,
viewPort_cannot_contain_itself,

cannot_set_unique_items_attribute_when_picture_
contains_items,
layer_style_cannot_contain_a_face,
layer_glyph_shape_cannot_contain_nil_styles,

   /* validation errors */
object_wrong_type,
shape_wrong_type,
style_wrong_type,
ink_wrong_type,
transform_wrong_type,
device_wrong_type,
port_wrong_type,
```

```
/*cache validation errors */
shape_cache_wrong_type,
style_cache_wrong_type,
ink_cache_wrong_type,
transform_cache_wrong_type,
port_cache_wrong_type,
shape_cache_parent_mismatch,
style_cache_parent_mismatch,
ink_cache_parent_mismatch,
transform_cache_parent_mismatch,
port_cache_parent_mismatch,
invalid_shape_cache_port,
invalid_shape_cache_device,
invalid_ink_cache_port,
invalid_ink_cache_device,
invalid_style_cache_port,
invalid_style_cache_device,
invalid_transform_cache_port,
invalid_transform_cache_device,
recursive_caches,

/*shape cache validation errors */
invalid_fillShape_ownerCount,
recursive_fillShapes,

/*memory block validation errors */
indirect_memory_block_too_small,
indirect_memory_block_too_large,
unexpected_nil_pointer,
bad_address,

/* object validation errors */
no_owners,
invalid_pointer,
invalid_seed,
invalid_frame_seed,
invalid_text_seed,
invalid_draw_seed,
bad_private_flags,

/* path and polygon validation errors */
invalid_vector_count,
invalid_contour_count,
```

```
/* validation bitmap errors */
bitmap_ptr_too_small,
bitmap_ptr_not_aligned,
bitmap_rowBytes_negative,
bitmap_width_negative,
bitmap_height_negative,
invalid_pixelSize,
bitmap_rowBytes_too_small,
bitmap_rowBytes_not_aligned,
bitmap_rowBytes_must_be_specified_for_user_image_buffer,

/* bitmap validation image errors */
invalid_bitImage_fileOffset,
invalid_bitImage_owners,
invalid_bitImage_rowBytes,
invalid_bitImage_internal_flag,

/* text validation errors */
text_bounds_cache_wrong_size,
text_metrics_cache_wrong_size,
text_index_cache_wrong_size,

/* glyph validation errors */
glyph_run_count_negative,
glyph_run_count_zero,
glyph_run_counts_do_not_sum_to_character_count,
glyph_first_advance_bit_set_not_allowed,
glyph_tangent_vectors_both_zero,

/* layout validation errors */
layout_run_length_negative,
layout_run_length_zero,
layout_run_level_negative,
layout_run_lengths_do_not_sum_to_text_length,

/* picture validation errors */
bad_shape_in_picture,
bad_style_in_picture,
bad_ink_in_picture,
bad_transform_in_picture,
bad_shape_cache_in_picture,
bad_seed_in_picture,
invalid_picture_count,
```

```
/* text face validation errors */
bad_textLayer_count,
bad_fillType_in_textFace,
bad_style_in_textFace,
bad_transform_in_textFace,

/* transform validation errors */
invalid_matrix_flag,
transform_clip_missing,

/* font cache validation errors */
metrics_wrong_type,
metrics_point_size_probably_bad,
scalar_block_wrong_type,
scalar_block_parent_mismatch,
scalar_block_too_small,
scalar_block_too_large,
invalid_metrics_range,
invalid_metrics_flags,
metrics_maxWidth_probably_bad,
font_wrong_type,
font_wrong_size,
invalid_font_platform,
invalid_lookup_range,
invalid_lookup_platform,
font_not_in_font_list,
metrics_not_in_metrics_list,

/* view device validation errors */
bad_device_private_flags,
bad_device_attributes,
invalid_device_number,
invalid_device_viewGroup,
invalid_device_bounds,
invalid_bitmap_in_device,
/* color set validation errors */
colorSet_wrong_type,
invalid_colorSet_viewDevice_owners,
invalid_colorSet_colorSpace,
invalid_colorSet_count,
```

```
   /* color profile validation errors */
   colorProfile_wrong_type,
   invalid_colorProfile_flags,
   invalid_colorProfile_response_count,

   /* internal backing store validation errors */
   backing_free_parent_mismatch,
   backing_store_parent_mismatch
   #endif
   };
```

QuickDraw GX non-debugging errors are listed in the section “Errors” beginning on page 3-6. Debugging errors are listed in the section “Errors” beginning on page 3-6.

## Warnings

QuickDraw GX provides you with an extended set of warnings in the debugging version and a reduced set of warnings in the non-debugging version. Each QuickDraw GX warning has a warning number described by the `gxGraphicsWarning` type definition and the `gxGraphicWarnings` enumeration:

```
typedef long gxGraphicsWarning;

enum gxGraphicWarnings {

   /* warnings about warnings */
   warning_stack_underflow = -26999,
   warning_stack_overflow,
   notice_stack_underflow,
   notice_stack_overflow,
   about_to_grow_heap,
   about_to_unload_objects,

   /* result went out of range */
   map_shape_out_of_range = -26950,
   move_shape_out_of_range,
   scale_shape_out_of_range,
   rotate_shape_out_of_range,
   skew_shape_out_of_range,
   map_transform_out_of_range,
   move_transform_out_of_range,
   scale_transform_out_of_range,
   rotate_transform_out_of_range,
   skew_transform_out_of_range,
   map_points_out_of_range,
```

```
   /* gave a parameter out of range */
   contour_out_of_range = -26900,
   index_out_of_range_in_contour,
   picture_index_out_of_range,
   color_index_requested_not_found,
   colorSet_index_out_of_range,
   index_out_of_range,
   count_out_of_range,
   length_out_of_range,
   font_table_index_out_of_range,
   font_glyph_index_out_of_range,
   point_out_of_range,
   profile_response_out_of_range,

      /* gxFont scaler warnings */
   font_scaler_no_output = -26850,
   font_scaler_fake_metrics,
   font_scaler_fake_linespacing,
   font_scaler_glyph_substitution,
   font_scaler_no_kerning_applied,

   /* might not be what you expected */
   character_substitution_took_place,
   unable_to_get_bounds_on_multiple_devices,
   font_language_not_found,
   font_not_found_during_unflattening,

   /*storage */
   unrecognized_stream_version,
   bad_data_in_stream

#ifdef debugging
   /*available only in debugging init */
   ,
   /* nonsense data */
   new_shape_contains_invalid_data = -26700,
   new_tag_contains_invalid_data,
   extra_data_passed_was_ignored,
   font_table_not_found,
   font_name_not_found,
```

```
/* doesn't make sense to do */
unable_to_traverse_open_contour_that_starts_or_
   ends_off_the_curve,
unable_to_draw_open_contour_that_starts_or_ends_
   off_the_curve,
cannot_dispose_default_shape,
cannot_dispose_default_style,
cannot_dispose_default_ink,
cannot_dispose_default_transform,
cannot_dispose_default_colorProfile,
cannot_dispose_default_colorSet,
shape_direct_attribute_not_set,

/* couldn't find what you were looking for */
point_does_not_intersect_port,
cannot_dispose_non_font,
face_override_style_font_must_match_style,
union_of_area_and_length_returns_area_only,
insufficient_coordinate_space_for_new_device,

/* other */
shape_passed_has_no_bounds,
tags_of_type_flst_removed,
translator_not_installed_on_this_grafport
#endif
};
```

Non-debugging warnings are listed in the section “Warnings” beginning on page 3-10.
Debugging warnings are listed in the section “Warnings” beginning on page 3-10.

## Notices

QuickDraw GX provides you with a set of notices in the debugging version, but no notices in the non-debugging version. Each QuickDraw GX notice has a notice number described by the `gxGraphicsNotice` type definition and the `gxGraphicNotices` enumeration:

```
typedef long gxGraphicsNotice;

#ifdef debugging
enum gxGraphicNotices {
   parameters_have_no_effect = -25999,
   attributes_already_set,
   caps_already_set,
   clip_already_set,
   color_already_set,
   curve_error_already_set,
   dash_already_set,
   default_colorProfile_already_set,
   default_ink_already_set,
   default_transform_already_set,
   default_shape_already_set,
   default_style_already_set,
   dither_already_set,
   encoding_already_set,
   face_already_set,
   fill_already_set,
   font_already_set,
   font_variations_already_set,
   glyph_positions_are_already_set,
   glyph_tangents_are_already_set,
   halftone_already_set,
   hit_test_already_set,
   ink_already_set,
   join_already_set,
   justification_already_set,
   mapping_already_set,
   pattern_already_set,
   pen_already_set,
   style_already_set,
   tag_already_set,
   text_attributes_already_set,
   text_size_already_set,
   transfer_already_set,
```

```
    translator_already_installed_on_this_grafport,
    transform_already_set,
    type_already_set,
    validation_level_already_set,
    viewPorts_already_set,
    viewPort_already_in_viewGroup,
    viewDevice_already_in_viewGroup,
    geometry_unaffected,
    mapping_unaffected,
    tags_in_shape_ignored,
    shape_already_in_primitive_form,
    shape_already_in_simple_form,
    shape_already_broken,
    shape_already_joined,
    cache_already_cleared,
    shape_not_disposed,
    style_not_disposed,
    ink_not_disposed,
    transform_not_disposed,
    colorSet_not_disposed,
    colorProfile_not_disposed,
    font_not_disposed,
    glyph_tangents_have_no_effect,
    glyph_positions_determined_by_advance,
    transform_viewPorts_already_set,
    directShape_attribute_set_as_side_effect,
    lockShape_called_as_side_effect,
    lockTag_called_as_side_effect,
    shapes_unlocked_as_side_effect,
    shape_not_locked,
    tag_not_locked,
    disposed_dead_caches,
    disposed_live_caches,
    low_on_memory,
    very_low_on_memory
    transform_references_disposed_viewPort
};
```

Debugging notices are listed in the section “Notices” beginning on page 3-27.

## Error, Warning, and Notice Number Ranges

QuickDraw GX specifies the defined ranges of error, warning, and notice numbers. The `gxFirstAppError`, `gxLastAppError`, `gxFirstAppWarning`, `gxLastAppWarning`, `gxFirstAppNotice`, and `gxLastAppNotice` types define the allowable ranges for application-defined errors, warnings, and notices.

```
#define gxFirstSystemError                -27999
#define gxFirstFatalError                 -27999
#define gxLastFatalError                  -27951
#define gxFirstNonfatalError              -27950
#define gxFirstFontScalerError            -27900
#define gxLastFontScalerError             -27851
#define gxFirstParameterError             -27850
#define gxFirstImplementationLimitError   -27800
#define gxFirstSystemDebuggingError       -27700
#define gxLastSystemError                 -27000
#define gxFirstAppError                   2097152
#define gxLastAppError                    4194303

#define gxFirstSystemWarning              -26999
#define gxFirstResultOutOfRangeWarning    -26950
#define gxFirstParameterOutOfRangeWarning -26900
#define gxFirstFontScalerWarning          -26850
#define gxFirstSystemDebuggingWarning     -26700
#define gxLastSystemWarning               -26000
#define gxFirstAppWarning                 5242880
#define gxLastAppWarning                  7340031

#define gxFirstSystemNotice               -25999
#define gxLastSystemNotice                -25500
#define gxFirstAppNotice                  7602146
#define gxLastAppNotice                   8388607
```

# Functions

This section describes the QuickDrawGX functions you can use to control the generation of errors, warnings, and notices.

## Error Posting and Handling

This section describes the QuickDraw GX functions you can use to

- obtain the first and last QuickDraw GX errors posted
- replace the current error name with another error name
- install the application-defined error handler function
- obtain the installed application-defined error handler function

### GXGetGraphicsError

You can use the `GXGetGraphicsError` function to obtain the first and last QuickDraw GX errors posted.

```
gxGraphicsError GXGetGraphicsError(gxGraphicsError *stickyError);
```

`stickyError`
On return, a pointer to the first error posted.

*function result* The last error posted.

**DESCRIPTION**

The `GXGetGraphicsError` function returns the last error posted, or 0 if no error has been posted. This function clears the last error so that all calls to this function return 0 until an error is posted.

The `stickyError` parameter, if not `nil`, is a pointer to the first error posted since the last call to the `GXGetGraphicsError` function. QuickDraw GX clears the `stickyError` parameter at the end of every call to the `GXGetGraphicsError` function.

**SEE ALSO**

The use of this function is described in the section “Obtaining Errors, Warnings, and Notices” beginning on page 3-30. Non-debugging errors that may be posted are listed in the section “Errors” beginning on page 3-6. Debugging errors that may be posted are listed in the section “Errors” beginning on page 3-6.

An alternative method of posting errors is to include an application-defined error handler. This topic is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXSetUserGraphicsError` function is used to install the error handler and is described on page 3-58.

## GXPostGraphicsError

You can use the `GXPostGraphicsError` function to replace the current QuickDraw GX error with another error.

```
void GXPostGraphicsError(gxGraphicsError error);
```

`error` The error to be posted.

**DESCRIPTION**

The `GXPostGraphicsError` function replaces the QuickDraw GX error about to be posted with an error message defined by the `error` parameter. You may use the QuickDraw GX errors or define your own error number and error name. This function stores the error posted so that subsequent calls to the `GXGetGraphicsError` function return the error substituted by this function.

The `GXPostGraphicsError` function is available only when the debugging version is installed.

**SPECIAL CONSIDERATIONS**

The error number must be within the range defined by QuickDraw GX. This range is bounded by error numbers –27999 through –27000, or is in the application range.

**SEE ALSO**

The use of this function is described in the section “Changing the Error, Warning, or Notice Posted” beginning on page 3-35.

Non-debugging errors that can be replaced are listed in the section “Errors” beginning on page 3-6. Non-debugging errors that can be replaced are listed in the section “Errors” beginning on page 3-6.

## GXSetUserGraphicsError

You can use the `GXSetUserGraphicsError` function to install an error handling function.

```
void GXSetUserGraphicsError(gxUserErrorFunction userFunction,
                            long reference);
```

`userFunction`
: The application's error handling function that is to be passed the error code.

`reference`
: A `long` value that is passed each time an error occurs. This value can be used by the application for any purpose.

#### DESCRIPTION

The `GXSetUserGraphicsError` function installs an application-defined error handling function. This function installs a function pointer that is called whenever an error is posted. Setting the `userFunction` parameter to `nil` removes the error handling function.

The `userFunction` parameter points to an application-defined error handler defined by the following type:

```
typedef void (*gxUserErrorProcPtr)(gxGraphicsError status,
               long reference);

typedef gxUserErrorProcPtr gxUserErrorFunction;
```

The second parameter is the `long` reference number. Whenever the application posts an error, the installed error handling function is called with the error number. The reference number is passed to the `GXSetUserGraphicsError` function.

You can install an error handler before calling the `GXEnterGraphics` function, but you should call the `GXNewGraphicsClient` function first. If you don't, `GXNewGraphicsClient` will be called for you.

#### SPECIAL CONSIDERATIONS

If the error number posted by the application is within the QuickDraw GX range of fatal errors, execution continues with undefined results. The fatal error range is bounded by error numbers –27999 and –27951.

If the error number posted by the application is within the QuickDraw GX range of nonfatal errors, execution continues, but results may be other than that expected. The nonfatal error range is bounded by error numbers –27950 and –27000.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXGetUserGraphicsError` function used to return a pointer to the application-defined error-handling function is described in the next section.

An alternative method of posting errors is to use the QuickDraw GX error messages. This topic is discussed in the section “Obtaining Errors, Warnings, and Notices” beginning on page 3-30.

The `GXGetGraphicsError` function described on page 3-56 is used to obtain the first and last QuickDraw GX errors posted.

The application-defined error handler is described on page 3-72.

## GXGetUserGraphicsError

You can use the `GXGetUserGraphicsError` function to obtain the currently installed application-defined error handler.

```
gxUserErrorFunction GXGetUserGraphicsError(long *reference);
```

`reference` A pointer to a `long` value that gets called each time an error occurs. This value can be used by the application for any purpose.

*function result* A pointer to the installed application-defined error handler function.

**DESCRIPTION**

The `GXGetUserGraphicsError` function returns a pointer to the function that the application uses to handle errors. The function returns `nil` if no application-defined error handler is provided.

If an error-handling function is installed and the `reference` parameter is not `nil`, then the `reference` parameter passed to the `GXSetUserGraphicsError` function is returned.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXSetUserGraphicsError` function used to install the error handler is described in the previous section.

An alternative method to the use of an application-defined error handler is the use of the QuickDraw GX error set.

The `GXGetGraphicsError` function, described in the section "Obtaining Errors, Warnings, and Notices" beginning on page 3-30, returns the first and last QuickDraw GX errors that have been posted.

## Warning Posting and Handling

This section describes the QuickDraw GX functions you can use to

- obtain the first and last QuickDraw GX warnings posted
- replace the current error name with another error name
- install the application-defined warning handler function
- obtain the installed application-defined warning handler function
- add a warning to the ignore warning stack
- remove the last warning to be added to the warning stack

## GXGetGraphicsWarning

You can use the `GXGetGraphicsWarning` function to obtain the first and last warning posted.

```
gxGraphicsWarning GXGetGraphicsWarning
                                (gxGraphicsWarning *stickyWarning);
```

`stickyWarning`
On return, a pointer to the first warning posted.

*function result* The last warning posted.

**DESCRIPTION**

The `GXGetGraphicsWarning` function returns the last warning posted, or 0 if none.

The `stickyWarning` parameter, if not `nil`, receives the first warning posted since the last call to the `GXGetGraphicsWarning` function. QuickDraw GX clears the `stickyWarning` parameter at the end of every call to the `GXGetGraphicsWarning` function.

**SEE ALSO**

The use of this function is described in the section“Obtaining Errors, Warnings, and Notices” beginning on page 3-30.

QuickDraw GX non-debugging warnings that may be posted are listed in the section “Warnings” beginning on page 3-10. Debugging warnings are listed in the section “Warnings” beginning on page 3-10.

An alternative method of posting warnings is to include an application-defined warning handler. This topic is described in the section “Changing the Error, Warning, or Notice Posted” beginning on page 3-35.

The `GXSetUserGraphicsWarning` function is used to install the warning handler and is described on page 3-62.

## GXPostGraphicsWarning

You can use the `GXPostGraphicsWarning` function to post your own warnings from your application.

```
void GXPostGraphicsWarning(gxGraphicsWarning warning);
```

`warning` The warning to be posted.

**DESCRIPTION**

The `GXPostGraphicsWarning` function replaces the QuickDraw GX warning about to be posted with a warning message defined by the `warning` parameter.

You may use the QuickDraw GX warnings or define your own warning number and warning name. This function stores the warning posted so that subsequent calls to the `GXGetGraphicsWarning` function return the warning substituted by this function.

If the warning to be posted is in the ignore warning stack, the warning is not posted and execution continues.

If an application-defined warning handler is provided, the warning is passed to the warning handler.

**SPECIAL CONSIDERATIONS**

The warning number must be within the range defined by QuickDraw GX. This range is bounded by warning numbers –26999 through –26000 or is in an application range.

**SEE ALSO**

The use of this function is described in the section “Changing the Error, Warning, or Notice Posted” beginning on page 3-35.

QuickDraw GX non-debugging warnings that may be replaced are listed in the section “Warnings” beginning on page 3-10. Debugging warnings are listed in the section “Warnings” beginning on page 3-10.

Ignoring warnings is discussed in the section “Ignoring Warnings and Notices” beginning on page 3-37.

## GXSetUserGraphicsWarning

You can use the `GXSetUserGraphicsWarning` function to install an application-defined warning handler.

```
void GXSetUserGraphicsWarning(gxUserWarningFunction userFunction,
                              long reference);
```

`userFunction`
The application’s warning function that is to be passed the warning code.

`reference` A `long` value that gets called each time a warning occurs. This value can be used by the application for any purpose.

**DESCRIPTION**

The `GXSetUserGraphicsWarning` function installs an application-defined warning handler. This function installs a function pointer that is called whenever a warning is posted. Setting the `userFunction` parameter to `nil` removes the error function.

The `userFunction` parameter points to an application-defined warning handler defined by the following type:

```
typedef void (*gxUserWarningProcPtr)(gxGraphicsWarning status,
               long refcon)

typedef gxUserWarningProcPtr gxUserWarningFunction;
```

The second parameter is the `long` reference parameter. Whenever a warning is posted by the application, the installed warning handler is called with the warning number. The reference number is passed to the `GXSetUserGraphicsError` function.

You can install a warning handler before calling the `GXEnterGraphics` function , but you should call the `GXNewGraphicsClient` function first. If you don’t, `GXNewGraphicsClient` will be called for you.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXGetUserGraphicsWarning` function described in the next section is used to return a pointer to the application-defined warning handler.

An alternative method of posting warnings is to use the QuickDraw GX warning messages. This topic is discussed in the section “Obtaining Errors, Warnings, and Notices” beginning on page 3-30.

The `GXGetGraphicsError` function described on page 3-56 is used to obtain the first and last QuickDraw GX errors posted.

The application-defined warning handler is described on page 3-73.

## GXGetUserGraphicsWarning

You can use the `GXGetUserGraphicsWarning` function to obtain the currently installed application-defined warning handler.

```
gxUserWarningFunction GXGetUserGraphicsWarning(long *reference);
```

`reference` A `long` value that gets called each time a warning occurs. This value can be used by your application for any purpose.

*function result* A pointer to the installed application-defined warning handler.

**DESCRIPTION**

The `GXGetUserGraphicsWarning` function returns a pointer to the function that the application uses to handle warnings. The function returns `nil` if no application-defined warning handler is provided.

If a warning handler is installed and the `reference` parameter is not `nil`, then the `reference` parameter passed to the `GXSetUserGraphicsWarning` function is returned.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXSetUserGraphicsWarning` function used to install the warning handler is described in the previous section.

An alternative method to the use of an application-defined warning handler is the use of the QuickDraw GX warnings.

The GXGetGraphicsWarning function, described in the section "Obtaining Errors, Warnings, and Notices" beginning on page 3-30, returns the first and last QuickDraw GX warnings that have been posted.

## GXIgnoreGraphicsWarning

You can use the GXIgnoreGraphicsWarning function to ignore warnings.

```
void GXIgnoreGraphicsWarning(gxGraphicsWarning warning);
```

`warning` The warning number or warning name to ignore.

#### DESCRIPTION

The GXIgnoreGraphicsWarning function adds the warning to be ignored to the ignore warning stack. The posting of warnings is suppressed for all warnings on the ignore warning stack. Warnings may be removed from the ignore warnings stack by the use of the GXPopGraphicsWarning function.

You may use any Quickdraw GX warning numbers and warning names or, if you have installed an application-defined warning handler, you may use your own warning numbers and warning names, as long as they use a numbering system different than that provided by QuickDraw GX.

#### SPECIAL CONSIDERATIONS

The GXIgnoreGraphicsWarning function saves warning numbers in a warning stack of limited size, so that a limited number of warnings can be ignored at one time. If the GXIgnoreGraphicsWarning function has been called too many times with no matching calls to the GXPopGraphicsWarning function, subsequent calls to the GXIgnoreGraphicsWarning function do not cause the warning to be ignored and a warning_stack_overflow warning is posted.

#### ERRORS, WARNINGS, AND NOTICES

**Warnings**
warning_stack_overflow

**SEE ALSO**

The use of this function is described in the section “Ignoring Warnings and Notices” beginning on page 3-37.

QuickDraw GX non-debugging warnings that may be posted are listed in the section “Warnings” beginning on page 3-10. Debugging warnings are listed in the section “Warnings” beginning on page 3-10.

The `GXPopGraphicsWarning` function is described in the next section.

## GXPopGraphicsWarning

You can use the `GXPopGraphicsWarning` function to remove ignore warnings from the ignore warning stack.

```
void GXPopGraphicsWarning(void);
```

**DESCRIPTION**

The `GXPopGraphicsWarning` function removes the last warning placed on the ignore warning stack by the `GXIgnoreGraphicsWarning` function. The `GXPopGraphicsWarning` function removes warnings from the stack in the opposite order that they were added to the stack (last in, first out). Calls to the `GXIgnoreGraphicsWarning` and `GXPopGraphicsWarning` functions can be nested.

**SPECIAL CONSIDERATIONS**

If no warning is on the warning stack when you call this function, a `warning_stack_underflow` warning is posted.

**ERRORS, WARNINGS, AND NOTICES**

**Warnings**
`warning_stack_underflow`

**SEE ALSO**

The use of this function is described in the section “Ignoring Warnings and Notices” beginning on page 3-37.

QuickDraw GX non-debugging warnings that may be added to and removed from the ignore warning stack are listed in the section “Warnings” beginning on page 3-10. Debugging warnings are listed in the section “Warnings” beginning on page 3-10.

The `GXIgnoreGraphicsWarning` function is described in the previous section.

## Notice Posting and Handling

This section describes the QuickDraw GX functions you can use to

- obtain the first and last notice posted
- install the current notice
- install an application-defined function for posted notices
- obtain an application-defined notice handler function for posted notices
- add a notice to the ignore notice stack
- remove the last notice to be added to the notice stack

## GXGetGraphicsNotice

You can use the `GXGetGraphicsNotice` function to obtain the first and last notices posted.

```
gxGraphicsNotice GXGetGraphicsNotice
                                    (gxGraphicsNotice *stickyNotice);
```

`stickyNotice`
On return, a pointer to the first notice posted.

*function result* The last notice posted.

**DESCRIPTION**

The `GXGetGraphicsNotice` function returns the last notice posted, or 0 if none. The `stickyNotice` parameter, if not `nil`, receives the first notice posted since the last call to the `GXGetGraphicsNotice` function.

**SPECIAL CONSIDERATIONS**

QuickDraw GX clears the `stickyNotice` argument at the end of every call to the `GXGetGraphicsNotice` function. It always returns 0 on non-debugging versions.

**SEE ALSO**

The use of this function is described in the section“Obtaining Errors, Warnings, and Notices” beginning on page 3-30.

An alternative method of posting notices is to include an application-defined notice handler. This topic is described in the section “Changing the Error, Warning, or Notice Posted” beginning on page 3-35.

The `GXSetUserGraphicsNotice` function that is used to install the notice handler is described on page 3-68.

## GXPostGraphicsNotice

You can use the `GXPostGraphicsNotice` function to post your own notices from inside your application.

```
void GXPostGraphicsNotice(gxGraphicsNotice notice);
```

`notice` The notice to be posted.

**DESCRIPTION**

The `GXPostGraphicsNotice` function replaces the QuickDraw GX notice about to be posted with a notice message defined by the `notice` parameter.

You may use the QuickDraw GX notices or define your own notice number and notice name. This function stores the posted notice so that subsequent calls to the `GXGetGraphicsNotice` function return the notice substituted by this function.

If the notice to be posted is in the ignore notice stack, the notice is not posted and execution continues. Ignoring notices is discussed in the section “Ignoring Warnings and Notices” beginning on page 3-37.

If an application-defined notice handler is provided, the notice is passed to the handler.

The `GXIgnoreGraphicsNotice` function has no effect in the non-debugging version.

**SPECIAL CONSIDERATIONS**

The notice number must be within the range defined by QuickDraw GX. This range is bounded by notice numbers –25999 through –25500 or is in an application range.

**SEE ALSO**

The use of this function is described in the section “Changing the Error, Warning, or Notice Posted” beginning on page 3-35.

Notice handlers are discussed in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

## GXSetUserGraphicsNotice

You can use the GXSetUserGraphicsNotice function to install a notice handler.

```
void GXSetUserGraphicsNotice(gxUserNoticeFunction userFunction,
                             long reference);
```

`userFunction`
The application function that is to be passed the notice result code.

`reference` A `long` value that is called each time a notice occurs. This value can be used by the application for any purpose.

**DESCRIPTION**

The GXSetUserGraphicsNotice function installs an application-defined notice-handling function. This function installs a function pointer that is called whenever a notice is posted. Setting the `userFunction` parameter to `nil` removes the notice function.

The `userFunction` parameter points to an application-defined notice handler defined by the following type:

```
typedef void (*gxUserNoticeProcPtr)(gxGraphicsNotice status,
               long reference)

typedef gxUserNoticeProcPtr gxUserNoticeFunction;
```

The second parameter is the `long` reference. Whenever a notice is posted by the application, the installed notice handler is called with the notice number. The reference number is passed to the GXSetUserGraphicsNotice function.

You can install a notice handler before calling the GXEnterGraphics function, but you should call the GXNewGraphicsClient function first. If you don't, it will be called for you.

The GXSetUserGraphicsNotice function has no effect in the non-debugging version.

**SEE ALSO**

The use of this function is described in the section "Installing an Error, Warning, or Notice Handler" beginning on page 3-40.

The GXGetUserGraphicsNotice function used to return a pointer to the application-defined notice handler is described in the next section.

The application-defined notice handler is described on page 3-74.

## GXGetUserGraphicsNotice

You can use the `GXGetUserGraphicsNotice` function to obtain the currently installed application-defined notice handler.

```
gxUserNoticeFunction GXGetUserGraphicsNotice(long *reference);
```

`reference` A `long` value that is called each time a notice occurs. This value can be used by the application for any purpose.

*function result* A pointer to the installed application-defined notice handler.

**DESCRIPTION**

The `GXGetUserGraphicsNotice` function returns a pointer to the function that the application uses to handle notices. The function returns `nil` if no application-defined notice handler is installed.

If a notice handler function is installed and the `reference` parameter is not `nil`, then the `reference` parameter passed to the `GXSetUserGraphicsNotice` function is returned.

The `GXGetUserGraphicsNotice` function has no effect in the non-debugging version.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The `GXSetUserGraphicsNotice` function used to install the notice handler is described in the previous section.

An alternative method to the use of an application-defined notice handler is the use of QuickDraw GX notices. The `GXGetGraphicsNotice` function, described in the section “Obtaining Errors, Warnings, and Notices” beginning on page 3-30, returns the first and last QuickDraw GX notices that have been posted.

## GXIgnoreGraphicsNotice

You can use the `GXIgnoreGraphicsNotice` function to ignore QuickDraw GX notices that may occur when specific parts of your application execute.

```
void GXIgnoreGraphicsNotice(gxGraphicsNotice notice);
```

`notice` The graphics notice number or name to ignore.

### DESCRIPTION

The `GXIgnoreGraphicsNotice` function adds the notice to be ignored to the ignore notice stack. The posting of notices is suppressed for all notices on the ignore notice stack. Notices may be removed from the ignore notice stack by the use of the `GXPopGraphicsNotice` function.

You may use any QuickDraw GX notice numbers and notice names or, if you have installed an application-defined notice handler, you may use your own notice numbers and notice names, as long as they use a numbering system different than that provided by QuickDraw GX.

This function has no effect in non-debugging versions

### SPECIAL CONSIDERATIONS

The `GXIgnoreGraphicsNotice` function saves notice numbers in a warning stack of limited size. If the `GXIgnoreGraphicsNotice` function has been called too many times with no matching calls to the `GXPopGraphicsNotice` function, subsequent calls to the `GXIgnoreGraphicsNotice` function do not cause the notice to be ignored and a `notice_stack_overflow` warning is be posted.

### ERRORS, WARNINGS, AND NOTICES

#### Warnings

`notice_stack_overflow`

### SEE ALSO

The use of this function is described in the section “Ignoring Warnings and Notices” beginning on page 3-37.

QuickDraw GX notices that may be posted are listed in the section “Notices” beginning on page 3-27.

The `GXPopGraphicsNotice` function is described in the next section.

## GXPopGraphicsNotice

You can use the `GXPopGraphicsNotice` function to remove notices from the ignore notice stack.

```
void GXPopGraphicsNotice(void);
```

**DESCRIPTION**

The `GXPopGraphicsNotice` function removes the last notice added to the ignore notice stack by the `GXIgnoreGraphicsNotice` function. The `GXPopGraphicsNotice` function removes notices from the stack in the opposite order that they were added to the stack (last in, first out). Calls to the `GXIgnoreGraphicsNotice` function and the `GXPopGraphicsNotice` function can be nested.

The `GXPopGraphicsNotice` function has no effect in the non-debugging version.

**SPECIAL CONSIDERATIONS**

If no notice is on the ignore notice stack when you call this function, a `notice_stack_underflow` warning is posted.

**ERRORS, WARNINGS, AND NOTICES**

**Warnings**

`notice_stack_underflow`

**SEE ALSO**

The use of this function is described in the section “Ignoring Warnings and Notices” beginning on page 3-37.

QuickDraw GX notices that may be added and removed from the ignore notice stack are listed in the section “Notices” beginning on page 3-27.

The `GXIgnoreGraphicsNotice` function is described in the previous section.

## Application-Defined Functions

QuickDraw GX supports application-defined error, warning, and notice handlers. These handlers are installed by the use of the `GXSetUserGraphicsError`, `GXSetUserGraphicsWarning`, and `GXSetUserGraphicsNotice` functions.

### MyUserGraphicsError

You can use the `MyUserGraphicsError` function to provide an application-defined error handler for your application.

```
void MyUserGraphicsError(gxGraphicsError error, long reference);
```

`error` The QuickDraw GX error being passed to the handler.

`reference` A `long` value passed each time that an error occurs. This value can be used by the error handler for any purpose.

#### DESCRIPTION

The `MyUserGraphicsError` function is called with the error number posted by the failed function. The `MyUserGraphicsError` function can evaluate the error and respond in any appropriate manner.

The error handler is enabled and disabled by the use of the `GXSetUserGraphicsError` function. If its parameter is set to `nil`, the error handler is disabled. If its parameter is not `nil`, the error handler is enabled and all errors detected by QuickDraw GX are passed to the error handler for processing and possible response.

The `GXGetUserGraphicsError` function returns the currently installed application-defined error handler.

#### SEE ALSO

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” on page 3-40.

QuickDraw GX non-debugging errors that may be sent to the error handler are listed in section “Errors” beginning on page 3-6. Debugging errors are listed in the section “Errors” beginning on page 3-6.

The `GXSetUserGraphicsError` function is described on page 3-58.

The `GXGetUserGraphicsError` function is described on page 3-59.

## MyUserGraphicsWarning

You can use the `MyUserGraphicsWarning` function to provide an application-defined warning handler for your application.

```
void MyUserGraphicsWarning(gxGraphicsWarning warning,
                           long reference);
```

`warning` The QuickDraw GX warning being passed to the handler.

`reference` A `long` value passed each time that a warning occurs. This value can be used by the warning handler for any purpose.

**DESCRIPTION**

The `MyUserGraphicsWarning` function is called with the warning number posted by the defective function. The `MyUserGraphicsWarning` function can evaluate the warning and respond in any appropriate manner.

The warning handler is enabled and disabled by the use of the `GXSetUserGraphicsWarning` function. If its parameter is set to `nil`, the warning handler is disabled. If its parameter is not `nil`, the warning handler is enabled and all warnings detected by QuickDraw GX are passed to the warning handler for processing and possible response.

The `GXGetUserGraphicsWarning` function returns the currently installed application-defined warning handler.

**SEE ALSO**

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” on page 3-40.

Warnings that may be sent to the warning handler are listed in the section “Warnings” beginning on page 3-10.

The `GXSetUserGraphicsWarning` function is described on page 3-62.

The `GXGetUserGraphicsWarning` function is described on page 3-63.

## MyUserGraphicsNotice

You can use the `MyUserGraphicsNotice` function to provide an application-defined notice handler for your application.

```
void MyUserGraphicsNotice(gxGraphicsNotice notice,
                          long reference);
```

`notice` The QuickDraw GX notice being passed to the handler.

`reference` A `long` value passed each time that a notice occurs. This value can be used by the notice handler for any purpose.

### DESCRIPTION

The `MyUserGraphicsNotice` function is called with the notice number posted by the defective function. The `MyUserGraphicsNotice` function can evaluate the notice and respond in any appropriate manner.

The notice handler is enabled and disabled by the use of the `GXSetUserGraphicsNotice` function. If its parameter is set to `nil`, the notice handler is disabled. If its parameter is not `nil`, the notice handler is enabled and all notices detected by QuickDraw GX are passed to the notice handler for processing and possible response.

The `GXGetUserGraphicsNotice` function returns the currently installed application-defined notice handler. This function will never be called in the non-debugging version of QuickDraw GX.

### SEE ALSO

The use of this function is described in the section “Installing an Error, Warning, or Notice Handler” on page 3-40.

Notices that may be sent to the notice handler are listed in the section “Notices” beginning on page 3-27.

The `GXSetUserGraphicsNotice` function is described on page 3-68.

The `GXGetUserGraphicsNotice` function is described on page 3-69.

# Summary of Errors, Warnings, and Notices

## Constants and Data Types

### QuickDraw GX Errors

```
typedef long gxGraphicsError
```

### QuickDraw GX Warnings

```
typedef long gxGraphicsWarning
```

### QuickDraw GX Notices

```
typedef long gxGraphicsNotice
```

### Application-Defined Handlers

```
typedef void                (*gxUserErrorProcPtr)(gxGraphicsError status,
                             long refcon)
typedef gxUserErrorProcPtr gxUserErrorFunction;
typedef void                (*gxUserWarningProcPtr)(gxGraphicsWarning
                             status,long refcon)
typedef gxUserWarningProcPtr gxUserWarningFunction;
typedef void                (*gxUserNoticeProcPtr)(gxGraphicsNotice status,
                             long refcon)
typedef gxUserNoticeProcPtr gxUserNoticeFunction;
```

## Functions

### Error Posting and Handling

```
gxGraphicsError GXGetGraphicsError
                            (gxGraphicsError *stickyError);
void GXPostGraphicsError    (gxGraphicsError error);
void GXSetUserGraphicsError (gxUserErrorFunction userFunction,
                             long reference);
gxUserErrorFunction GXGetUserGraphicsError
                            (long *reference);
```

### Warning Posting and Handling

```
gxGraphicsWarning GXGetGraphicsWarning
                            (gxGraphicsWarning *stickyWarning);
void GXPostGraphicsWarning  (gxGraphicsWarning warning);
void GXSetUserGraphicsWarning
                            (gxUserWarningFunction userFunction,
                             long reference);
gxUserWarningFunction GXGetUserGraphicsWarning
                            (long *reference);
void GXIgnoreGraphicsWarning
                            (gxGraphicsWarning warning);
void GXPopGraphicsWarning   (void);
```

### Notice Posting and Handling

```
gxGraphicsNotice GXGetGraphicsNotice
                            (gxGraphicsNotice *stickyNotice);
void GXPostGraphicsNotice   (gxGraphicsNotice notice);
void GXSetUserGraphicsNotice
                            (gxUserNoticeFunction userFunction,
                             long reference);
gxUserNoticeFunction GXGetUserGraphicsNotice
                            (long *reference);
void GXIgnoreGraphicsNotice
                            (gxGraphicsNotice notice);
void GXPopGraphicsNotice    (void);
```

## Application-Defined Functions

```
void MyUserGraphicsError    (gxGraphicsError error, long reference);
void MyUserGraphicsWarning  (gxGraphicsWarning warning, long reference);
void MyUserGraphicsNotice   (gxGraphicsNotice notice, long reference);
```