This chapter describes the QuickDraw GX application debugging environment and the functions and utilities that you can use to debug your application. Read this chapter if you are developing a QuickDraw GX application and want to use these features.

Before reading this chapter, you should be familiar with the debugging and non-debugging versions of QuickDraw GX described in the chapter "Errors, Warnings, and Notices" in this book. You should also read the chapter "Introduction to QuickDraw GX" in *Inside Macintosh: QuickDraw GX Objects*.

For more information on debugging printing applications, see *Inside Macintosh: QuickDraw GX Printing* and *Inside Macintosh: QuickDraw GX Printing Extensions and Drivers*.

This chapter introduces the QuickDraw GX debugging environment. It then describes how to use this environment during application development to

- analyze drawing problems
- validate public and internal function parameters for all allocated objects
- validate public and internal function parameters for specific objects
- distinguish between application and QuickDraw GX bugs
- detect corrupted objects
- install a debugging function
- use the GraphicsBug utility

This chapter also contains reference information for all data types and functions associated with QuickDraw GX debugging.

About QuickDraw GX Debugging

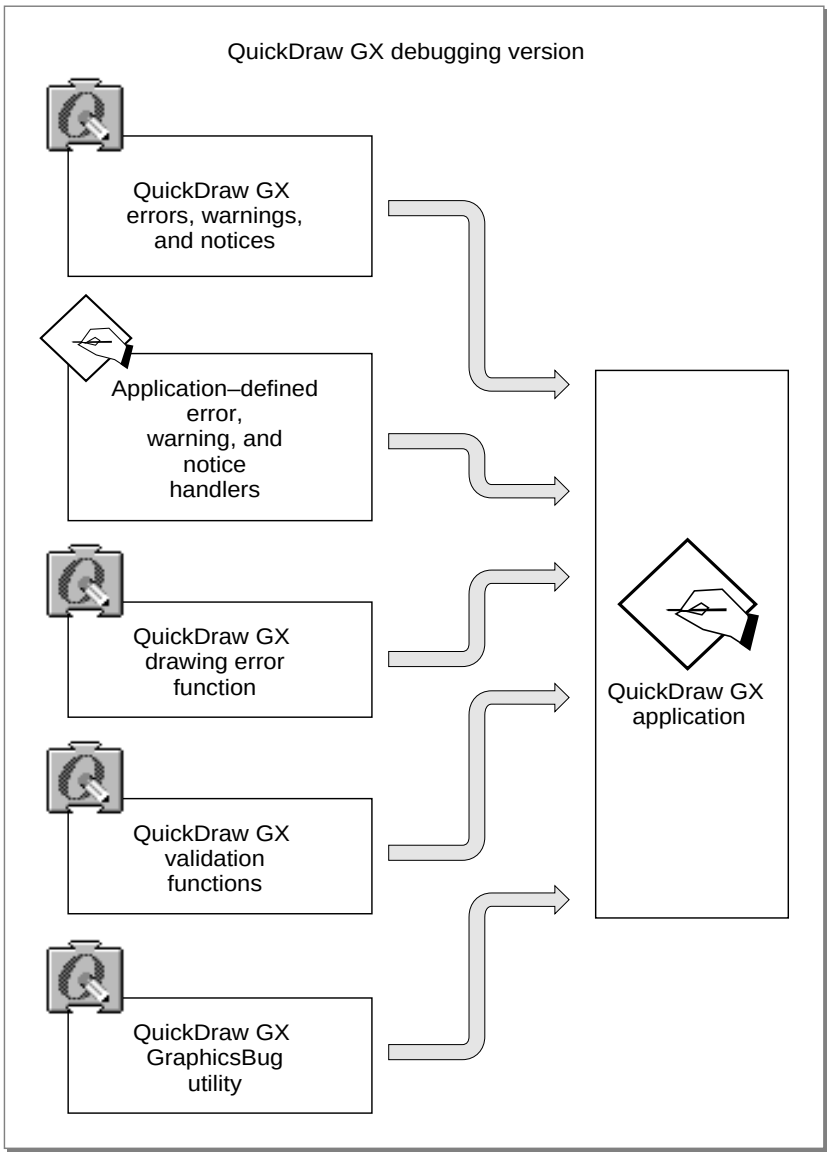
QuickDraw GX provides both a **debugging environment** and a **non-debugging environment.** The non-debugging environment is present whenever you install the non-debugging version of QuickDraw GX. You install the non-debugging version after completely debugging your application. Users of your application will use the non-debugging version of QuickDraw GX.

You can develop applications that use QuickDraw GX graphics and typography functions using the QuickDraw GX debugging environment. The debugging environment consists of

- the QuickDraw GX debugging version
- QuickDraw GX errors, warnings, and notices
- application-defined error, warning, and notice handlers
- a QuickDraw GX drawing error function
- QuickDraw GX validation functions
- the QuickDraw GX GraphicsBug utility

Figure 4-1 shows the QuickDraw GX application development environment.

Figure 4-1 The QuickDraw GX debugging environment



As a direct result of the extensive error, warning, and notice checking the debugging environment performs, the debugging version of QuickDraw GX is significantly slower than that of the non-debugging environment. Invoking additional optional error checking using the validation functions further affects performance.

Debugging Version of QuickDraw GX

You should use the debugging version of QuickDraw GX when you are writing and debugging applications. This version provides an extensive set of errors, warnings, and notices to help you understand the problems you may encounter during the execution of your application. In addition, this version provides special functions that allow you to manage errors, warnings, and notices and to provide public and private error validation.

The debugging version runs slower than the non-debugging version. The reasons for this are that the debugging version:

- performs additional error checking
- posts additional errors, warnings, and notices
- does not provide speed optimization, such as in-line functions
- generates MacsBug messages
- provides additional debugging functions, such as validation

To determine if the debugging or non-debugging version of QuickDraw GX is installed, see the chapter “QuickDraw GX and the Macintosh Environment.”

QuickDraw GX Errors, Warnings, and Notices

QuickDraw GX posts errors, warnings, and notices whenever an execution problem occurs while an application is running. You can obtain errors, warnings, and notices by polling or by the use of application-defined error, warning, and notice handlers.

The debugging and non-debugging versions of QuickDraw GX and the errors, warnings, and notices that may be posted from each version are described in the chapter “Errors, Warnings, and Notices.”

Application-Defined Error, Warning, and Notice Handlers

You can use error, warning, and notice handlers to manage problems that occur when your application is running. When QuickDraw GX detects an error, warning, or notice it will call your handler. Your function can then respond accordingly. You can also use error and warning handlers with the non-debugging version of QuickDraw GX to provide part of your user interface.

Application-defined error, warning, and notice handlers are described in the section “Installing an Error, Warning, or Notice Handler” beginning on page 3-40.

The Drawing Error Function

The debugging version of QuickDraw GX provides a drawing error function that you can use if you have run your application and get an unexpected result. This function reparses the entire QuickDraw GX operation, analyzes your application's draw procedure, and posts a single error that will assist you in determining what went wrong with your application. The drawing error function is described in the section "Analyzing Drawing Problems" beginning on page 4-8.

Validation Functions

The debugging version of QuickDraw GX provides validation for applications using graphics and typographic functions, but does not provide validation for QuickDraw GX printing functions.

The validation functions check function parameters of allocated objects to see if they are valid. If QuickDraw GX finds one or more parameters of a function to be invalid, it posts a validation error. All of the validation errors that may be posted are listed in the chapter "Errors, Warnings, and Notices."

There are two modes of validation that control when validation occurs:

- public validation
- internal validation

Public validation occurs whenever the public validation flag is set and your application uses a public function. A public function is any function that you use in your application. This is the mode of validation developers use most.

Internal validation occurs whenever the internal validation flag is set and your application uses a public function and whenever QuickDraw GX uses one of its internal (private) functions. Application developers do not usually use internal validation. Internal validation performs checking on functions that you have no control over. As a result, you will rarely need to perform this type of validation. However, QuickDraw GX provides internal validation to allow you to distinguish between bugs that appear in public functions and bugs that are present in the QuickDraw GX internal functions, as discussed in the section "Distinguishing Between Application Bugs and QuickDraw GX Bugs" beginning on page 4-22.

There are three levels of validation that control what is checked during validation:

- type validation
- structure validation
- all object validation

The validation these three levels provide is cumulative and progressively more complex. For example, all object validation includes type validation and structure validation.

In addition to these three levels, there are separate object validation functions.

Type validation confirms the validity of references to object types. For example, when you call the `GXDrawShape` function, type validation confirms that a shape type is passed.

Structure validation confirms the validity of references to object types and the properties of the function, and also checks internal caches. For example, when you call the `GXDrawShape` function, structure validation confirms not only that the function passes a shape, but also confirms the validity of the properties specified in the shape's style, ink, and transform objects.

All object validation confirms the validity of references to a specific object type, the validity of the properties of all objects, and all internal caches.

Specific object validation functions are used to confirm that all references to a specific object type are valid, that the properties of all objects are valid, and that all internal caches built for the specific object type are valid. Specific object validation functions are provided for shapes, styles, inks, transforms, color sets, color profiles, tags, view devices, view ports, view groups, and graphics clients.

It is important to note that not all parameters of all functions are checked by validation. Validation does not check scalars and structures, such as bitmaps and dash records.

For example, the second parameter of the `GXSetShapePen` function is the pen size. If you pass a negative value to the second parameter, QuickDraw GX will not post a validation error. Fortunately, QuickDraw GX often provides an overlap in its debugging capabilities, and in this case, the `GXSetShapePen` function would post an error indicating that the size is invalid.

Validation does check

- objects that are indicated by pointer values, such as shapes
- objects that are indicated by references, such as view devices

Note
You should not make an application dependent on whether an object is referred to by pointer or reference. This is subject to change in future versions of QuickDraw GX. ◆

You can enable validation selectively over the selected problem area of code. Rather than turning validation on at the beginning of your application, you may find it is more useful to concentrate on an area where a problem is suspected and to turn validation on and off selectively in that area or selectively use the specific object validation functions.

MacsBug and GraphicsBug

Both the debugging and non-debugging versions of QuickDraw GX support MacsBug and GraphicsBug. **MacsBug** is Apple Computer, Inc.'s, assembly-language debugger that was developed for Macintosh programmers. MacsBug is not very useful for debugging QuickDraw GX applications because GX data structures are private. For additional information about MacsBug, see *MacsBug Reference and Debugging Guide.*

GraphicsBug is Apple Computer Inc.'s symbolic debugger for QuickDraw GX applications. This utility assists in finding bugs by allowing you to display and check QuickDraw GX objects. GraphicsBug is modeled after MacsBug. In fact, many of the commands are similar.

The use of GraphicsBug to analyze a QuickDraw GX graphics client heap is described in the section "Debugging With GraphicsBug" beginning on page 4-23.

Using QuickDraw GX Debugging

You can use the QuickDraw GX debugging environment to help you debug your application. This section shows how you can

- determine why a shape didn't draw
- validate using public, internal, or object modes of validation
- validate types, structures, and all objects
- validate memory
- distinguish between QuickDraw GX bugs and application bugs
- validate public objects
- analyze the QuickDraw GX graphics heap with the GraphicsBug utility

Analyzing Drawing Problems

If you have run your application and a shape didn't draw as you anticipated, you can use the `GXGetShapeDrawError` function to have QuickDraw GX analyze why the shape didn't draw correctly. This function checks the content of a shape and all of the objects referenced by the shape for a condition that explains why the shape has no visible effect when drawn. As a result, `GXGetShapeDrawError` returns a single **drawing error** from the `gxDrawErrors` enumeration that may describe why the shape failed to draw correctly. The `gxDrawErrors` enumeration is listed in the section "Drawing Errors" beginning on page 4-29. The `GXGetShapeDrawError` function is described on page 4-33.These errors should not be confused with `gxGraphicsErrors`.

If the drawing was completed successfully, QuickDraw GX posts the `NoDrawError` drawing error. If you don't see the drawing, remember that it may have been drawn to a different view device or may have just redrawn over the previous shape that was drawn. The posting of a `NoDrawError` drawing error does not mean that the shape drawn is the one you expected or the correct shape. It just means that QuickDraw GX detected no drawing problems during the processing of the shape drawn.

The drawing error QuickDraw GX posts is selected from a special subset of the QuickDraw GX error codes. This set of drawing error codes is structured with respect to the stage in the **drawing process sequence** that the drawing failed. The earliest stage of failure will be described in the posted drawing error. The single error code posted attempts to indicate the reason that you do not see the drawing that you anticipated.

Drawing errors are grouped into categories that correspond to the approximate sequence of QuickDraw GX processing, as shown in Table 4-1.

Table 4-1 QuickDraw GX drawing process sequence

Drawing process sequence	Object processed
1	Shape type
2	Style
3	Ink
4	Transform
5	View port
6	View device

The processing sequence is also the sequence of drawing errors posted. QuickDraw GX posts the first drawing error that is detected. It does not post subsequent drawing errors until the error posted earlier in the process sequence is corrected. For example, if an application attempts to draw a defective shape with a defective view port, QuickDraw GX posts a single shape type drawing error and does not post a view port drawing error. This is because QuickDraw GX analyzes the integrity of the shape earlier in the drawing process. It analyzes the integrity of the view port toward the end of the process. Once you correct the defective shape, QuickDraw GX can detect the defective view port in subsequent analysis with the `GXGetShapeDrawError` function.

Table 4-2 shows the `GXGetShapeDrawError` function **shape type** drawing errors that QuickDraw GX may post.

Table 4-2 Shape type drawing errors

Error	Description
`shape_emptyType`	An empty type doesn't have an area to draw.
`shape_inverse_fullType`	An inverse full type doesn't have an area to draw.
`rectangle_zero_width`	The rectangle doesn't have an area to draw.
`rectangle_zero_height`	The rectangle doesn't have an area to draw.
`polygon_empty`	There is no contour to draw.
`path_empty`	There is no contour to draw.
`bitmap_zero_width`	The bitmap doesn't have an area to draw.
`bitmap_zero_height`	The bitmap doesn't have an area to draw.
`text_empty`	There is no character to draw.
`glyph_empty`	There is no glyph to draw.
`layout_empty`	There is no layout to draw.
`picture_empty`	There is no shape in the picture.
`shape_no_fill`	The shape fill is set to `gxNoFill`, which will not draw.
`shape_no_enclosed_area`	There is no enclosed area to draw.
`shape_no_enclosed_pixels`	There is an enclosed area, but it is so small that it does not cross any pixel centers.
`shape_very_small`	There is a shape to draw, but it is extremely small (on the order of the size of a pixel).
`shape_very_large`	Part of the shape may be drawn outside the bounds of the coordinate system (±32,768).
`shape_contours_cancel`	The shapes contours overlap and cancel each other out.

Table 4-3 shows the `GXGetShapeDrawError` function style drawing errors.

Table 4-3 Style drawing errors

Error	Description
`pen_too_small`	The pen width is so small that it doesn't enclose any pixels and therefore doesn't draw.
`text_size_too_small`	The text size is so small that it doesn't enclose any pixels and therefore doesn't draw.
`dash_empty`	The dash shape was specified as an empty type shape.
`start_cap_empty`	The start cap shape was specified as an empty type shape.
`pattern_empty`	The pattern shape was specified as an empty type shape.
`textFace_Empty`	Each layer of the text face has a shape fill equal to `gxNoFill`.
`shape_primitive_empty`	The original shape enclosed an area. There is no stylized shape to draw. An example is a pattern shape that contains overlapping patterns that cancel.
`shape_primitive_very_small`	There is a shape to draw, but it is extremely small (on the order of the size of a pixel). An example is a scaled transform that shrinks the shape.

Table 4-4 shows the `GXGetShapeDrawError` function ink drawing errors.

Table 4-4 Ink drawing errors

Error	Description
`transfer_equals_noMode`	The transfer mode `gxNoMode` suppresses drawing.
`transfer_matrix_ignores_source`	The transfer mode's mapping scales all values greater than 1 or less than 0 and the `overComponent` flag is not set.
`transfer_matrix_ignores_device`	The transfer mode's mapping scales all values greater than 1 or less than 0 and the `overComponent` flag is not set.
`transfer_source_reject`	The color is not within the source minimum and the source maximum.
`transfer_mode_ineffective`	The transfer mode has no effect on the device. An example is a blend with an operand of 0.
`colorSet_no_entries`	There are no colors in the color set so there is nothing to draw.
`bitmap_colorSet_one_entry`	The bitmap drew, but it is probably not the desired result, since all colors map to the one color of the entry. An example is when the colors are off the end of the color set.

Table 4-5 shows the `GXGetShapeDrawError` function transform drawing errors.

Table 4-5 Transform drawing errors

Error	Description
`transform_scale_too_small`	The transform has reduced the shape to less than 1/72 inch. You may see a few pixels drawn, depending on the resolution of your view port.

`transform_map_too_large`
`transform_move_too_large`
`transform_scale_too_large`
`transform_rotate_too_large`
`transform_perspective_too_large`
`transform_skew_too_large`
	The transform has moved all or part of the shape outside the bounds of the coordinate system (±32,768). This may be the result of a move, scale, rotate, perspective, or skew transformation.

`transform_clip_no_intersection`	The clip shape does not intersect any view port.
`transform_clip_empty`	The transform clip is an empty type shape.
`transform_no_viewPorts`	The number of entries in the view port list is zero.

Table 4-6 shows the `GXGetShapeDrawError` function view port drawing errors.

Table 4-6 View port drawing errors

Error	Description
`viewPort_disposed`	The view port that was to be drawn to has already been disposed of. There is no view port to draw to.
`viewPort_clip_empty`	The view port clip is an empty type shape.
`viewPort_clip_no_intersection`	The view port clip does not intersect the view device.
`viewPort_scale_too_small`	The map to global space has been completed. The object is less than 1/72 inch. You may see a few pixels drawn, depending on the resolution of your view port.

`viewPort_map_too_large`
`viewPort_move_too_large`
`viewPort_scale_too_large`
`viewPort_rotate_too_large`
`viewPort_perspective_too_large`
`viewPort_skew_too_large,`
	The view port mapping has moved all or part of the shape outside the bounds of the coordinate system (±32,768). This may be the result of a move, scale, rotate, perspective, or skew transformation.

`viewPort_viewGroup_offscreen`	The shape is drawn to an off-screen view device. This may be normal. This error is returned to alert you in the event that the drawing result was unexpected.

Table 4-7 shows the GXGetShapeDrawError function view device drawing errors.

Table 4-7 View device drawing errors

Error	Description
viewDevice_clip_no_intersection	The view device clip does not intersect the bounds described by the view device bitmap shape.
viewDevice_scale_too_small	The mapping to global space has been completed. The object is less than 1/72 inch. You may see a few pixels drawn, depending on the resolution of your draw view port.
viewDevice_map_too_large viewDevice_move_too_large viewDevice_scale_too_large viewDevice_rotate_too_large viewDevice_perspective_too_large viewDevice_skew_too_large	The view port mapping has moved the shape outside the bounds of the coordinate system (±32,768). This may be the result of a move, scale, rotate, perspective, or skew transformation.

Using Validation Functions

QuickDraw GX provides validation functions that check the function parameters of all allocated objects. You can validate the public functions that you use in your application or choose to validate the internal QuickDraw GX functions.

Type validation is the simplest level of validation. QuickDraw GX provides successively more complicated levels of validation when you also check structures and internal caches. The various validation modes and validation levels are described in the section "Validation Functions" beginning on page 4-6.

Controlling Validation

You can use the GXSetValidation function to control the validation of public and private functions used by your application. You control validation by using the GXSetValidation function to set validation level flags for the gxValidationLevel parameter.

```
void GXSetValidation(gxValidationLevel, level);
```

You set one flag from the modes in Table 4-8, one flag from the options in Table 4-9, and one or more flags from Table 4-10. The validation modes and levels are defined in the gxValidationLevel enumeration that appears in the section "Drawing Errors" beginning on page 4-29. The GXSetValidation function is described on page 4-34

Once you set the gxValidationLevel parameter, you can use the GXGetValidation function to return the current gxValidationLevel parameter.

The three validation mode options are validation off, public validation, and internal validation. You may choose only one of these validation options. Table 4-8 summarizes the public and internal validation mode options.

Table 4-8 Validation modes

Constant	Value	Explanation
`gxNoValidation`	0x00	Turns off QuickDraw GX validation.
`gxPublicValidation`	0x01	Performs validation whenever your application uses a public function.
`gxInternalValidation`	0x02	Performs validation whenever your application uses a public function or an internal function.

The validation mode flags allow you to selectively turn validation options on and off. You should experience reduction in performance only when validation is on. In the non-debugging version, validation is not operational. However, it is best just to turn validation off by setting the parameter of the `GXSetValidation` function to `gxNoValidation`.

If you activate either public validation or internal validation mode, then you must also specify either type validation, structure validation, or all object validation. You may choose only one option. Table 4-9 summarizes the type, structure, and object validation level options.

Table 4-9 Validation levels

Constant	Value	Explanation
`gxTypeValidation`	0x00	Validates object types of function parameters.
`gxStructureValidation`	0x10	Validates object structures, caches and function parameters.
`gxAllObjectValidation`	0x20	Validates object types, structures, and all internal caches built for all objects.

Type Validation

You can select the `gxTypeValidation` level to check the type passed to all objects. The type validation errors are listed in the section "Debugging Version" in the chapter "Errors, Warnings, and Notices."

The simplest and most commonly used `gxValidationLevel` parameter value combination is of the `gxPublicValidation` and `gxTypeValidation` options:

```
GXSetValidation(gxPublicValidation | gxTypeValidation);
```

This combination of options causes QuickDraw GX to verify that the objects used by all public functions your application calls are the correct type. For example, if you call the `GXDrawShape` function and pass it a style, the `GXSetValidation` function posts a `shape_wrong_type` error.

If you want to check the type of all objects that your application passes to both public and internal functions, you can use the `gxInternalValidation` option plus the `gxTypeValidation` option for the `gxValidationLevel` parameter:

```
GXSetValidation(gxInternalValidation | gxTypeValidation);
```

This is useful only for detecting GX internal errors.

Structure Validation

You can set the `gxStructureValidation` validation parameter to check the type and structure for all objects. The structure validation errors are listed in the section "Debugging Version" in the chapter "Errors, Warnings, and Notices."

If you want to check the type of all objects and structure your application passes to public functions, you can use the `gxPublicValidation` and `gxStructureValidation` options for the `gxValidationLevel` parameter:

```
GXSetValidation(gxPublicValidation | gxStructureValidation);
```

If you want to check the type of all objects and the structure your application passes to public and internal functions, you can use the `gxInternalValidation` and `gxStructureValidation` options for the `gxValidationLevel` parameter:

```
GXSetValidation(gxInternalValidation | gxStructureValidation);
```

This is useful only for detecting internal GX errors.

The `gxStructureValidation` option might generate validation errors that are not part of the public interface. For example, these options may post a `shape_cache_wrong_type` error. This suggests only that the application erroneously changed the internal information that identifies a specific shape cache or an internal GX error occured. The correct shape and the correct value for a shape cache are private. The `bad_private_flags` error means that the application corrupted the flags internal to some structure. This is a private structure and QuickDraw GX provides no additional information for these posted errors. However it is useful for a developer to report the circumstances that produced these errors so that Apple Computer, Inc. can investigate them.

All Object Validation

You can use the `gxAllObjectValidation` validation level to check the type, structure, and internal caches built for all objects. In addition, it checks objects written to disk and the file structure itself to see if they are corrupt. The all object validation errors are listed in the section "Debugging Version" in the chapter "Errors, Warnings, and Notices."

If an application has an error that randomly writes to some portion of memory, the error can corrupt one object as easily as another. As a result, it is necessary to check all objects to detect this type of error. If a random write occurs in a free memory block or the value is already in the shape type, QuickDraw GX doesn't detect it. Again, this validation allows the developer to discriminate between QuickDraw GX and application problems.

If you want to check the type of all objects, the structure, and the internal caches for all objects each time public functions are called by your application, you can use the `gxPublicValidation` and `gxAllObjectValidation` options for the `gxValidationLevel` parameter:

```
GXSetValidation(gxPublicValidation | gxAllObjectValidation);
```

As an alternative to using the `gxAllObjectValidation` options for the `gxValidationLevel` parameter of the `GXSetValidation` function, you can use the `GXValidateAll` function, described in the section "Validating Objects" beginning on page 4-20. The `GXValidateAll` function is described on page 4-43.

Memory Validation

Once you pick a validation mode and a validation level, you can then also choose to include or not include memory validation options. Memory validation does not post validation errors. If QuickDraw GX detects a memory validation problem, it drops you into Macsbug or the debugging utility that is installed on your system.

Table 4-10 summarizes the memory validation options, all of which are associated with QuickDraw GX private data structures.

Table 4-10 Memory validation options

Constant	Value	Explanation
`gxNoMemoryManagerValidation`	0x0000	Turns off memory validation.
`gxApBlockValidation`	0x0100	Enables additional error checking on application blocks passed as parameters to internal memory routines.
`gxFontBlockValidation`	0x0200	Enables additional error checking on system blocks, often font caches, passed as parameters to internal memory routines.
`gxApHeapValidation`	0x0400	Checks all objects in a heap for validity each time an internal memory routine is called.
`gxFontHeapValidation`	0x0800	Checks all font objects in a heap for validity each time an internal memory routine is called.
`gxCheckApHeapValidation`	0x1000	When used with `gxInternalValidation`, checks the application heap on every internal function call. When used with `gxPublicValidation`, checks the application heap on every public function call.
`gxCheckFontHeapValidation`	0x2000	When used with `gxInternalValidation`, checks the font heap on every internal function call. When used with `gxPublicValidation`, checks the font heap on every public function call.

If you want to check the type of all objects that your application passes to public functions and also check the application heap on every public call, you can use the `gxPublicValidation` option plus the `gxTypeValidation` option plus the `gxCheckApHeapValidation` option for the `gxValidationLevel` parameter:

```
GXSetValidation(gxPublicValidation | gxTypeValidation |
                gxCheckApHeapValidation);
```

▲ **WARNING**

If the `gxApHeapValidation` or `gxFontHeapValidation` flag is enabled and the platform that it is running on locates the graphics memory below the bottom 14 megabytes of memory, then the addresses on the stack and master pointers that refer to QuickDraw GX objects will be scrambled. This is a method of finding internal errors that may lead to unexpected erroneous behavior. For example, if the application has a path type shape and one `long` parameter of the path data happens to exactly equal the address of a graphics object, then QuickDraw GX might scramble the one `long` of path data and the path may draw one point off of the screen. This is expected behavior. These functions can scramble addresses without knowing that the addresses are really points on a path. Since these two validation types produce these apparent bugs, an application cannot use the `gxApHeapValidation` and `gxFontHeapValidation` options to ensure that QuickDraw GX has no internal bugs. These validation types are useful in tracking down bugs related to QuickDraw GX memory management. ▲

For additional information about using QuickDraw GX memory, see the chapter "QuickDraw GX Memory Management."

The `GXSetValidation` function is described on page 4-34. The `GXGetValidation` function is described on page 4-35.

Validating Objects

QuickDraw GX also provides separate functions that validate the parameters passed to specific objects, their structures, and any internal caches built for specific objects.

You can use the `GXValidateAll` function to check the type, structure, and internal caches built for all objects. This is an alternative to using the `GXSetValidation` function with the `gxInternalValidation` and `gxAllObjectValidation` options selected, as described in the section "Using Validation Functions" beginning on page 4-15.

The following functions validate specific objects:

- The `GXValidateColorSet` function checks parameters for the color space, color-value array, owner count, and tag list properties for the specified color set object. The `GXValidateColorSet` function is described on page 4-38.
- The `GXValidateColorProfile` function checks the specified color profile object. The `GXValidateColorProfile` function is described on page 4-39.

- The `GXValidateGraphicsClient` function checks all properties of a specified graphics client object. The `GXValidateGraphicsClient` function is described on page 4-42.
- The `GXValidateInk` function checks parameters for the color, transfer mode, attributes, owner count, and tag list properties for a specified ink object. The `GXValidateInk` function is described on page 4-37.
- The `GXValidateShape` function checks parameters for the type, geometry, fill, style, ink, transform, attributes, owner count, and tag list properties for a specified shape object. The `GXValidateShape` function is described on page 4-36.
- The `GXValidateStyle` function checks parameters for the pen size, cap, join, dash, pattern, curve error, attributes, text face, text size, justification, font variations, platform, text attributes properties, run controls, run features array, glyph substitutions array, kerning adjustments, priority justification override, and glyph justification overrides array properties for the specified style object. The `GXValidateStyle` function is described on page 4-36.
- The `GXValidateTag` function checks the parameters for the tag type, size, contents, and owner count properties for a specified tag object. The `GXValidateTag` function is described on page 4-39.
- The `GXValidateTransform` function checks the parameters for the clip, mapping, view port list, hit-test parameters, attributes, owner count, and tag list properties for a specified transform object. The `GXValidateTransform` function is described on page 4-38.
- The `GXValidateViewDevice` function checks parameters for the clip, mapping, bitmap, attributes, and tag list properties for a specified view device object. The `GXValidateViewDevice` function is described on page 4-40.
- The `GXValidateViewPort` function checks parameters for the clip, mapping, dither, halftone, parent view port, child view port list, view device, attributes, owner count, and tag list properties for all view port objects. The `GXValidateViewPort` function is described on page 4-40.
- The `GXValidateViewGroup` function checks parameters for the clip, mapping, dither, halftone, parent view port, child view port list, view device, attributes, owner count, and tag list properties of the view port object and the clip, mapping, bitmap, attributes, and tag list properties of the view device object. The `GXValidateViewGroup` function is described on page 4-41.

Analyzing the Cause of Validation Errors

You can use the `GXGetValidationError` function to determine the function and parameter that caused the last validation error. This function works like other QuickDraw GX functions that return variable-length data. There are three steps:

1. Call the function to determine the length of data that will be returned. If no validation error is posted, a 0 is returned.
2. Allocate memory to store the data that will be returned.
3. Call the function a second time to obtain pointers to the function, parameter name, and parameter number that caused the validation error.

Listing 4-1 gives an example of using the `GXGetValidationError` function to obtain the function and parameter that caused the last validation error. The `GXGetValidationError` function is described on page 4-35.

Listing 4-1 Determining the function and parameter that caused the last validation error

```
static void DisplayErrorMessage(gxGraphicsError errorID,
long context)
{
   char buffer[255];
   void * graphicsObject;
   long argNum;

   if (GXGetValidationError(buffer, &thing, &argNum)) {
      GXValidationError(buffer, nil, nil);
      printf("gxValidationError: %ld (routine: %s) ",
         errorID, buffer);
      printf("(argument[%ld]: 0x081x)\n",argNum, graphicsObject);
   } else
      printf("gxGraphicsError: 0x%081x\n", errorID);
}
```

Distinguishing Between Application Bugs and QuickDraw GX Bugs

All QuickDraw GX functions have been extensively tested prior to shipment. However, during your application debugging process, you may find anomalous behavior that you attribute to QuickDraw GX private functions.

Validation checking allows you to distinguish between your application bugs and QuickDraw GX bugs. If QuickDraw GX posts validation errors when internal validation is set, but not when public validation is set, it is possible that you have found an error in the QuickDraw GX internal private code. Please contact Apple Developer Technical Support and provide a detailed report of the bug encountered. For more information concerning public and internal validation modes, see the section "Controlling Validation" beginning on page 4-15.

Detecting Corrupted Objects

Normally, there is no way for an application using the public interface to corrupt the content of an object. If an error occurs with structure validation and not with type validation, either the error is a QuickDraw GX error or the application has corrupted memory. The most probable method of corrupting memory is by calling the `GXGetShapeStructure` function and altering the content directly or by writing randomly into memory. For more information concerning type and structure validation levels, see the section "Controlling Validation" beginning on page 4-15.

Debugging With GraphicsBug

GraphicsBug reads and verifies only graphics objects. It does not create objects, dispose of objects, or modify objects in any manner. GraphicsBug never interferes with an application and does not cause bugs to appear or disappear.

Table 4-11 summarizes the GraphicsBug commands. This list is available online by typing "?", "help", or "HELP" when in the command line of GraphicsBug. You can copy or save the brief explanations as a text file.

Table 4-11 GraphicsBug commands and responses

Command	Response
`DA [bu(sy) di(rect) fr(ee) i(ndirect) t(emp) u(n)b(usy) u(n)l(oaded)] [<type>[type>...]]`	Display all blocks in the heap, or all that match parameters. Example: `DA bu line layout polygon`.
`DM addr[n|t(ype)]`	Display memory from `addr` for *n* bytes or as a type. Example: `DM 1b2358 t`.
`DV`	Display version.
`ER number`	Display error name that matches this number.
`F addr[number[start[end]]] [bu(sy) di(rect) fr(ee) i(ndirect) t(emp) u(n)b(usy) u(n)l(oaded)] [<type>[type>...]]`	Find references to `addr` in the heap blocks that match parameters. Example: `F 0x4456A 3 ul picture`.
`FL addr[filename]`	Display the stream produced by flattening this shape. Example: `FL 0x3321A "flat shapes"`.
`GG`	Display graphics globals
`HC`	Check the heap.
`HD [bu(sy) di(rect) fr(ee) i(ndirect) t(emp) u(n)b(usy) u(n)l(oaded)] HD [<type> [<type>...]]`	Dump the heap or the heap parts that match parameters. Example: `HD bu line layout polygon`.

continued

Table 4-11 GraphicsBug commands and responses (continued)

Command	Response
HT	Total the heap.
HX addr|<heapname>	Switch to the heap containing addr, or named <heapname>. Example: HX System.
HZ	List the known heaps.
IG	Display initialization globals.
LC (process)	List the known graphics clients.
LP	List the known processes that have a graphics client.
CG	Display other (generic, nongraphic) globals.
Q	Quit.
UF filename[page number]	Display the contents of the file by flattening it. Use page number to specify a page of a print file.
V [addr]	Validate all (no parameters) or validate specific block.
GG	Graphics globals.
WH addr	Display the block containing addr. Operators: –, +, *, /, %, ^, |, &, [, @, *,], ~, (,) Numbers: .0x$#3 “strings: “”

In addition to the GraphicsBug commands above, you can Option-double-click (hold down the Option key and double-click) on a memory address to display memory as a type, use the up / down arrow keys to set the scrolling speed, use dot ‘.’ to represent the last displayed address, and use shape as an argument to the DA, F, and HD commands to display all graphics client-owned shapes.

Analyzing a Picture Shape

The following sections demonstrate the use of GraphicsBug for the analysis of a picture containing seven shapes. The code that creates the picture and the analysis of the data stream for each flattened shape is given in the section "Analyzing the Data Streams of Flattened Shapes" in the chapter "QuickDraw GX Stream Format."

Determining the Heap Size for All Shapes in the Picture

You can use the GraphicsBug `HT` command to display the heap total in bytes for a specified graphics client heap. First run the application, then select the graphics client heap from the GraphicsBug heap menu, then apply the `HT` command. Listing 4-2 showsa sample output of the `HT` command: the GraphicsBug heap size in bytes. Note that the size of the graphics client and its heap is 86724 bytes. You can use this procedure to select the initial size of your application's graphics client heap or heaps. For additional information about specifying the size of your graphics client heap, see the section "Creating a Graphics Client and its Graphics Client Heap" in the chapter "QuickDraw GX Memory Management."

Listing 4-2 Totaling the graphics client and its heap

```
Totaling the heap at 00c07de8 (all shapes heap).
          Total Blocks                  Total of Block Sizes
Free      0000001b    #       27        0000fff8    #    65528
Direct    00000044    #       68        00001dbc    #     7612
Indirect  00000047    #       71        000031bc    #    12732
Sub Heaps 00000000    #        0        00000000    #        0
Heap Size 000000a6    #      166        000152c4    #    86724
```

Analyzing the Shapes in the Picture

You can use the GraphicsBug `HD PIC` command to display the memory locations of the seven shapes in the picture. Listing 4-3 shows the GraphicsBug output for the picture shape created by the application "all shapes." User input is shown in boldface.

The GraphicsBug command lines shown in Listing 4-3 are used as follows:

- hd pic command turns `pic` into `picture`.
- `dm 00c0886c t` command displays default picture data.
- dm 00c0a4a0 t displays the data for the picture with seven shapes. Note that there are multiple text shapes displayed because the `gxUniqueItemsShape` attribute was set.

Listing 4-3 Determining the memory locations of the shapes in the picture

```
hx "all shapes"
heap set to 00c07de8  "all shapes"
hd pic
 Start    Length  Δ   Typ Busy Mstr Ptr Temp TBsy Disk   Object
00c0886c 00000048+00   i       00c1d02c                picture
00c0a4a0 00000108+00   i       00c1d010                picture
          Total Blocks            Total of Block Sizes
Blocks    00000002  #        2    00000150    #      336
dm 00c0886c t
displaying picture gxShape from 00c0886c
  devShape        nil
  owners            1
  seed              0
  flags      isDefaultShape
  attributes gxMapTransformShape
  gxStyle      00c083b0
  gxInk        00c08460
  gxTransform  00c088b4
  tagList         nil
  cacheList       nil
  geo.flags         0
  fillType   evenOddFill
  entries    0
  references 00000000
      gxShape        (type)       gxStyle           gxInk         gxTransform
dm 00c0a4a0 t
displaying picture gxShape from 00c0a4a0
  devShape   00c0a98c
  owners            1
  seed              0
  flags             0
  attributes
/*
There are multiple text shapes because the gxUniqueItemsShape attribute was
set.
*/
  gxStyle      00c083b0
  gxInk        00c08460
  gxTransform  00c088b4
  tagList         nil
  cacheList       nil
```

```
geo.flags         0
fillType   evenOddFill
entries    12
references 00c08d1c
    gxShape   (type)      gxStyle        gxInk      gxTransform
  00c08dd0  (line)       00000000       00000000       00000000
  00c0949c  (rectangle)  00000000       00000000       00000000
  00c099e4  (curve)      00000000       00000000       00000000
  00c09bd0  (path)       00000000       00000000       00000000
  00c0a220  (text)       00000000       00000000       00000000
  00c0a220  (text)       00c0a268       00c08e1c       00c0997c
  00c0a220  (text)       00c0a268       00c09b98       00c0a350
  00c0a220  (text)       00c0a268       00c0a640       00c0bc30
  00c0a220  (text)       00c0a268       00c0a678       00c0a6b0
  00c0a220  (text)       00c0a268       00c0a750       00c0a788
  00c0a828  (polygon)    00000000       00000000       00000000
  00c0bc94  (bitmap)     00000000       00000000       00000000
```

Analyzing the Rectangle in the Picture

You can use the `dm` command or Option-double-click command on the memory location of one of the seven shapes from Listing 4-3 to display information about the shape. Listing 4-4 shows the GraphicsBug output for the rectangle shape. The command line is shown in boldface.

Listing 4-4 Analyzing the rectangle shape in the picture

```
dm 00c0949c t
Displaying rectangle gxShape from 00c0949c
  devShape        nil
  owners            1
  seed              0
  flags             0
  attributes no attributes
  gxStyle      00c0984c
  gxInk        00c098fc
  gxTransform  00c0961c
  tagList          nil
  cacheList        nil
  geo.flags          0
  fillType   closedFrameFill
{   150.0000,    25.0000}  {   200.0000,    75.0000}
```

Analyzing the Ink in the Rectangle

You can select a memory location of one of the objects in the rectangle from Listing 4-4 and use the `dm` command or GraphicsBug Option-double click command to display information about the object. Listing 4-5 shows the GraphicsBug output for the ink in the rectangle shape. The command line is shown in boldface.

Listing 4-5 Analyzing the ink in the rectangle shape

```
dm 00c098fc t
Displaying gxInk from 00c098fc
  devInk       00c094e8
  privateFlags        0
  attributes          0
  owners              1
  seed                0
  tagList           nil
  space        gxRGBSpace
  profile           nil
  value(s)       1.0000 (ffff)  0.0000 0x0000  0.0000 0x0000
  mode         gxCopyMode
```

QuickDraw GX Debugging Reference

This section describes the data structures and routines that are specific to the QuickDraw GX debugging environment.

The “Constants and Data Types” section shows the enumerations and structures for drawing errors and GraphicsBug parameters. A cross-reference is provided to the enumerated validation levels.

Constants and Data Types

This section describes the constants and data structures that you use to provide information to debugging functions.

Drawing Errors

QuickDraw GX posts drawing errors when you use the `GXGetShapeDrawError` function after an unsuccessful drawing operation. The `gxDrawError` enumeration defines the posted drawing errors.

```
enum gxDrawErrors {
   no_draw_error,

   /* gxShape type errors */
   shape_emptyType,
   shape_inverse_fullType,
   rectangle_zero_width,
   rectangle_zero_height,
   polygon_empty,
   path_empty,
   bitmap_zero_width,
   bitmap_zero_height,
   text_empty,
   glyph_empty,
   layout_empty,
   picture_empty,

   /* general gxShape errors */
   shape_no_fill,
   shape_no_enclosed_area,
   shape_no_enclosed_pixels,
   shape_very_small,
   shape_very_large,
   shape_contours_cancel,

   /* gxStyle errors */
   pen_too_small,
   text_size_too_small,
   dash_empty,
   start_cap_empty,
   pattern_empty,
   textFace_empty,
   shape_primitive_empty,
   shape_primitive_very_small,
```

```
/* gxInk errors */
transfer_equals_noMode,
transfer_matrix_ignores_source,
transfer_matrix_ignores_device,
transfer_source_reject,
transfer_mode_ineffective,
colorSet_no_entries,
bitmap_colorSet_one_entry,

/* gxTransform errors */
transform_scale_too_small,
transform_map_too_large,
transform_move_too_large,
transform_scale_too_large,
transform_rotate_too_large,
transform_perspective_too_large,
transform_skew_too_large,
transform_clip_no_intersection,
transform_clip_empty,
transform_no_viewPorts,

/* gxViewPort errors */
viewPort_disposed,
viewPort_clip_empty,
viewPort_clip_no_intersection,
viewPort_scale_too_small,
viewPort_map_too_large,
viewPort_move_too_large,
viewPort_scale_too_large,
viewPort_rotate_too_large,
viewPort_perspective_too_large,
viewPort_skew_too_large,
viewPort_viewGroup_offscreen,

/* gxViewDevice errors */
viewDevice_clip_no_intersection,
viewDevice_scale_too_small,
viewDevice_map_too_large,
viewDevice_move_too_large,
viewDevice_scale_too_large,
viewDevice_rotate_too_large,
```

```
   viewDevice_perspective_too_large,
   viewDevice_skew_too_large
};

typedef long gxDrawError;
```

Table 4-2 through Table 4-7 list the drawing errors and give a description of each error.

Validation Levels

The `GXSetValidation` function uses the `gxValidationLevel` enumeration to turn off or to control the QuickDraw GX validation.

```
typedef long gxValidationLevel;

enum gxValidationLevels {
/*
These levels tell how to validate routines.  Choose one.
*/
   gxNoValidation              = 0x00,
   gxPublicValidation          = 0x01,
   gxInternalValidation        = 0x02,
/*
These levels tell how to validate types.  Choose one.
*/
   gxTypeValidation            = 0x00,
   gxStructureValidation       = 0x10,
   gxAllObjectValidation       = 0x20,
/*
These levels tell how to validate memory manager blocks.  Choose
any combination.
*/
   gxNoMemoryManagerValidation = 0x0000,
   gxApBlockValidation         = 0x0100,
   gxFontBlockValidation       = 0x0200
   gxApHeapValidation          = 0x0400,
   gxFontHeapValidation        = 0x0800,
   gxCheckApHeapValidation     = 0x1000,
   gxCheckFontHeapValidation   = 0x2000
} ;
```

Field descriptions

`gxNoValidation`
: If set, QuickDraw GX performs no validation checking.

`gxPublicValidation`
: If set, QuickDraw GX checks parameters to public routines.

`gxInternalValidation`
: If set, QuickDraw GX checks parameters to internal routines.

`gxTypeValidation`
: If set, QuickDraw GX checks types of objects.

`gxStructureValidation`
: If set, QuickDraw GX checks fields of private structures.

`gxAllObjectValidation`
: If set, QuickDraw GX checks every object for each public routine called.

`gxNoMemoryManagerValidation`
: If set, QuickDraw GX does not check Memory Management calls.

`gxApBlockValidation`
: If set, QuickDraw GX checks the relevant block structures before each Memory Manager call.

`gxFontBlockValidation`
: If set, QuickDraw GX also checks the system heap block structures..

`gxApHeapValidation`
: If set, QuickDraw GX also checks all application heap blocks every time the heap changes.

`gxFontHeapValidation`
: If set, QuickDraw GX also checks all system heap blocks every time the heap changes..

`gxCheckApHeapValidation`
: If set, QuickDraw GX also checks all application heap blocks for each public or internal routine called.

`gxCheckFontHeapValidation`
: If set, QuickDraw GX also checks the system heap blocks for each public or internal routine called.

For information on how to use QuickDraw GX validation, see the section “Using Validation Functions” beginning on page 4-15. The `GXSetValidation` function is described on page 4-34.

Functions

The functions described in this section allow you to detect drawing errors, perform validation, and install debugging utility functions.

Obtaining Drawing Errors

This section describes the function that allows you to obtain a single error message that describes why a shape did not draw correctly.

GXGetShapeDrawError

You can use the `GXGetShapeDrawError` function to determine why a shape failed to draw.

```
gxDrawError GXGetShapeDrawError(gxShape source);
```

`source` A reference to the shape that didn't draw.

function result An error result code indicating why a shape didn't draw.

DESCRIPTION

The `GXGetShapeDrawError` function returns a single error code that indicates why a shape didn't draw. The error returneddepends on the step in the drawing process in which the drawing error occurred. QuickDraw GX returns the first drawing error it detects in the drawing process. A drawing error that may occur later in the drawing process is not returned until all prior drawing errors detected are resolved.

If you run your application and it does not draw what you expect, you can add the `GXGetShapeDrawError` function to the end of your application code and rerun your application. QuickDraw GX returns a single error from the `gxDrawErrors` enumeration that may assist in determining the drawing problem. If a drawing error is not detected, QuickDraw GX returns a `gxNoDrawError` error.

SEE ALSO

The use of the `GXGetShapeDrawError` is discussed in the section "Analyzing Drawing Problems" beginning on page 4-8.

The `gxDrawError` enumeration is described in the section "Drawing Errors" beginning on page 4-29.

Table 4-2 through Table 4-7 provide a description of each drawing error.

Table 4-1 gives the object processing sequence that determines which drawing error is posted.

Setting and Getting Validation Options and Errors

This section describes the functions that control QuickDraw GX validation. QuickDraw GX validation checks public and internal function parameters to ensure that they are valid. You can use validation functions and flag options to check types, structures, all objects, memory, and specific objects.

When validation error checking is on, QuickDraw GX may post the validation errors listed in the section "Debugging Version" in the chapter "Errors, Warnings, and Notices."

GXSetValidation

You can use the `GXSetValidation` function to control the type and level of validation checking.

```
void GXSetValidation(gxValidationLevel);
```

`gxValidationLevel`
The validation flags.

DESCRIPTION

The `GXSetValidation` function allows you to set the validation mode, as well as the validation levels, for type, structure, all object, and memory block validation options. You may pick one mode, one level, and any combination of memory options. The options are defined by the `gxValidationLevel` enumeration.

The `GXSetValidation` function turns validation on when you select any flags other than 0x00. If you set the `gxValidationLevel` flag to `gxNoAttributes`, validation is off.

This function has no effect in the non-debugging version of QuickDraw GX.

As an alternative to the use of the `GXSetValidation` function with the internal and all object validation flags set, you can use the `GXValidateAll` function.

SEE ALSO

To get the current `gxValidationLevel` parameter, use the `GXGetValidation` function, described on page 4-35.

The `gxValidationLevel` enumeration is described in the section "Validation Levels" beginning on page 4-31.

Table 4-8 on page 4-16 lists the public and internal validation options.

Table 4-9 on page 4-16 lists the type, structure, and all object validation options.

Table 4-10 on page 4-19 lists the memory validation options.

The `GXValidateAll` function is described on page 4-43.

GXGetValidation

You can use the `GXGetValidation` function to obtain the current validation flags that are set.

```
gxValidationLevel GXGetValidation(void);
```

function result The current flags set for validation error checking.

DESCRIPTION

The `GXGetValidation` function returns the `gxValidationLevel` parameter set by the `GXSetValidation` function.

This function always returns 0 in the non-debugging version of QuickDraw GX.

SEE ALSO

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

GXGetValidationError

You can use the `GXGetValidationError` function to determine the application function and parameter that caused the last validation error.

```
void GXGetValidationError(char *procedureName, void **argument,
                          long *argumentNumber);
```

`procedureName`
A pointer to the name of the function that produced the validation error.

`argument` A pointer to a list of the function's arguments.

`argumentNumber`
A pointer to the number of the argument that produced the validation error.

DESCRIPTION

The `GXGetValidationError` function provides the name of the function, a list of the function's parameters, and the number of the parameter that produced the last validation error. The `argumentNumber` parameter for the *n*th parameter is *n*. For example, the `argumentNumber` for the third parameter is 3. If you call the `GXGetValidationError` function and no validation errors have been posted, the function returns `nil`.

This function leaves its arguments unchanged in the non-debugging version of QuickDraw GX.

SEE ALSO

The use of the `GXGetValidationError` function is described in the section "Analyzing the Cause of Validation Errors" beginning on page 4-21.

Validating Objects

This section describes the functions that allow you to validate the function parameters of allocated QuickDraw GX objects. QuickDraw GX provides functions for specific object validation and all object validation.

When validation error checking is on, QuickDraw GX may post the validation errors listed in the section "Debugging Version" in the chapter "Errors, Warnings, and Notices."

GXValidateShape

You can use the `GXValidateShape` function to check the parameters of a shape object.

```
void GXValidateShape(gxShape target);
```

`target` A reference to a shape object to be validated.

DESCRIPTION

The `GXValidateShape` function checks parameters for the type, geometry, fill, style, ink, transform, attributes, owner count, and tag list properties for all shape objects. In addition, this function checks any internal caches built for the shape. If one or more of the parameters are not valid, a validation error is posted.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is discussed in the section "Controlling Validation" beginning on page 4-15.

GXValidateStyle

You can use the `GXValidateStyle` function to check the parameters of a style object.

```
void GXValidateStyle(gxStyle target);
```

`target` A reference to a style object to be validated.

DESCRIPTION

The `GXValidateStyle` function checks parameters for the pen size, cap, join, dash, pattern, curve error, and attributes properties for all graphics style objects. It also checks parameters for the text face, text size, justification, font variations, platform, and text attributes properties for all typographic style objects. In addition, it confirms parameters for the run controls, run features array, glyph substitutions array, kerning adjustments, priority justification override, and glyph justification overrides array typographic properties for all layout shapes objects. In addition, this function checks any internal caches built for the style. If one or more parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX. If a discrepancy is found, QuickDraw GX posts an error.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is described in the section "Controlling Validation" beginning on page 4-15.

GXValidateInk

You can use the `GXValidateInk` function to check the parameters of an ink object.

```
void GXValidateInk(gxInk target);
```

`target` A reference to an ink object to be validated.

DESCRIPTION

The `GXValidateInk` function checks parameters for the color, transfer mode, attributes, owner count, and tag list properties for all ink objects. In addition, this function checks any internal caches built for the ink. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is described in the section "Controlling Validation" beginning on page 4-15.

GXValidateTransform

You can use the `GXValidateTransform` function to check the parameters of a transform object.

```
void GXValidateTransform(gxTransform target);
```

`target` A reference to a transform object to be validated.

DESCRIPTION

The `GXValidateTransform` function checks the parameters for the clip, mapping, view port list, hit-test parameters, attributes, owner count, and tag list properties for all transform objects. In addition, this function checks any internal caches built for the transform. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is described in the section "Controlling Validation" beginning on page 4-15.

GXValidateColorSet

You can use the `GXValidateColorSet` function to check the parameters of a color set object.

```
void GXValidateColorSet(gxColorSet target);
```

`target` A reference to a color set object to be validated.

DESCRIPTION

The `GXValidateColorSet` function checks parameters for the color space, color-value array, owner count, and tag list properties for all color set objects. In addition, this function checks any internal caches built for the color set. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

GXValidateColorProfile

You can use the `GXValidateColorProfile` function to check the parameters of a color profile object.

```
void GXValidateColorProfile(gxColorProfile target);
```

`target` A reference to a color profile object to be validated.

DESCRIPTION

The `GXValidateColorProfile` function checks the content of the target color profile object. In addition, this function checks any internal caches built for the color profile. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

GXValidateTag

You can use the `GXValidateTag` function to check the parameters of a tag object.

```
void GXValidateTag(gxTag target);
```

`target` A reference to a tag object to be validated.

DESCRIPTION

The `GXValidateTag` function checks the parameters for the tag type, size, contents, and owner count properties for all tag objects. In addition, this function checks any internal caches built for the tag. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

GXValidateViewDevice

You can use the `GXValidateViewDevice` function to check the parameters of a view device object.

```
void GXValidateViewDevice(gxViewDevice target);
```

`target` A reference to a view device object to be validated.

DESCRIPTION

The `GXValidateViewDevice` function checks parameters for the clip, mapping, bitmap, attributes, and tag list properties for all view device objects. In addition, this function checks any internal caches built for the view device. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

GXValidateViewPort

You can use the `GXValidateViewPort` function to check the parameters of a view port object.

```
void GXValidateViewPort(gxViewPort target);
```

`target` A reference to a view port object to be validated.

DESCRIPTION

The `GXValidateViewPort` function checks parameters for the clip, mapping, dither, halftone, parent view port, child view port list, view device, attributes, owner count, and tag list properties for all view port objects. In addition, this function checks any internal caches built for the view port. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is described in the section "Controlling Validation" beginning on page 4-15.

GXValidateViewGroup

You can use the `GXValidateViewGroup` function to check the parameters of a view group object.

```
void GXValidateViewGroup(gxViewGroup target);
```

`target` A reference to a view group object to be validated.

DESCRIPTION

The `GXValidateViewGroup` function checks parameters for the clip, mapping, dither, halftone, parent view port, child view port list, view device, attributes, owner count, and tag list properties of the view port object and the clip, mapping, bitmap, attributes, and tag list properties of the view device object. In addition, this function checks any internal caches built for the view group. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section "Validation Functions" beginning on page 4-6. The use of validation functions is described in the section "Controlling Validation" beginning on page 4-15.

GXValidateGraphicsClient

You can use the `GXValidateGraphicsClient` function to check the parameters of a graphics client object.

```
void GXValidateGraphicsClient(gxGraphicsClient target);
```

`target` A reference to a graphics client object to be validated.

DESCRIPTION

The `GXValidateGraphicsClient` checks all parameters for all properties of a graphics client object. In addition, this function checks any internal caches built for the graphics client. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

GXValidateAll

You can use the `GXValidateAll` function to validate all objects that are allocated.

```
void GXValidateAll(void);
```

DESCRIPTION

The `GXValidateAll` function allows you to validate the parameters of all objects that are allocated in the QuickDraw GX heap. It also checks additional structures in the backing store. In addition, this function checks any internal caches built for the objects. If one or more of the parameters are not valid, QuickDraw GX posts a validation error.

This function is not operational in the non-debugging version of QuickDraw GX.

An alternative method of validating all of the objects in the heap is to use the `GXSetValidation` function with the `gxValidationLevel` parameter set to the `gxPublicValidation` plus `gxAllObjectValidation` options.

SEE ALSO

QuickDraw GX validation is introduced in the section “Validation Functions” beginning on page 4-6. The use of validation functions is described in the section “Controlling Validation” beginning on page 4-15.

The `GXSetValidation` function is described on page 4-34.

Summary of QuickDraw GX Debugging

Constants and Data Types

Drawing Errors

```
typedef long gxDrawError;

enum gxDrawErrors {
   no_draw_error,

   /* gxShape type errors */
   shape_emptyType,
   shape_inverse_fullType,
   rectangle_zero_width,
   rectangle_zero_height,
   polygon_empty,
   path_empty,
   bitmap_zero_width,
   bitmap_zero_height,
   text_empty,
   glyph_empty,
   layout_empty,
   picture_empty,

   /* general gxShape errors */
   shape_no_fill,
   shape_no_enclosed_area,
   shape_no_enclosed_pixels,
   shape_very_small,
   shape_very_large,
   shape_contours_cancel,

   /* gxStyle errors */
   pen_too_small,
   text_size_too_small,
   dash_empty,
   start_cap_empty,
   pattern_empty,
   textFace_empty,
```

```
shape_primitive_empty,
shape_primitive_very_small,

/* gxInk errors */
transfer_equals_noMode,
transfer_matrix_ignores_source,
transfer_matrix_ignores_device,
transfer_source_reject,
transfer_mode_ineffective,
colorSet_no_entries,
bitmap_colorSet_one_entry,

/* gxTransform errors */
transform_scale_too_small,
transform_map_too_large,
transform_move_too_large,
transform_scale_too_large,
transform_rotate_too_large,
transform_perspective_too_large,
transform_skew_too_large,
transform_clip_no_intersection,
transform_clip_empty,
transform_no_viewPorts,

/* gxViewPort errors */
viewPort_disposed,
viewPort_clip_empty,
viewPort_clip_no_intersection,
viewPort_scale_too_small,
viewPort_map_too_large,
viewPort_move_too_large,
viewPort_scale_too_large,
viewPort_rotate_too_large,
viewPort_perspective_too_large,
viewPort_skew_too_large,
viewPort_viewGroup_offscreen,

/* gxViewDevice errors */
viewDevice_clip_no_intersection,
viewDevice_scale_too_small,
viewDevice_map_too_large,
viewDevice_move_too_large,
viewDevice_scale_too_large,
```

```
   viewDevice_rotate_too_large,
   viewDevice_perspective_too_large,
   viewDevice_skew_too_large
};
```

Validation Levels

```
typedef long gxValidationLevel;

enum gxValidationLevels {
/*
These levels tell how to validate routines.  Choose one.
*/
   gxNoValidation = 0x00,        /* no validation */
   gxPublicValidation = 0x01,    /* check parameters to public routines */
   gxInternalValidation = 0x02,  /* check parameters to internal routines */

/*
These levels tell how to validate types.  Choose one.
*/
   gxTypeValidation = 0x00,      /* check types of objects */
   gxStructureValidation = 0x10, /* check fields of private structures */
   gxAllObjectValidation = 0x20, /* check every object over every call */

/*
These levels tell how to validate memory manager blocks.  Choose any
combination.
*/
   gxNoMemoryManagerValidation = 0x0000,/* no memory validation */
   gxApBlockValidation = 0x0100,       /* check the relevant block
                                        structures after each Memory Manager
                                        call */
   gxFontBlockValidation = 0x0200/* check the system gxHeap as well */
   gxApHeapValidation = 0x0400,  /* check the memory manager's gxHeap after
                                 every memory call */
   gxFontHeapValidation= 0x0800, /* also check the system gxHeap */
   gxCheckApHeapValidation = 0x1000,
                                 /* check the memory manager's
                                  gxHeap if checking routine
                                  parameters */
```

```
   gxCheckFontHeapValidation = 0x2000
                                  /* check the system gxHeap as
                                   well */
} ;
```

Functions

Obtaining Drawing Errors

```
gxDrawError GXGetShapeDrawError
                            (gxShape source);
```

Setting and Getting Validation Options and Errors

```
void GXSetValidation        (gxValidationLevel);
gxValidationLevel GXGetValidation
                            (void);
void GXGetValidationError   (char *procedureName, void **argument,
                             long *argumentNumber);
```

Validating Objects

```
void GXValidateShape        (gxShape target);
void GXValidateStyle        (gxStyle target);
void GXValidateInk          (gxInk target);
void GXValidateTransform    (gxTransform target);
void GXValidateColorSet     (gxColorSet target);
void GXValidateColorProfile
                            (gxColorProfile target);
void GXValidateTag          (gxTag target);
void GXValidateViewDevice   (gxViewDevice target);
void GXValidateViewPort     (gxViewPort target);
void GXValidateViewGroup    (gxViewGroup target);
void GXValidateGraphicsClient
                            (gxGraphicsClient target);
void GXValidateAll          (void);
```