This chapter describes the aspects of QuickDraw GX memory management that your application can control. QuickDraw GX manages the memory blocks used by your application automatically. Read this chapter if you want to understand how QuickDraw GX memory works or to supplement QuickDraw GX memory management operations.

Before reading this chapter, you should be familiar with QuickDraw GX objects. For more information on objects, see *Inside Macintosh: QuickDraw GX Objects.*

For more information regarding Macintosh memory, see *Inside Macintosh: Memory.*

This chapter starts by providing an overview of the QuickDraw GX memory management system. It then tells how to:

- create and dispose of a graphics client and its heap
- determine memory requirements for a graphics client heap

Additional memory management topics of concern to few developers are also addressed, such as how to:

- respond to low-memory conditions
- load and unload objects
- work with multiple graphics clients

This chapter also contains reference information for the constants, data types, and functions associated with QuickDraw GX memory management.



# About QuickDraw GX Memory Management

QuickDraw GX works with the Macintosh Memory Manager to manage the memory used by your application for creating and manipulating objects. QuickDraw GX memory management is automatic. Memory blocks are allocated and deallocated as your application needs them.

Nevertheless, the more memory that QuickDraw GX has available, the faster your application can run. As a result, you may be able to improve the performance of your application by using some of the QuickDraw GX memory management operations.

## Memory Heaps

QuickDraw GX applications use an **application heap** and one or more graphics client heaps. The application heap memory holds your code and data structures. This is the part of memory where you allocate variables and your application executes. You can access any data structure in the application heap. Your application manages its own structures in the application heap and makes function calls to obtain or modify the contents of the graphics client heap.

The QuickDraw GX **graphics client heap** memory holds the QuickDraw GX objects you create. The graphics client heap consists of one or more blocks of **discontiguous memory** that QuickDraw GX uses to allocate its objects, structures, and variables. QuickDraw GX memory is private so, in general, you cannot directly access the contents of a graphics client heap.

The graphics client heap and the application heap work independently. For example, QuickDraw GX can execute from the memory on an accelerator card. As a result, QuickDraw GX never moves application memory. In addition, Macintosh Memory Manager functions cannot modify QuickDraw GX objects. QuickDraw GX has its own internal memory manager and memory management functions for interacting with its objects.

## Graphics Clients and Graphics Client Heaps

At system startup, QuickDraw GX does not have any memory available in which to do work. To allocate memory, it needs a special QuickDraw GX object called a **graphics client.** This object is associated with a graphics client heap.

Because a graphics client stores bookkeeping data for its heap, including the memory starting address and the size of all of the heap's memory blocks, a graphics client can be associated with only one QuickDraw GX graphics client heap. A graphics client also stores the error, warning, and notice state.

An application heap is allocated by the Macintosh Memory Manager when an application is launched. This is the memory region used by your application for its own code and data structures. Graphics clients and their heaps are never allocated from memory blocks that have been allocated to the application heap.

QuickDraw GX provides functions to create a new graphics client and its graphics client heap. If you don't use these functions in your application, QuickDraw GX will call them for you to create a graphics client having a default memory size and location.

When you no longer need a graphics client and its heap, you dispose of them to release memory blocks. QuickDraw GX provides functions so that your application can dispose of a graphics client and its graphics client heap, but if you don't use these functions in your application, QuickDraw GX calls them for you at the appropriate time. Whenever an application requires **memory allocation,** QuickDraw GX automatically deallocates as many graphics client heap memory blocks as it needs to in order to satisfy the application's memory requirements. QuickDraw GX never deallocates graphics client heaps while they are in use. If QuickDraw GX cannot find sufficient memory blocks, it may automatically unload objects from memory to storage disk. QuickDraw GX automatically reloads these objects from storage disk to memory as it needs them.

## Additional Topics

The latter part of this chapter discusses some issues that are relevant in only a few very specific and uncommon situations: handling low-memory problems, manual loading and unloading of objects, using functions that do not require a graphics client or its heap, specifying a memory starting location for a graphics client and its heap, and working with multiple graphics clients.

Under normal circumstances, QuickDraw GX resolves low-memory conditions and handles the loading of objects as needed, performing these tasks automatically and in a way that is transparent to your application. However, you can affect some of its processing in these areas. You can also set an attribute that prevents QuickDraw GX from allocating additional memory blocks to your graphics client heap and you can manually **load** and **unload** objects. These topics are described in the section "Additional Memory Management Topics" beginning on page 2-10.



# Using Graphics Clients and Graphics Client Heaps

QuickDraw GX provides most memory management services automatically. This section describes how your application can override this automatic control to

- create a graphics client and its heap
- determine memory requirements for a graphics client heap
- dispose of a graphics client and its heap

## Creating a Graphics Client and Its Graphics Client Heap

Either QuickDraw GX or you can create a new graphics client and its heap. This section discusses how you can control these tasks.

### Implicit Creation

If your application does not explicitly create a graphics client or a graphics client heap, QuickDraw GX creates them for you when needed. QuickDraw GX calls the `GXNewGraphicsClient` and the `GXEnterGraphics` functions when the first function call is made in your application that requires a graphics client heap. Almost all QuickDraw GX functions require a graphics client heap. The few exceptions are listed in the section "Functions That Do Not Require a Graphics Client or Heap" beginning on page 2-14.

When QuickDraw GX calls the `GXNewGraphicsClient` function, it selects the starting memory location for the heap and creates a graphics client to provide the bookkeeping for the heap. When QuickDraw GX calls the `GXEnterGraphics` function, it uses the memory location and heap size stored in the graphics client to create the new heap.

QuickDraw GX looks for a resource of type `'gasz'` with an ID of 0 and uses the first long word of that resource as the number of bytes to be allocated to the graphics client heap. If your application does not provide this resource, QuickDraw GX version 1.0 uses a **default memory size** value of 600 KB. For additional information, see the description of the `GXNewGraphicsClient` routine beginning on page 2-19.

A `'gasz'` resource can only provide one graphics client heap size in a single application. QuickDraw GX uses this size for every graphics client with a `memoryLength` parameter of zero. Listing 2-1 shows how to create a type 'gasz' resource for a 512 KB graphics client heap.

**Listing 2-1** Creating a `'gasz'` resource

```
resource 'gasz' (0) {
    0x00080000        /* 512KB graphics client heap */
};
```

The `GXNewGraphicsClient` function is described on page 2-19. The `GXEnterGraphics` function is described on page 2-22.

## Explicit Creation

If you want to specify the characteristics of the graphics client heap, you can use the `GXNewGraphicsClient` function explicitly to create a graphics client.

The `GXNewGraphicsClient` function has parameters that specify the heap's starting memory location, **memory size** in bytes, and whether or not QuickDraw GX is permitted to later increase the heap's size by allocating additional memory blocks. The graphics client stores the data passed by the `GXNewGraphicsClient` function, but does not allocate memory to the heap. This requires the `GXEnterGraphics` function call.

Most applications should allow QuickDraw GX to select the memory starting location by passing `nil` for the `memoryStart` parameter. If you need to specify the memory starting location, see the section "Specifying the Starting Location of a Graphics Client" beginning on page 2-14.

If you pass zero for the `memoryLength` parameter, QuickDraw GX looks for a resource of type `'gasz'` with an ID of 0 and uses the first long word of that resource as the heap size (the number of bytes to allocate). If your application does not provide this resource, QuickDraw GX version 1.0 uses a default size of 600 KB. Alternatively, you can specify the requested heap size in bytes. To determine how many bytes to specify for your graphics client heap, see the next section. The 'gasz' resource is described in the previous section.

If you pass zero for the attribute parameter, QuickDraw GX can later add additional memory blocks to the heap when more memory is required. If the `attribute` parameter has value 1, indicating the `gxStaticHeapConstant` constant, QuickDraw GX cannot add more memory blocks to the graphics client heap allocated.

Once a graphics client is created, you use the `GXEnterGraphics` function to allocate memory for its heap. If you don't explicitly make the call, QuickDraw GX implicitly calls the `GXEnterGraphics` function for you when it executes the next function that requires a graphics client heap. Almost all QuickDraw GX functions require a graphics client heap. The exceptions are given in the section "Functions That Do Not Require a Graphics Client or Heap" beginning on page 2-14.

Listing 2-2 shows how to explicitly create a graphics client and allocate 10 KB of memory for its heap. Since the `attribute` parameter is 0, QuickDraw GX performs its default behavior to add **memory blocks** to the graphics client heap created, as required. For example, additional memory may be required as your application creates new objects. You should allocate your graphics client at the beginning of your application and poll for errors to ensure that the graphics client is allocated.

**Listing 2-2** Explicitly creating a graphics client and its heap

```
gxGraphicsClient  newClient;
long              graphicsHeapSizeRequested = 50K; // 50K GX heap

newClient = GXNewGraphicsClient(nil, graphicsHeapSizeRequested
                                * 1024,OL );

   // After we attempted to create the graphics client, we need to
   // determine if the call succeeded. If the call did not (as in
   // the case for all GX functions), "newClient" will be nil. If
   // it is, we alert the user to the problem, Otherwise, we will
   // attempt to allocate the GX heap.

if ( newClient ) {
   GXEnterGraphiccs();
   // Calling GXEnterGraphics allocates the memory within the GX
   // heap. The call would fail only if there is not enough
   // memory. In this case, the graphics error posted is -27999
   // (out of memory). At this point, we have not installed an
   // error handler, so we check for the error number
   // corresponding to the out-of-memory error.
```

```
if ( GXGetGraphicsError( nil ) == -27999 ) {

   // Because we cannot allocate the requested size for our GX
   // heap, we need to throw away the client we created and alert
   // the user that there is not enough memory to continue.

   GXDisposeGraphicsClient( newClient);

   >> application code to alert user and shut down app


} else {
   // Application error code to tell the user there is not enough
   // memory to create the graphics client. No error is
   // posted from GX because a graphics client does not
   // exist. The only reason you would not be able to create
   // a graphics client is if there is not enough memory.

   >> application code to alert user and shut down app
```

## Determining Memory Requirements for a Graphics Client Heap

Using the optimal heap size increases the performance of your application. If your application does not allocate sufficient memory, QuickDraw GX will need to add additional memory blocks to the initial graphics client heap. If your heap is sized too large, you are wasting memory space.

You can use the QuickDraw GX GraphicsBug utility to check the actual size of your graphics client heap to ensure that your application has allocated sufficient, but not excessive, memory. Once you determine the optimal graphics client heap size for your application, you can specify this size at the beginning of your application by using the `GXNewGraphicsClient` function.

You can use the following procedure to determine the memory requirements of your graphics client heap:

1. Start your application with the `GXNewGraphicsClient` function and specify a memory size, such as 1 MB.
2. Run your application and create a document. QuickDraw GX allocates or deallocates memory blocks to a size that it deems necessary and sufficient to accommodate the number and complexity of the objects you have created.

3. Use the Heap Total (HT) GraphicsBug command to determine the memory size that QuickDraw GX is currently using. This is the size of the graphics client heap.
4. Use the `GXNewGraphicsClient` function to specify the size of the QuickDraw GX graphics client heap to accommodate the actual memory required.

Repeat these steps varying the size of the document used in step 2.

By running your application with what you would consider to be a document of average size and then with a document of large size, you can arrive at an optimum graphics client heap size that is probably somewhere between these two heap sizes. One important consideration is to ensure that your largest objects have sufficient memory allocated for the graphics client heap that they reside in. This is because an object cannot be split into multiple memory blocks in the heap.

Because QuickDraw GX can grow the heap to accommodate the needs of your application, you don't need to allocate sufficient space for your largest document. Assuming you have not passed the `gxStaticHeapClient` attribute to `GXNewGraphicsClient`. This procedure provides only a preliminary evaluation of the memory requirements for your application.

For additional information on how to use the GraphicsBug utility, see the section "Debugging With GraphicsBug" in the chapter "QuickDraw GX Debugging," in this book.

## Disposing of a Graphics Client and Graphics Client Heap

When your application no longer needs a graphics client and its heap, you should **dispose** of them to free memory blocks. You can use the `GXExitGraphics` and `GXDisposeGraphicsClient` functions to do this.

While you are writing and debugging your application, it is a good idea to be meticulous about disposing of all graphics clients and graphics client heaps at the end of your application. As a result, you should use the `GXExitGraphics` function to dispose of the currently active graphics client heap and the `GXDisposeGraphicsClient` function to dispose of each active graphics client. If your application does not make these calls, QuickDraw GX automatically disposes of all graphics clients and heaps that belong to the exiting application. However, in this case, the graphics clients and heaps are considered aborted instead of being disposed of normally, and therefore QuickDraw GX does not report any errors that occur during the process of disposing of these graphics clients and heaps. Listing 2-3 shows how to properly dispose of a graphics client and its heap.

**Listing 2-3** Disposing of graphics clients and graphics client heaps

```
.
. /* QuickDraw GX application code */
.
GXExitGraphics(void);
GXDisposeGraphicsClient(client);
}
```

Once your application is ready to ship, be sure to remove the terminating `GXExitGraphics` and `GXDisposeGraphicsClient` function calls and rely on QuickDraw GX to automatically dispose of all of your graphics clients and their heaps for your exiting application. When your application quits, it is much faster for QuickDraw GX to throw away all of the graphics clients and their graphics client heaps, rather than to dispose of each of them sequentially. This approach is analogous to quitting an application rather than taking the extra time to close multiple application windows.

The `GXExitGraphics` function is described on page 2-23. The `GXDisposeGraphics` function is described on page 2-21.

# Additional Memory Management Topics

This section describes some additional memory management topics. Your application can supplement QuickDraw GX automatic memory management operations to

- respond to low-memory conditions
- load and unload objects
- work with graphics clients and graphics client heaps

## Low-Memory Conditions

QuickDraw GX may post memory-related errors, warnings, and notices while trying to allocate additional memory. These notifications indicate the status of QuickDraw GX memory management operations and, in some cases, provide the opportunity for your application to respond accordingly.

## Freeing Up Already Allocated Memory

When QuickDraw GX needs one or more additional memory blocks for a graphics client heap, it responds to the situation by performing one or more of the following sequential steps. If insufficient memory is freed in one step, QuickDraw GX proceeds to the next step in the sequence. When sufficient memory blocks are freed, QuickDraw GX allocates the memory blocks and processing continues. QuickDraw GX memory management steps 1 through 4 affect memory blocks that have already been allocated.

1. QuickDraw GX disposes of dead caches: A **QuickDraw GX cache** is temporary memory used by QuickDraw GX. A cache that contains out of date, and therefore invalid, information is called a **dead cache.** If it disposes of dead caches, QuickDraw GX posts a `disposed_dead_caches` notice in the debugging init when the operation is complete. This notice is posted once per graphics client. This notice is a one-time-only alert indicating that your graphics client heap is low on memory.
2. QuickDraw GX unloads objects in pictures that have the `gxDiskShape` shape attribute: All of the objects within the picture are unloaded before any other objects are unloaded. The picture object is not unloaded. The `gxDiskShape` shape attribute is described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects.*
3. QuickDraw GX disposes of live caches: A QuickDraw GX cache that contains current valid drawing information is called a **live cache.** After live caches are disposed of, they need to be rebuilt before the next time you draw the object. QuickDraw GX posts a `disposed_live_caches` notice in the debugging init when the operation is complete. This notice is only posted once per graphics client. This notice is a one-time-only alert indicating that your graphics client heap is low on memory.
4. QuickDraw GX relocates bit images: Bit images are moved in memory in order to free memory space adjacent to them. No memory error, warning, or notice is posted to notify you of this step.

## Allocating New Memory and Unloading Objects

If QuickDraw GX has not released sufficient memory after step 4, it attempts to add additional memory blocks to the graphics client heap. If sufficient memory is not available after step 5, QuickDraw GX begins to unload objects to disk storage.

5. QuickDraw GX adds additional memory blocks: QuickDraw GX adds additional memory blocks to the graphics client heap. Prior to adding memory blocks, QuickDraw GX posts an `about_to_grow_heap` warning. If the `gxStaticHeapClient` attribute is set for the graphics client heap, QuickDraw GX does not perform this step.
6. QuickDraw GX unloads objects: Prior to unloading objects, QuickDraw GX posts an `about_to_unload_objects` warning. First, shapes with the `gxDiskShape` shape attribute are unloaded. Then, objects without either the `gxDiskShape` or the `gxMemoryShape` attributes are unloaded. Finally, shapes with the `gxMemoryShape` attribute are unloaded. Unlike disposing of caches, unloading occurs without information loss, but it does take time and disk space. For additional information about object loading and unloading, see the section “Loading and Unloading Objects” beginning on page 2-12. If an object cannot be unloaded, QuickDraw GX posts a `could_not_create_backing_store` error or the appropriate system error.

When your application has received the `about_to_grow_heap` warning or the `could_not_create_backing_store` error, you can decide to free up some memory before GX does. However, you must be very careful if you decide to dispose of a GX object. You cannot dispose of anything that is currently in use. The only way to determine if something is in use would be by carefully tracking the GX objects used within your application. Most likely, you would only want to dispose of off-screen worlds used by your application and let GX free up memory by releasing other blocks. GX knows what is and is not busy.

If steps 1 through 6 fail to release sufficient memory to accommodate the allocation of the required additional blocks of memory, QuickDraw GX posts an `out_of_memory` error.

### Functions That Create Additional Memory Demands

Individual QuickDraw GX functions have different memory-allocation consequences:

- Many QuickDraw GX functions explicitly allocate memory. For example, the `GXNewShape`, `GXCopyToShape`, and `GetShapeParts` functions allocate memory. An `out_of_memory` error may be posted when a memory allocation fails.
- Most QuickDraw GX functions can implicitly allocate memory to load a required object. For example, the `GXGetShapeAttributes` function may need to load a shape into memory to retrieve its attributes. QuickDraw GX loads objects automatically and does not post an error, warning, or notice. The exception is when QuickDraw GX posts an `out_of_memory` error when a memory allocation fails or a disk error occurs.
- Some functions do not allocate memory explicitly or implicitly. These functions might require a graphics client heap and do not post an `out_of_memory` error. These include math routines, error routines, and the `GXClone`*Object*, `GXDispose`*Object*, `GXUnload`*Object*, `GXValidate`*Object* function sets and all of the functions listed in the second part of Table 2-1 on page 2-14.

The GraphicsBug utility is useful in debugging memory problems. This utility is described in the chapter “QuickDraw GX Debugging” in this book.

QuickDraw GX errors, warnings, and notices are described in the chapter “Errors, Warnings, and Notices” in this book.

## Loading and Unloading Objects

If your application needs more memory during execution, QuickDraw GX automatically unloads objects to disk storage to free memory. QuickDraw GX automatically reloads previously unloaded objects when it needs them.

QuickDraw GX only begins to unload objects after it has failed to free sufficient memory by disposing of dead caches, unloading picture shape objects, disposing of live caches, relocating bit images, and adding additional memory blocks to the graphics client heap. Unless you choose to control loading and unloading of objects to memory, QuickDraw GX performs these tasks for you automatically. Your application never needs to load or unload an object.

The order in which QuickDraw GX automatically loads and unloads objects depends upon the objects' shape attributes. QuickDraw GX first unloads shape objects with the `gxDiskShape` attribute. QuickDraw GX then unloads shapes without special attributes, style, ink, transform, color set, color profile, and tag objects. Finally, after all other objects are unloaded, QuickDraw GX unloads objects with the `gxMemoryShape` attribute.

You can use the `GXSetShapeAttribute` function to set or clear an object's shape attribute and the `GXGetShapeAttribute` function to determine which attributes of a shape object are set. Shape attributes are described in the chapter "Shape Objects" in *Inside Macintosh: QuickDraw GX Objects*.

Objects are unloaded to a temporary file created on the startup disk in the invisible temporary items folder. When an object is unloaded, a 4-byte stub remains in memory to describe the location of each object on disk so that it can be reloaded when required. Sufficient disk storage space must be available to accommodate all of the objects that are unloaded or a file system error is posted.

You can supplement QuickDraw GX automatic loading and unloading operations by using function calls. These functions may be useful in increasing application performance. For example, a multimedia application with time-critical processing may need to control specific objects to ensure that they are resident in memory when they are to be displayed and removed from memory when they are no longer required.

You can use the `GXUnloadShape` function to move a shape from memory to disk storage and the `GXLoadShape` function to move a shape from disk storage to memory. QuickDraw GX provides loading and unloading functions for shape, style, ink, transform, color set, color profile, and tag objects.

When you unload an object, QuickDraw GX always first disposes of all of the live and dead caches for the object.

A recommended approach is to initially write your application without the use of object loading and unloading functions. Then, experiment with loading and unloading functions to improve performance.

The QuickDraw GX loading and unloading functions are described in the section "Loading and Unloading Objects" beginning on page 2-26.

When objects are loaded and unloaded by QuickDraw GX is discussed in the section "Low-Memory Conditions" beginning on page 2-10.

## Functions That Do Not Require a Graphics Client or Heap

Almost all QuickDraw GX functions require both a graphics client and a graphics client heap to execute. Table 2-1 lists the functions that either do not require a graphics client or require a graphics client but not its heap to execute.

**Table 2-1** QuickDraw GX functions that do not require a graphics client or heap

| Memory requirements | Function |
|---|---|
| graphics client not required | `GXValidateGraphicsClient`<br>`GXGetUserGraphicsDebug`<br>`GXSetUserGraphicsDebug`<br>`GXNewGraphicsClient`<br>`GXGetGraphicsClient`<br>`GXSetGraphicsClient`<br>`GXDisposeGraphicsClient`<br>`GXGetGraphicsClients`<br>`GXGetConvertQDFont`<br>`GXSetConvertQDFont` |
| graphics client required;<br>graphics client heap not required | All of the functions described in the chapter “Errors, Warnings, and Notices” (excluding the application-defined functions)<br><br>All of the functions described in the chapter “QuickDraw GX Mathematics”<br><br>`GXSetValidation`<br>`GXGetValidation`<br>`GXSetValidationError`<br>`GXGetGraphicsPollingHandler`<br>`GXSetGraphicsPollingHandler`<br>`GXEnterGraphics` |

## Specifying the Starting Location of a Graphics Client

If you use the `GXNewGraphicsClient` function to specify the starting location of a new graphics client, you must also specify the requested size of the graphics client heap. In this case, the size in bytes of the graphics client heap requested is used for a contiguous block of memory for both the graphics client and heap. In all other cases, the graphics client heap is allocated as a discontiguous memory block and the entire memory allocation requested by specifying a `memoryLength` parameter for the `GXNewGraphicsClient` function is assigned to the new graphics client heap.

Use the `GXNewGraphicsClient` function if you need to create a graphics client without allocating any memory. This allows you to draw at interrupt time. For example, you may want to report `out_of_memory` errors in a dialog box.

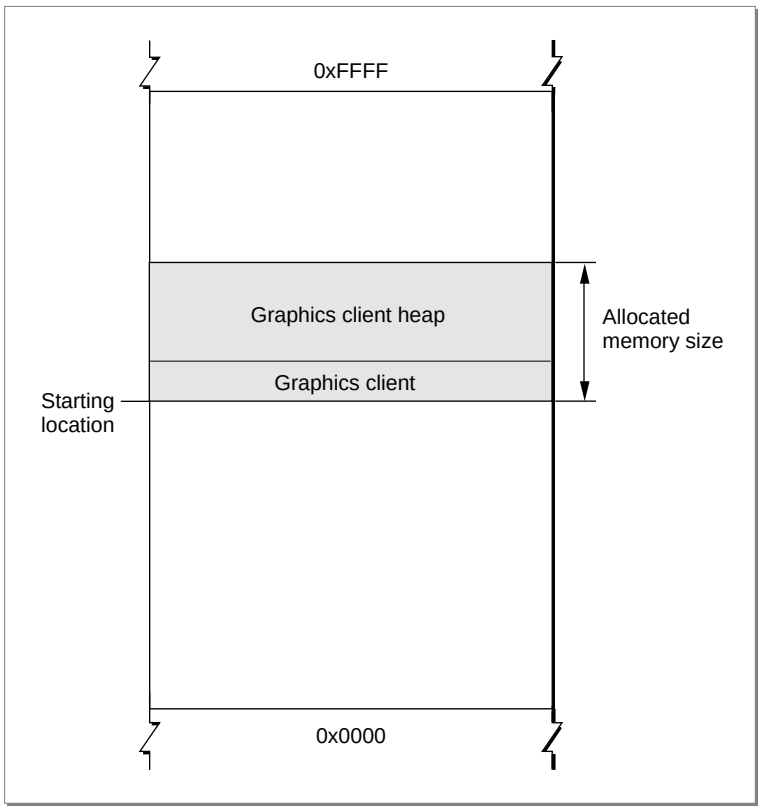
Listing 2-4 demonstrates how to specify a memory size and a memory starting location for a graphics client and its heap.

**Listing 2-4** Specifying the starting location and size for a graphics client and its heap

```
gxGraphicsClient     newClient;
char                 memoryBuffer[10000]

newClient = GXNewGraphicsClient(&memoryBuffer[0],
                                sizeof (memoryBuffer),
                                gxNoAttributes);
   // After we attempted to create the graphics client, we need
   // to determine if the call succeeded.If it did not ( as in the
   // case for all GX functions), “newClient” will be nil. If it
   // is, we alert the user to the problem. Otherwise, we attempt
   // to allocate the GX heap.

if (newClient)

   GXEnterGraphics();
   // Calling GXEnterGraphics allocates the memory within the GX
   // heap. The only reason the call would not succeed is if
   // there is not enough memory. In this case, the graphics
   // error which is posted is -27999 (out of memory). At this
   // point, we have not installed an error handler, so we check
   // for the error number corresponding to the out-of-memory
   // error.

if ( GXGetGraphicsError( nil ) == -27999 ) {
   // Because we canot allocate the requested size for our GX
   // heap, we need to throw away the client we created and alert
   // the user that there is not enough memory to continue..
   //
   GXDisposeGraphicsClient(newClient);
      >>application code to alert user and shut down app
} else {
      >>application code to alert user and shut down app
}}
```

The `myClient` variable holds the new graphics client. You can use this variable to access the graphics client any time you need it. The combined size of the graphics client and the graphics client heap is 10 KB and its starting location in memory is at the starting location of the buffer. Since the memory starting location is specified, the new graphics client and its heap use contiguous memory, as shown in Figure 2-1.

**Figure 2-1** Creating a graphics client by specifying the memory starting location




## Working With Multiple Graphics Clients

The exceptional QuickDraw GX application may need multiple graphics clients to provide special features. For example, an application may want to create multiple graphics clients to provide a QuickDraw GX environment with

- segmented memory to allow some sets of objects to have more memory than others
- different error states so that one state would have an error condition and the other would have a normal condition.

Another example is a QuickDraw GX application that needs to display a dialog box to convey status information while continuing to perform other tasks. By using separate graphics clients for the dialog box and the other task-oriented part of the application, you can guarantee that QuickDraw GX will not affect the memory being used for the dialog box.

Two disadvantages of having multiple graphics clients are that

- objects cannot be shared between graphics clients
- memory may become fragmented as the memory size grows

Without object sharing, if an object is to be used by more than one graphics client, the object must be duplicated and this requires additional memory overhead. Fragmented memory results from QuickDraw GX objects being initially allocated to a large block of memory and subsequent addition of multiple discontiguous memory blocks.

If you are going to have multiple graphics clients, you must explicitly create them using the `GXNewGraphicsClient` and `GXEnterGraphics` functions. This assures that a reference is returned for each new graphics client. If you allow QuickDraw GX to implicitly create a graphics client, QuickDraw GX has no way of returning a reference.

The `GXGetGraphicsClient`, `GXGetGraphicsClients`, and `GXSetGraphicsClient` functions allow you to work with the graphics clients that you create.

You can use the `GXGetGraphicsClients` function to return some or all of the graphics client references that have been allocated by QuickDraw GX. The `GXGetGraphicsClients` function is described on page 2-25.

You can use the `GXSetGraphicsClient` function to change the active graphics client for your application and the `GXGetGraphicsClient` function to return the active graphics client for your application. These functions may be used prior to calling `GXEnterGraphics` and `GXExitGraphics` to specify the active graphics client. The `GXSetGraphicsClient` function is described on page 2-26 and the `GXGetGraphicsClients` function is described on page 2-25.

Creating graphics clients and graphics client heaps explicitly is described in the section “Explicit Creation” beginning on page 2-6.

# QuickDraw GX Memory Management Reference

This section describes the constants, data types, and functions related to QuickDraw GX memory management. The section “Constants and Data Types” gives the type definition of the graphics client and the graphics client attributes enumeration. The section “Functions” describes the functions used for creating and disposing of a graphics client, working with multiple graphics clients, and loading and unloading objects.

# Constants and Data Types

This section describes the constants and data types that are used for memory management.

## Graphics Client Object

QuickDraw GX provides you with access to the graphics client object through a graphics client reference:

```
typedef struct gxPrivateGraphicsClientRecord *gxGraphicsClient;
```

In this type definition, `gxGraphicsClient` is a type-checked reference, not an actual pointer to any defined structure. The contents of the graphics client object are private.

## Graphics Client Attributes

The options for the `attribute` parameter of the `GXNewGraphicsClient` function are defined in the `gxClientAttributes` enumeration:

```
enum gxClientAttributes {
   gxStaticHeapClient= 0x0001
};

typedef long gxClientAttribute;
```

#### Constant descriptions

`gxStaticHeapClient`
: QuickDraw GX will never add additional memory blocks to the graphics client heap.

A graphics client having a `gxClientAttributes` value of 0 may add additional memory blocks to its heap, as required. This is the standard default behavior.

For additional information, see the section “Creating a Graphics Client and Its Graphics Client Heap” beginning on page 2-5. The `GXNewGraphicsClient` function is described on page 2-19.

# Functions

This section describes the Quickdraw GX functions you can use to

- create and dispose of graphics clients
- allocate and dispose of graphics client heaps
- load and unload objects
- work with multiple graphics clients

## Creating and Disposing of a Graphics Client

This section describes the QuickDraw GX functions you can use to

- create a new graphics client
- dispose of a graphics client



## GXNewGraphicsClient

You can use the `GXNewGraphicsClient` function to create a new graphics client.

```
gxGraphicsClient GXNewGraphicsClient(void *memoryStart,
                  long memoryLength, gxClientAttribute attribute);
```

`memoryStart`
A pointer to the memory location where the new graphics client will begin.

`memoryLength`
The requested size in bytes of the QuickDraw GX graphics client heap.

`attribute` The attributes flag set for the new graphics client.

*function result* A reference to the new graphics client object.

**DESCRIPTION**

The `GXNewGraphicsClient` function creates a new graphics client and makes it the active graphics client for this application. The graphics client specifies the memory location, the size in bytes, and the attributes of its graphics client heap. When additional memory blocks are allocated to the graphics client heap, their locations and sizes are also stored in the graphics client. The `GXNewGraphicsClient` function does not allocate memory for the graphics client heap. Calling the `GXEnterGraphics` functionallocates the heap.

If you are going to make a `GXNewGraphicsClient` call, it must be the first QuickDraw GX call in your application. Otherwise, a Quickdraw GX call may implicitly create the first graphics client and any subsequent `GXNewGraphicsClient` call creates another graphics client. If you want to create multiple graphics client objects, you can call this routine several times.

The `memoryStart` parameter specifies the starting location in memory for the graphics client and its graphics client heap. If you specify `nil`, QuickDraw GX selects the location for you. This is the most common selection. Since QuickDraw GX is managing memory, it selects what it believes is the optimum location in memory for the new graphics client. However, in the rare case in which you need to specify the memory location, you can use

the `memoryStart` parameter to specify the exact location of the graphics client. If you specify the `memoryStart` parameter, you must also specify the `memoryLength` parameter.

The `memoryLength` parameter specifies the size of the heap in bytes. If you pass 0 and there is no '`gasz`' resource, QuickDraw GX version 1.0 creates a graphics client with a default heap size of 600 KB. If there is a `'gasz'` resource, QuickDraw GX uses its size value instead.

The `attributes` parameter is a flag from the `gxClientAttributes` enumeration that defines whether QuickDraw GX will or will not add additional memory blocks to the newly defined, but not allocated, graphics client heap. A flag of default value 0 indicates that QuickDraw GX may add memory blocks to the graphics client heap, as required. A flag of value 1 is the `gxStaticHeapClient` constant and indicates that QuickDraw GX will never add memory blocks to the initially allocated graphics client heap.

If QuickDraw GX is unable to create a graphics client, there probably is not sufficient memory. As a result, the function returns `nil`. Note that QuickDraw GX does not post an error since there is no graphics client to post the error to.

#### SPECIAL CONSIDERATIONS

If no error results, the `GXNewGraphicsClient` function creates a graphics client object; you are responsible for disposing of that object when you no longer need it.

#### SEE ALSO

The use of the `GXNewGraphicsClient` function to create a new graphics client is described in the section “Creating a Graphics Client and Its Graphics Client Heap” beginning on page 2-5.

To determine the correct size of the memory for your graphics client, see the section “Determining Memory Requirements for a Graphics Client Heap” beginning on page 2-8.

The `gxClientAttribute` enumeration is described in the section “Graphics Client Attributes” beginning on page 2-18.

QuickDraw GX functions that do not require a graphics client or a graphics client heap are described in the section “Functions That Do Not Require a Graphics Client or Heap” beginning on page 2-14.

If you need to specify the memory starting location of the graphics client and its graphics client heap, see the section “Specifying the Starting Location of a Graphics Client” beginning on page 2-14.

## GXDisposeGraphicsClient

You can use the `GXDisposeGraphicsClient` function to dispose of a specific graphics client.

```
void GXDisposeGraphicsClient(gxGraphicsClient client);
```

`client` A reference to the graphics client to be disposed of.



**DESCRIPTION**

The `GXDisposeGraphicsClient` function is the last QuickDraw GX call that an application being debugged should make. It disposes of all the data structures associated with the passed graphics client, including its heap. If the application does not make this call, QuickDraw GX automatically disposes of all graphics clients that belong to the exiting application. However, in this case the graphics clients are considered aborted instead of being disposed of normally, and therefore QuickDraw GX does not report any errors that occur during the process of disposing of these graphics clients.

**SPECIAL CONSIDERATIONS**

If your `GXNewGraphicsClient` call failed to create a graphics client and returned `nil`, this function accepts `nil` as a valid graphics client and disposes of the referenced graphics client.

When your application is ready to ship, you should remove the terminating `GXDisposeGraphicsClient` function and rely on QuickDraw GX to automatically dispose of your graphics clients.

**SEE ALSO**

The role of the `GXDisposeGraphicsClient` function in disposing of a graphics client is described in the section “Disposing of a Graphics Client and Graphics Client Heap” beginning on page 2-9.

The `GXNewGraphicsClient` function is used to create a new graphics client from memory and is described on page 2-19.

## Allocating and Disposing of a Graphics Client Heap

This section describes the QuickDraw GX functions you can use to

- allocate a graphics client heap
- obtain a list of all of the allocated graphics clients
- dispose of a graphics client heap

## GXEnterGraphics

You can use the `GXEnterGraphics` function to allocate memory for a QuickDraw GX graphics client heap.

```
void GXEnterGraphics(void);
```

#### DESCRIPTION

The `GXEnterGraphics` function allocates memory for a graphics client heap and initializes the default data structures. QuickDraw GX obtains the memory starting location, memory length, and attributes for the new graphics client heap from the active graphics client.

Normally, you never need to call `GXEnterGraphics`. You should call this function only in the specific instance that you want to isolate the QuickDraw GX call to a specific part of the application. You then usethe `GXExitGraphics` function to remove all memory used by QuickDraw GX and then use the `GXEnterGraphics` function to begin using QuickDraw GX memory again.

#### ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`

#### SEE ALSO

The use of the `GXEnterGraphics` function to allocate memory to a graphics client is described in the section “Creating a Graphics Client and Its Graphics Client Heap” beginning on page 2-5.

The `GXExitGraphics` function deallocates memory for the QuickDraw GX graphics client heap and removes the default data structures. This function is described in the next section.

QuickDraw GX functions that do not require a graphics client or a graphics client heap are described in the section “Functions That Do Not Require a Graphics Client or Heap” beginning on page 2-14.

## GXExitGraphics

You can use the `GXExitGraphics` function to dispose of the default structures and the active QuickDraw GX graphics client heap.

```
void GXExitGraphics(void);
```



#### DESCRIPTION

The `GXExitGraphics` function disposes of all of the default data structures that you have created in your QuickDraw GX application and disposes of the active graphics client heap. If a notice handler routine has been installed, it is called to report any objects allocated by the application that have not been disposed of.

Normally, you never need to call the `GXExitGraphics` function if you use the `GXDisposeGraphicsClient` function.

#### SPECIAL CONSIDERATIONS

In the debugging version of QuickDraw GX, you can call the `GXExitGraphics` function if you want to confirm that all QuickDraw GX objects that you allocated have been disposed of.

When your application is ready to ship, you should remove the terminating `GXExitGraphics` function and rely on QuickDraw GX to automatically dispose of your graphics client heaps.

#### ERRORS, WARNINGS, AND NOTICES

**Notices (debugging only)**

```
shape_not_disposed
font_not_disposed
style_not_disposed
ink_not_disposed
transform_not_disposed
colorSet_not_disposed
colorProfile_not_disposed
```

#### SEE ALSO

The role of the `GXExitGraphicsClient` function in disposing of a graphics client heap is described in the section “Disposing of a Graphics Client and Graphics Client Heap” beginning on page 2-9.

The `GXDisposeGraphicsClient` function is described on page 2-21.

The `GXEnterGraphics` function allocates memory for the QuickDraw GX graphics client heap and initializes the default data structures. This function is described on page 2-22.

## Working With Multiple Graphics Clients

This section describes the QuickDraw GX functions you can use to

- return the active graphics client
- change the active graphics client

## GXGetGraphicsClient

You can use the `GXGetGraphicsClient` function to return the active graphics client to your application.

```
gxGraphicsClient GXGetGraphicsClient(void);
```

*function result* The active graphics client.

**DESCRIPTION**

The `GXGetGraphicsClient` function returns the active graphics client. Each application has its own active graphics client. The only way that the active graphics client is changed within an application is when the application calls the `GXSetGraphicsClient` function or when a new graphics client is created by the `GXNewGraphicsClient` call.

**SEE ALSO**

For additional information about graphics clients, see the section “About QuickDraw GX Memory Management” beginning on page 2-3.

Multiple graphics clients are discussed in the section“Working With Multiple Graphics Clients” beginning on page 2-16.

The `GXGetGraphicsClients` function returns all or some of the graphics clients that have been allocated by QuickDraw GX. This function is described in the next section.

## GXGetGraphicsClients

You can use the `GXGetGraphicsClients` function to list all of the graphics clients that have been allocated by QuickDraw GX.

```
long GXGetGraphicsClients(long index, long count,
                          gxGraphicsClient clients[]);
```

`index` The one-based index into the list of all graphics clients that indicates the first client to return.

`count` The number of graphics clients to be returned.

`clients` An array of graphics client references. On return, the array contains references to the allocated graphics clients.

*function result* The number of graphics clients returned.

**DESCRIPTION**

The `GXGetGraphicsClients` function copies the graphics client references specified by the `index` and `count` parameters into the array. It will return the graphics clients that are owned by other applications in addition to the ones owned by the calling application. Specifying the value 1 for the `index` parameter returns the first client. Specifying the `gxSelectToEnd` constant for the `count` parameter returns all remaining graphics clients, starting with the indexed graphics client. If `nil` is passed for the `clients` parameter, no graphics clients are returned.

**SEE ALSO**

For additional information about graphics clients, see the section “About QuickDraw GX Memory Management” beginning on page 2-3.

Multiple graphics clients are discussed in the section“Working With Multiple Graphics Clients” beginning on page 2-16.

## GXSetGraphicsClient

You can use the `GXSetGraphicsClient` function to change the active graphics client for your application.

```
void GXSetGraphicsClient(gxGraphicsClient client);
```

`client` A reference to the graphics client that is to become active.

**DESCRIPTION**

The `GXSetGraphicsClient` function can be used to switch to any of the graphics clients that your application owns; it may not switch to graphics clients that other applications own.

The active graphics client determines which QuickDraw GX graphics client heap to use for subsequent QuickDraw GX calls. Note that if you create a QuickDraw GX object with one graphics client active and switch to another one, you may not make calls that use the object. This is because an object cannot be shared by graphics clients. The object must be duplicated.

**SEE ALSO**

See the section“Working With Multiple Graphics Clients” beginning on page 2-16 for more information about multiple graphics clients.

# Loading and Unloading Objects

This section describes the functions you use to load objects from disk storage to memory and to unload objects from memory to disk storage.

## GXLoadShape

You can use the `GXLoadShape` function to load a shape into memory.

```
void GXLoadShape(gxShape target);
```

`target` A reference to the shape object to be loaded into memory.

**DESCRIPTION**

The `GXLoadShape` function moves a shape object from disk storage to the active graphics client heap. When you or QuickDraw GX unload a shape object from memory to disk storage using the `GXUnloadShape` function, QuickDraw GX creates a 4-byte stub that remains in the active graphics client heap. When you use the `GXLoadShape` function to retrieve the stored object, QuickDraw GX obtains the location of the stored shape object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
shape_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the sections “Loading and Unloading Objects” beginning on page 2-12.

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

## GXUnloadShape

You can use the `GXUnloadShape` function to unload a shape from memory.

```
void GXUnloadShape(gxShape target);
```

`target` A reference to the shape object to be unloaded from memory.

**DESCRIPTION**

The `GXUnloadShape` function moves a shape object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadShape` function to unload a shape object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadShape` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored shape object.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
shape_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXLoadStyle

You can use the `GXLoadStyle` function to load a style into memory.

```
void GXLoadStyle(gxStyle target);
```

`target` A reference to the style object to be loaded into memory.

**DESCRIPTION**

The `GXLoadStyle` function moves a style object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload a style object from memory to disk storage using the `GXUnloadStyle` function, QuickDraw GX creates a 4-byte stub that remains in the graphics client heap. When you use the `GXLoadStyle` function to retrieve the stored style, QuickDraw GX obtains the location of the stored style object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
style_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadStyle

You can use the `GXUnloadStyle` function to unload a style from memory.

```
void GXUnloadStyle(gxStyle target);
```

`target` A reference to the style object to be unloaded from memory.

**DESCRIPTION**

The `GXUnloadStyle` function moves a style object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadStyle` function to unload a style object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadStyle` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored style object.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
style_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section “Loading and Unloading Objects” beginning on page 2-12.

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

## GXLoadInk

You can use the `GXLoadInk` function to load an ink into memory.

```
void GXLoadInk(gxInk target);
```

`target` A reference to the ink object to be loaded into memory.

**DESCRIPTION**

The `GXLoadInk` function moves an ink object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload an ink object from memory to disk storage using the `GXUnloadInk` function, QuickDraw GX creates a

4-byte stub that remains in the graphics client heap. When you use the `GXLoadInk` function to retrieve the stored object, QuickDraw GX obtains the location of the stored ink object from the stub.

#### ERRORS, WARNINGS, AND NOTICES

**Errors**

```
out_of_memory
ink_is_nil
```

#### SEE ALSO

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadInk

You can use the `GXUnloadInk` function to unload an ink from memory.

```
void GXUnloadInk(gxInk target);
```

`target` A reference to the ink object to be unloaded from memory.

#### DESCRIPTION

The `GXUnloadInk` function moves an ink object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadInk` function to unload an ink object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadInk` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored ink object.

#### ERRORS, WARNINGS, AND NOTICES

**Errors**

```
out_of_memory
ink_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXLoadTransform

You can use the `GXLoadTransform` function to load a transform into memory.

```
void GXLoadTransform(gxTransform target);
```

`target` A reference to the transform object to be loaded into memory.

**DESCRIPTION**

The `GXLoadTransform` function moves a transform object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload a transform object from memory to disk storage using the `GXUnloadTransform` function, QuickDraw GX creates a 4-byte stub that remains in the graphics client heap. When you use the `GXLoadTransform` function to retrieve the stored object, QuickDraw GX obtains the location of the stored transform object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
transform_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadTransform

You can use the `GXUnloadTransform` function to unload a transform from memory.

```
void GXUnloadTransform(gxTransform target);
```

`target` A reference to the transform object to be unloaded from memory.

### DESCRIPTION

The `GXUnloadTransform` function moves a transform object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadTransform` function to unload a transform object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadTransform` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored transform object.

### ERRORS, WARNINGS, AND NOTICES

#### Errors

```
out_of_memory
transform_is_nil
```

### SEE ALSO

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section “Loading and Unloading Objects” beginning on page 2-12.

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

## GXLoadColorSet

You can use the `GXLoadColorSet` function to load a color set into memory.

```
void GXLoadColorSet(gxColorSet target);
```

`target` A reference to the color set object to be loaded into memory.

**DESCRIPTION**

The `GXLoadColorSet` function moves a color set object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload a color set object from memory to disk storage using the `GXUnloadColorSet` function, QuickDraw GX creates a 4-byte stub that remains in the graphics client heap. When you use the `GXLoadColorSet` function to retrieve the stored object, QuickDraw GX obtains the location of the stored color set object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
color_set_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadColorSet

You can use the `GXUnloadColorSet` function to unload a color set from memory.

```
void GXUnloadColorSet(gxColorSet target);
```

`target` A reference to the color set object to be unloaded from memory.

**DESCRIPTION**

The `GXUnloadColorSet` function moves a color set object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadColorSet` function to unload a color set object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadColorSet` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored color set object.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
color_set_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXLoadColorProfile

You can use the `GXLoadColorProfile` function to load a color profile into memory.

```
void GXLoadColorProfile(gxColorProfile target);
```

`target` A reference to the color profile object to be loaded into memory.

**DESCRIPTION**

The `GXLoadColorProfile` function moves a color profile object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload a color profile object from memory to disk storage using the `GXUnloadColorProfile` function, QuickDraw GX creates a 4-byte stub that remains in the graphics client heap. When you use the `GXLoadColorProfile` function to retrieve the stored object, QuickDraw GX obtains the location of the stored color profile object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
color_profile_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadColorProfile

You can use the `GXUnloadColorProfile` function to unload a color profile from memory.

```
void GXUnloadColorProfile(gxColorProfile target);
```

`target` A reference to the color profile object to be unloaded from memory.

**DESCRIPTION**

The `GXUnloadColorProfile` function moves a color profile object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadColorProfile` function to unload a color profile object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadColorProfile` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored color profile object.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
color_profile_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section “Loading and Unloading Objects” beginning on page 2-12.

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

## GXLoadTag

You can use the `GXLoadTag` function to load a tag into memory.

```
void GXLoadTag(gxTag target);
```

`target` A reference to the tag object to be loaded into memory.

**DESCRIPTION**

The `GXLoadTag` function moves a tag object from disk storage to the active graphics client heap of your application. When you or QuickDraw GX unload a tag object from memory to disk storage using the `GXUnloadTag` function, QuickDraw GX creates a 4-byte stub that remains in the graphics client heap. When you use the `GXLoadTag` function to retrieve the stored object, QuickDraw GX obtains the location of the stored tag object from the stub.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
tag_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section "Loading and Unloading Objects" beginning on page 2-12.

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

## GXUnloadTag

You can use the `GXUnloadTag` function to unload a tag from memory.

```
void GXUnloadTag(gxTag target);
```

`target` A reference to the tag object to be unloaded from memory.

**DESCRIPTION**

The `GXUnloadTag` function moves a tag object from the active graphics client heap to disk storage. When you or QuickDraw GX use the `GXUnloadTag` function to unload a tag object from memory to disk storage, QuickDraw GX stores its location in a 4-byte stub in the active graphics client heap. When you use the `GXLoadTag` function to reload the object from disk storage to memory, QuickDraw GX uses the stub to find the stored tag object.

**ERRORS, WARNINGS, AND NOTICES**

**Errors**

```
out_of_memory
tag_is_nil
```

**SEE ALSO**

For additional information about loading objects from disk storage to memory and unloading objects from memory to disk storage, see the section “Loading and Unloading Objects” beginning on page 2-12.

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

# Summary of QuickDraw GX Memory Management

## Constants and Data Types

### Graphics Client Object

```
typedef struct gxPrivateGraphicsClientRecord *gxGraphicsClient;
```

### Graphics Client Attributes

```
enum gxClientAttributes {
   gxStaticHeapClient = 0x0001
};

typedef long gxClientAttribute;
```

## Functions

### Creating and Disposing of a Graphics Client

```
gxGraphicsClient GXNewGraphicsClient
                            (void *memoryStart, long memoryLength,
                             gxClientAttribute attribute);
void GXDisposeGraphicsClient(gxGraphicsClient client);
```

### Allocating and Disposing of a Graphics Client Heap

```
void GXEnterGraphics        (void);
void GXExitGraphics         (void);
```

### Working With Multiple Graphics Clients

```
gxGraphicsClient GXGetGraphicsClient
                            (void);
long GXGetGraphicsClients   (long index, long count,
                             gxGraphicsClient clients[]);
void GXSetGraphicsClient    (gxGraphicsClient client);
```

### Loading and Unloading Objects

```
void GXLoadShape            (gxShape target);
void GXUnloadShape          (gxShape target);
void GXLoadStyle            (gxStyle target);
void GXUnloadStyle          (gxStyle target);
void GXLoadInk              (gxInk target);
void GXUnloadInk            (gxInk target);
void GXLoadTransform        (gxTransform target);
void GXUnloadTransform      (gxTransform target);
void GXLoadColorSet         (gxColorSet target);
void GXUnloadColorSet       (gxColorSet target);
void GXLoadColorProfile     (gxColorProfile target);
void GXUnloadColorProfile   (gxColorProfile target);
void GXLoadTag              (gxTag target);
void GXUnloadTag            (gxTag target);
```