

Tag Objects

This chapter describes tag objects and the functions you can use to manipulate them. Tag objects encapsulate application-defined information that can provide information about or modify the behavior of the QuickDraw GX objects associated with those tag objects.

Read this chapter if you need to create or modify tag objects. Other chapters in this book describe the functions you use to add tag objects to or delete tag objects from specific other kinds of QuickDraw GX objects, such as shapes or styles.

Before reading this chapter, you should be familiar with the information in the chapter “Introduction to QuickDraw GX” in this book.

This chapter introduces QuickDraw GX tag objects and describes their properties. It then shows how to use the QuickDraw GX tag-manipulation functions to

- create and manipulate tag objects
- manipulate tag object properties
- directly manipulate tag contents

About Tag Objects

A tag object is a private data structure whose purpose is to allow any kind of application-defined information to be attached to a QuickDraw GX object. An object such as a shape or transform can be “tagged” with data or code that alters its behavior in specific situations or provides extra information about it. For example, you can attach identifying strings to objects with tags, or you can alter the way an object is displayed on a particular imaging device by attaching a tag to it that contains imaging commands specific to that device. For example, QuickDraw GX uses tag objects to hold PostScript commands used for printing to PostScript devices.

QuickDraw GX identifies an individual tag object through a *tag reference*. To obtain information about a tag object, you must send its reference as a parameter to a QuickDraw GX function (except that you can determine if two references identify the same tag object simply by comparing them for equality, and you can examine a reference to see if it is `nil`).

Tag objects are further identified by *tag type*, a designation that you can use to identify the tag object’s purpose and format.

A tag object is attached to its associated object by means of a *tag list*, a property that most QuickDraw GX objects have. A tag list is an array of references to the tag objects attached to an object. Objects can thus have more than one attached tag object. You cannot attach a tag object to a printing object, a font object, a graphics client object, or a tag object itself; there is no tag list property for those objects.

Tag Objects

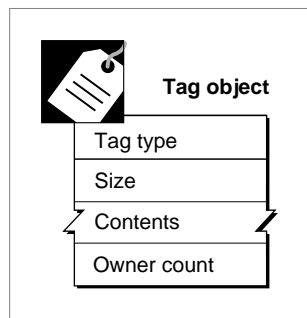
Because tags are QuickDraw GX objects, they can be shared. A single tag object can be attached to more than one other object; the owner count of the tag object tells you how many references to it exist. Tags also have all the other advantages of QuickDraw GX objects: they are accessible from objects in accelerator memory, they can be unloaded to disk and reloaded automatically, and they can be flattened and included in a spool file.

Tag Object Properties

The interface to tag objects is entirely procedural. You manipulate the information in a tag object by modifying its properties using QuickDraw GX functions.

Tag objects have four accessible properties, as shown in Figure 8-1. Note that, because a tag is an object and not a data structure, the order of the properties as shown in Figure 8-1 is completely arbitrary.

Figure 8-1 The tag object and its properties



These are the four accessible properties of a tag object:

- **Tag type.** A 4-byte value that specifies the type of this tag object. On the Macintosh computer, tag types are typically represented with four-character mnemonics, such as 'DAVE'.
- **Size.** The size in bytes of the contents of the tag object.
- **Contents.** The data that makes up this tag object. QuickDraw GX is unconcerned with the nature of the data; you can place whatever information you wish into the contents of a tag object.
- **Owner count.** The number of existing references to this tag object.

QuickDraw GX provides functions to manipulate each of these tag object properties.

Tag Types

Tag objects have types in order to identify their purpose. For example, if you want to identify a circle that is approximated by a QuickDraw GX path shape, you might attach to it a tag of type 'CRCL'. Then, whenever your application scales a path shape, it can first check to see if there is a tag object of type 'CRCL' attached to that shape. If there is, your application can make sure that the scaling preserves the circularity of the result. If your application has its own circle-drawing function, it can call that function instead of calling `GXDrawShape` to draw the circle.

The creator of a tag object can give it any 4-byte type value, although it is customary to make it a value that can be represented with four 1-byte ASCII characters. Apple Computer, Inc., reserves all tag types that can be represented with lowercase characters only, such as 'dave'. There are no other restrictions, except that a tag type cannot be 0. If you intend your tag type to be exportable (usable by other applications), you can be certain that it will not conflict with other applications' tag types if you use your application's creator type, as registered with Macintosh Developer Technical Support, as your tag type.

Note

A four-character tag type is not portable. On systems other than the Macintosh, the tag type may print or display quite differently, and one with the same appearance may have a very different numeric value. For maximum portability, it is best to define tag types with hexadecimal values, such as

```
#define daveTag 0x44415645          /* 'DAVE' */
```

In this way, the tag type `daveTag` will be correct regardless of the architecture of the machine it is defined on. ♦

Some tag types have already been defined for specific purposes. QuickDraw GX uses tag objects for printing synonyms, which include data such as PostScript commands that replace the QuickDraw GX drawing commands for printing on PostScript printers. QuickDraw GX also uses tag objects to list fonts and individual glyphs used by flattened shapes. There are several currently defined tag types for printing synonyms, one for flattened fonts and glyphs, plus other tag types for various other purposes. Table 8-1 lists some of the currently defined tag types.

Tag Objects

Table 8-1 Defined tag types for tag objects

Tag type	Constant	Explanation
'flst'	gxFlatFontListItemTag	Tag object contains a list of fonts used by the associated object (a flattened shape).
'bfil'	gxBitmapFileAliasTagType	Tag object contains an alias record specifying the file that holds the pixel image for the associated bitmap shape object.
'post'	gxPostScriptTag	Tag object contains PostScript instructions replacing the information in the associated object.
'psct'	gxPostControlTag	Tag object contains a control flag plus font and encoding information for a PostScript printer.
'sdsh'	gxDashSynonymTag	Tag object contains dash information to be used by the PostScript setdash operator.
'lcap'	gxLineCapSynonymTag	Tag object contains cap information to be used by the PostScript setlinecap operator.
'half'	gxFormathalftoneTag	Tag object contains halftone information to be used by a PostScript printer.
'ptrn'	gxPatternSynonymTag	Tag object contains pattern information to be used by vector devices.
'cubx'	gxCubicSynonymTag	Tag object contains a cubic Bézier representation of a curve or path.

Uses for Tag Objects

Tags were originally devised as generalized, object-based equivalents to QuickDraw picture comments. Picture comments are used for sending PostScript commands during printing and for other purposes. A tag is like a structured comment: it has a specific type, it is attached to a specific item (an object), and it has a specific scope (that object).

Tag objects can, however, be used for more than picture comments. For example, tags can provide general information. For a large, complex document that can be represented as a single picture shape, it may be important to know what application originally created the shape, or what ranges of properties (colors, pixel depths, page sizes, and so on) may be found in it. The shape may contain one or more references to tag objects that hold that information.

Tag Objects

You can also use tags to attach identifying strings to objects, for debugging or other purposes. You could name shapes with strings like “oval” or “topographic contour 3242”; you could name ink objects with strings like “cobalt blue” or “blend mode.” You could also use tag objects to attach user comments or descriptions to shapes.

Tag objects may also provide alternate behavior for an object when it is used outside the QuickDraw GX environment. For example, QuickDraw GX uses tag objects to store PostScript commands for drawing shapes to PostScript printers.

If you want to be able to draw a shape object on a system that uses a different coordinate system from QuickDraw GX, you could calculate and store the alternate coordinates in a tag attached to the shape. If you are working in a completely different graphics system that is a superset of QuickDraw GX, you could store that system’s graphics information as tag objects attached to the QuickDraw GX objects you create.

IMPORTANT

In most cases, an application-created tag object cannot change the behavior of its associated object within the QuickDraw GX environment. No geometric operations, no drawing operations, and no testing operations (such as `GXEqualShape`) take the existence of tag objects into account. (One minor exception is `GXFlattenShape`; see its description in the chapter “Shape Objects” in this book. A second exception is that drawing a bitmap whose pixel image is disk-based requires QuickDraw GX to use information in a tag object.) Other than that, tag objects can alter behavior only where graphics operations are overridden (as in printing), or where your application itself changes an operation based on the contents of a tag object. ▲

Using Tag Objects

This section describes how to create and manipulate tag objects and their contents. It describes how you can

- create and manipulate tag objects
- manipulate tag object properties
- directly manipulate tag contents

Creating and Manipulating Tag Objects

This section describes how you can create and interact with tag objects as whole entities. To manipulate tag object properties, use the functions described in the section “Manipulating Tag Object Properties” beginning on page 8-9.

Tag Objects

Creating and Deleting a Tag Object

QuickDraw GX provides the `GXNewTag` function to allow you to create a new tag object. When you create the tag object, you provide its contents and you specify its tag type. Once you have created the tag object, you can attach it to any QuickDraw GX object (except another tag object) by making a call such as `GXSetShapeTags`, `GXSetInkTags`, or `GXSetColorProfileTags`.

Except when it overrides its own functions (as during printing), QuickDraw GX does not access or use the internal structure of the tag object you create; its contents and function are entirely up to you. Nor does QuickDraw GX make any restrictions on the tag type designation you provide, except that it cannot be zero. QuickDraw GX does not make any use of tag type except to use it for retrieving and replacing tag objects according to your instructions.

To delete your application's reference to a tag object, call the `GXDisposeTag` function. Calling `GXDisposeTag` may or may not actually release the memory allocated for the object, depending on the object's owner count. The function decreases the owner count of the tag object by 1; if that brings the owner count to zero, the object is completely deleted and its memory released. See "Manipulating a Tag Object's Owner Count" on page 8-11. Owner counts and what it means to dispose of an object are described in general in the chapter "Introduction to Objects" in this book.

Listing 8-1 is a library function that takes an arbitrary amount of data, makes it into a tag object of a given tag type, and attaches it to a specified shape. The function uses the `GXNewTag` function to create the tag object, and the `GXDisposeTag` function to dispose of its reference to the tag object after attaching it to the shape.

Listing 8-1 Adding data to a shape as a tag object

```
void AddShapeUser(gxShape source, const void *data,
                 long length, long type)
{
    gxTag tempItem;
    tempItem = GXNewTag(type, length, data);
    GXSetShapeTags(source, 0, 0, 0, 1, &tempItem);
    GXDisposeTag(tempItem);
}
```

The `GXNewTag` function is described on page 8-13. The `GXDisposeTag` function is described on page 8-14.

Copying, Comparing, and Cloning Tag Objects

You can use the `GXCopyToTag` function to copy the information from one tag object to another or to create a new copy of an existing tag object.

You can test if two references refer to the same tag object by simply testing the references for equality. You can also compare two different tag objects for equality with the `GXEqualTag` functions. For two tag objects to be equal, their tag types and contents must be identical, although their owner counts need not be.

Object copies created with the `GXCopyToTag` function are always equal, by the criteria of `GXEqualTag`, to the objects from which they were copied.

In certain circumstances, you may want to copy a reference to a tag object without actually copying the object. This is called cloning, and you can use the `GXCloneTag` function to clone a tag object. Functionally, `GXCloneTag` does nothing more than increase the owner count of the tag object. For more information about cloning objects, see the chapter “Introduction to Objects” in this book. For information on manipulating owner counts, see the section “Manipulating a Tag Object’s Owner Count” on page 8-11.

The `GXCopyToTag` function is described on page 8-15. The `GXEqualTag` function is described on page 8-16. The `GXCloneTag` function is described on page 8-17.

Loading and Unloading Tag Objects

Although you rarely need to, you can influence memory-allocation decisions involving objects that you have created. If your application needs to have a tag object in memory, you can force QuickDraw GX to load the tag object into memory. When your application no longer needs the tag object in a loaded state, you can instruct QuickDraw GX to unload it.

You call the `GXLoadTag` function to make sure that a tag object is in memory; if necessary, QuickDraw GX brings the object into memory from an unloaded state. You can call the `GXUnloadTag` function to instruct QuickDraw GX that it is free to unload the tag object at any time. These functions are described in the memory management chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

Manipulating Tag Object Properties

This section describes how to manipulate the object properties of tag objects: tag type, contents, and owner count. To create and interact with tag objects as whole entities, use the functions described in the previous section, “Creating and Manipulating Tag Objects” beginning on page 8-7.

Tag Objects

Getting and Setting a Tag Object's Tag Type and Contents

Fundamentally, tag objects are nothing but holders for whatever information or data that you want to attach to a QuickDraw GX object. If you want to access that information to inspect it or modify it, you can set up a buffer and call the `GXGetTag` function for a given tag object. `GXGetTag` places a copy of the tag object's information in your buffer, and it also returns the tag type of the tag object.

You can then modify the information in the buffer in any way you need to, and you can also change the tag type of the object, if desired. (Tag types that can be represented as four lowercase characters, such as 'abcd', are reserved by Apple Computer, Inc.) To return the modified contents or tag type to the tag object they came from, you then call the `GXSetTag` function.

Listing 8-2 is a library function that retrieves the data of the first tag object of a specified tag type or index attached to a given shape. If you specify an index and a buffer length to put the data in, the function returns the contents and tag type of the found tag. The function result is the number of tags found that fit the criteria you specify. The function uses the `GXGetTag` function to retrieve the tag object's contents.

Listing 8-2 Retrieving the contents of a tag object

```
long GetShapeUser(gxShape source, void *data, long *length, long
requestedType, long *foundType, long index)
{
    if( index )           /* if index nonzero, get specific tag */
    {
        gxTag tempItem;

        if( GXGetShapeTags(source, requestedType,
                           index, 1, &tempItem) )
        {
            long tempLength = GXGetTag(tempItem, foundType, data);
            if( length )
                *length = tempLength;
            return 1;           /* no. of tags found */
        } else
            return 0;
    }
    else                   /* otherwise just get tags of req. type */
        return GXGetShapeTags(source, requestedType,
                              1, gxSelectToEnd, nil);
}
```

The `GXGetTag` function is described on page 8-18.

Tag Objects

Using the `GXGetTag` and `GXSetTag` functions involves working with a copy of the tag object's contents in a buffer in application memory. You can also gain direct access to a tag object's contents (but not tag type) in QuickDraw GX memory, if desired. See the section "Directly Manipulating Tag Object Contents" beginning on page 8-11.

Manipulating a Tag Object's Owner Count

The owner count of an object indicates the number of current references to that object. In general, QuickDraw GX manages owner counts for you. For example, when you create a new tag object, QuickDraw GX sets the owner count of the new tag object to 1, which corresponds to the variable your application uses to reference the tag object. As another example, when you assign an existing tag object to a shape or transform (or any other object), QuickDraw GX increments the tag object's owner count, corresponding to the new reference to the tag object contained in the style or transform.

If you want to directly manage the owner count of a tag object yourself, or if you want to know whether a tag object is shared, you can

- use the function `GXGetTagOwners` to determine the current owner count
- use the function `GXCloneTag` to increment the owner count, whenever you create a new reference to the object
- use the function `GXDisposeTag` to decrement the owner count, deleting the tag object and freeing the memory used by it if the owner count goes to 0

The `GXGetTagOwners` function is described on page 8-20.

Note

In the chapter "Style Objects" in this book, the section on manipulating a style object's owner count discusses two common owner-count problems and how to avoid them. The problems are discussed in terms of style objects, but they apply equally well to tag objects. Refer to that discussion if you find that tag objects you create have owner counts that are higher or lower than you expect. ♦

Directly Manipulating Tag Object Contents

Unlike with most properties of most objects, QuickDraw GX allows you to directly manipulate a tag object's contents in QuickDraw GX memory. This capability is provided as a convenience, so that you do not have to make a copy of the data; you can achieve the same results by working with a copy of the information in application memory and then replacing it in the object, using the `GXGetTag` and `GXSetTag` functions. See "Getting and Setting a Tag Object's Tag Type and Contents" on page 8-10 for information on working with `GXGetTag` and `GXSetTag`.

As with `GXGetTag` and `GXSetTag`, the direct-manipulation functions do not provide you with information about the format or organization of the contents of a tag object; they simply give you a pointer to the contents. How you manipulate that information depends on the type of tag you are accessing.

Tag Objects

Working with data in QuickDraw GX memory requires that you lock the data before accessing it so that it cannot be relocated or unloaded from memory until you are finished. You first call the `GXLockTag` function to make sure the tag object doesn't move until you are finished with it. You then call the `GXGetTagStructure` function, which returns a pointer to and the size of the shape's contents. You can then modify the data as needed. Once finished, you call `GXUnlockTag` to free the tag object for relocation as needed by QuickDraw GX.

IMPORTANT

Memory-handling complications can occur with locked objects. Locking an object fragments the QuickDraw GX heap, which can result in lower performance. Furthermore, if a fragmented-memory condition occurs during a call, QuickDraw GX may unlock all objects and restart the call. Therefore, be careful about performing memory-intensive operations while there are locked objects in QuickDraw GX memory; they may become unlocked without warning. ▲

The `GXLockTag` function is described on page 8-21. The `GXGetTagStructure` function is described on page 8-23. The `GXUnlockTag` function is described on page 8-22.

Attaching Tags to a QuickDraw GX Object

Most QuickDraw GX objects (other than tag objects themselves) can have one or more attached tag objects. Each object has a property called a *tag list*, which is an array of references to tag objects. You can retrieve and assign tag references with QuickDraw GX functions. For example, to retrieve one or more of the tags attached to a shape object, you call the `GXGetShapeTags` function. To add one or more tags to a shape object's tag list, you call the `GXSetShapeTags` function.

Besides tag objects, other objects that you cannot directly attach tags to include printing objects, font objects, and graphics client objects.

The `GXGetShapeTags` and `GXSetShapeTags` functions are described in the chapter "Shape Objects" in this book; the equivalent calls for other kinds of objects are described in the chapters that document those objects.

Tag Objects Reference

This section provides reference information to the data structures and functions that allow you to create and manipulate tag objects and alter their properties.

Constants and Data Types

This section describes the data type that you use to gain access to tag objects.

The Tag Object

QuickDraw GX provides you with access to an individual tag object through a `gxTag` reference:

```
typedef struct gxPrivateTagRecord *gxTag;
```

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

Functions

This section describes the functions with which you can

- create and manipulate tag objects
- manipulate tag object properties, including the contents of the tag object
- directly manipulate the contents of a tag object, in-place in QuickDraw GX memory

Creating and Manipulating Tag Objects

The functions in this section allow you to create and manipulate tags as QuickDraw GX objects.

GXNewTag

You can use the `GXNewTag` function to create a new tag object.

```
gxTag GXNewTag(long tagType, long length, const void *data);
```

<code>tagType</code>	A 4-byte identifier specifying the type of tag object to be created.
<code>length</code>	The length in bytes of the data to place in the tag object.
<code>data</code>	A pointer to the data to place in the tag object.

function result A reference to the newly created tag object.

Tag Objects

DESCRIPTION

The `GXNewTag` function creates a tag object, with an owner count of 1, containing whatever data you supply. The tag type is an application-defined, 4-byte tag (commonly expressed on the Macintosh with four characters, such as 'Cary') that you supply to specify the type of data contained in the tag object. You cannot supply a value of 0.

You can specify a value of zero for the `length` parameter, in which case `GXNewTag` creates the tag object but places no data into it. (In that case, the data pointer must be `nil`, or else `GXNewTag` posts an `inconsistent_parameters` error.)

SPECIAL CONSIDERATIONS

You cannot specify a tag type of zero. Apple Computer, Inc., reserves all tag types that can be expressed as four lowercase characters (such as 'atag').

Tag types expressed as four characters may not be portable to systems other than the Macintosh; tag types defined numerically are portable.

If no error occurs, the `GXNewTag` function creates a tag object; you are responsible for disposing of that object when you no longer need it.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`type_is_nil`
`inconsistent_parameters`

SEE ALSO

Currently defined tag types for tag objects are listed in Table 8-1 on page 8-6.

GXDisposeTag

You can use the `GXDisposeTag` function to release a reference to a tag object.

```
void GXDisposeTag(gxTag target);
```

`target` A reference to the tag object to dispose of.

Tag Objects

DESCRIPTION

The `GXDisposeTag` function decrements the owner count of the tag object specified by the `target` parameter. `GXDisposeTag` deletes the tag object and releases any memory used by it if the owner count goes to zero.

SPECIAL CONSIDERATIONS

If you attempt to alter a tag object associated with a screen view device, this function posts a `tag_access_restricted` error.

ERRORS, WARNINGS, AND NOTICES**Errors**

<code>tag_is_nil</code>	
<code>cannot_dispose_locked_tag</code>	(debugging version)
<code>tag_access_restricted</code>	(debugging version)

SEE ALSO

Owner counts for tag objects are discussed in the section “Copying, Comparing, and Cloning Tag Objects” beginning on page 8-9, and in the section “Manipulating a Tag Object’s Owner Count” beginning on page 8-11.

To examine the owner count of a tag object, use the `GXGetTagOwners` function, described on page 8-18. To increment the owner count of a tag, use the `GXCloneTag` function, described on page 8-17.

GXCopyToTag

You can use the `GXCopyToTag` function to copy the contents of one existing tag object into another, or to create a new tag object and copy the contents of an existing tag object into it.

```
gxTag GXCopyToTag(gxTag target, gxTag source);
```

target A reference to the tag object to copy into. If you specify `nil` for this parameter, the `GXCopyToTag` function creates a new tag object.

source A reference to the tag object whose contents you want to copy.

function result A reference to the copy (that is, the target tag object).

Tag Objects

DESCRIPTION

The `GXCopyToTag` function copies the contents of an existing tag object to another, or it creates a new tag object and copies the contents of an existing tag object to it. The function copies the tag type and contents (but not the owner count) of the tag object specified by the `source` parameter into the tag object specified by the `target` parameter.

If you specify `nil` for the `target` parameter, the `GXCopyToTag` function creates a new tag object and copies the source tag object's tag type, contents, and owner count into it.

You can use the `GXCopyToTag` function to create a copy of a tag object and then modify it without changing the original.

SPECIAL CONSIDERATIONS

If you specify `nil` for the `target` parameter and no error occurs, the `GXCopyToTag` function creates a tag object; you are responsible for disposing of that object when you no longer need it.

If you attempt to alter a tag object associated with a screen view device, this function posts a `tag_access_restricted` error.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`tag_is_nil`
`tag_access_restricted` (debugging version)

SEE ALSO

To create a new tag object that is not a copy of an existing tag object, use the `GXNewTag` function, described on page 8-13.

To compare two tag objects, use the `GXEqualTag` function, described in the next section.

GXEqualTag

You can use the `GXEqualTag` function to determine whether two tag objects are equal.

```
boolean GXEqualTag(gxTag one, gxTag two);
```

`one` A reference to one of the tag objects to test for equality.

`two` A reference to the other tag object to test for equality.

function result `true` if the two tag objects are equal; `false` otherwise.

Tag Objects

DESCRIPTION

The `GXEqualTag` function returns `true` if the two specified tag objects are equal. For two tag objects to be equal, they must have identical tag types and contents, although their owner counts need not be identical.

ERRORS, WARNINGS, AND NOTICES**Errors**

`out_of_memory`
`tag_is_nil`

SEE ALSO

To make a copy of a tag object that is equal by the criteria of this function, use the `GXCopyToTag` function, described in the previous section.

GXCloneTag

You can use the `GXCloneTag` function to clone a tag object—that is, to add a reference to it and increment its owner count.

```
gxTag GXCloneTag(gxTag source);
```

`source` A reference to the tag object to clone.

function result A reference to the cloned tag object.

DESCRIPTION

The `GXCloneTag` function increments the owner count of the tag object referenced in the `source` parameter. You typically use this function when you want to create another reference to an existing tag object instead of creating a distinct copy of the tag.

This function returns as its function result a reference to the tag object—the same reference you pass in as the `source` parameter. It also increments the tag object's owner count.

ERRORS, WARNINGS, AND NOTICES**Errors**

`tag_is_nil`
`tag_access_restricted` (debugging version)

Tag Objects

SEE ALSO

Owner counts for tag objects are discussed in the section “Copying, Comparing, and Cloning Tag Objects” beginning on page 8-9, and in the section “Manipulating a Tag Object’s Owner Count” beginning on page 8-11.

To examine the owner count of a tag object, use the `GXGetTagOwners` function, described on page 8-20. To decrement the owner count of a tag, use the `GXDisposeTag` function, described on page 8-14.

Manipulating Tag Object Properties

The functions in this section allow you to manipulate the object properties of tag objects: their tag types, contents, and owner counts.

GXGetTag

You can use the `GXGetTag` function to retrieve the tag type and contents of a tag object.

```
long GXGetTag(gxTag source, long *tagType, void *data);
```

<code>source</code>	A reference to the tag object whose contents you want to retrieve.
<code>tagType</code>	A pointer to a value specifying a tag object type. On return, specifies the type of tag object referenced in the <code>source</code> parameter.
<code>data</code>	A pointer to a buffer. On return, the buffer holds the contents of the source tag object.

function result The size in bytes of the contents of the source tag object.

DESCRIPTION

The `GXGetTag` function returns the tag type and contents of the source tag object in the `tagType` and `data` parameters, respectively. Its function result is the size of the information returned in the `data` array.

Before calling `GXGetTag`, you must allocate an array of sufficient size to hold the contents of the tag object. If instead you pass `nil` for the `data` parameter, `GXGetTag` does not return the tag contents, but nonetheless returns (as its function result) the size of the contents. Thus you can make an initial call to `GXGetTag` to determine the size of buffer to allocate, and then call `GXGetTag` once more to get the contents.

The `GXGetTag` function is different from the `GXGetTagStructure` function in that it returns a copy of the tag object’s contents in a buffer that you have allocated in application memory. The `GXGetTagStructure` gives you direct access to the contents of a tag object in QuickDraw GX memory.

Tag Objects

Although `GXGetTag` returns the contents of a tag object, it returns no information other than size about the format or organization of the tag's contents. You must know the internal structure of a tag object's contents in order to manipulate it.

*ERRORS, WARNINGS, AND NOTICES***Errors**

`out_of_memory`
`tag_is_nil`

SEE ALSO

To replace the tag type and contents of a tag object, use the `GXSetTag` function, described in the next section.

The `GXGetTagStructure` function is described on page 8-23.

GXSetTag

You can use the `GXSetTag` function to replace the tag type or contents of a tag object.

```
void GXSetTag(gxTag target, long tagType, long length,
              const void *data);
```

<code>target</code>	A reference to the tag object whose contents you want to replace.
<code>tagType</code>	The new tag type to assign to the tag object referenced in the <code>target</code> parameter. If you pass 0 for this parameter, the tag type remains unchanged.
<code>length</code>	The length in bytes of the new data to place in the tag object. If you pass 0 for this parameter, the contents of the tag object remain unchanged and the <code>data</code> parameter is ignored.
<code>data</code>	A pointer to the new data to place in the tag object. If you pass <code>nil</code> for this parameter, the contents of the tag object (up to the length specified by <code>length</code>) remain unchanged.

DESCRIPTION

The `GXSetTag` function assigns the specified tag type and contents to the target tag object. You can set three of its parameters for different purposes:

- To change only the tag type and not the contents of a tag object, pass 0 in the `length` parameter, `nil` in the `data` parameter, and a nonzero value for `tagType`.
- To change only the contents and not the tag type, pass 0 in the `tagType` parameter, and valid values for `length` and `data`.

Tag Objects

- To resize the tag object without changing its contents or type, pass the new size in the `length` parameter, `nil` in the `data` parameter, and 0 in the `tagType` parameter. If the new size of the contents is smaller than the previous size, the data is truncated to fit the new size. If the new size is greater than the previous size, the tag object is resized accordingly, but a `new_tag_contains_invalid_data` warning is posted.

Note that calling `GXSetTag` is different from using the `GXGetTagStructure` function to manipulate the contents of a tag object. Unlike `GXGetTagStructure`, `GXSetTag` allows you to change the size of the tag object.

ERRORS, WARNINGS, AND NOTICES

Errors`out_of_memory``tag_is_nil``inconsistent_parameters`

(debugging version)

`tag_access_restricted`

(debugging version)

Warnings`new_tag_contains_invalid_data`

SEE ALSO

To retrieve the tag type and contents of a tag object, use the `GXGetTag` function, described in the previous section.

To directly manipulate the contents of a tag object in QuickDraw GX memory, rather than replacing its entire contents and tag type, use the `GXGetTagStructure` function, described on page 8-23.

GXGetTagOwners

You can use the `GXGetTagOwners` function to determine the number of references to a particular tag object.

```
long GXGetTagOwners(gxTag source);
```

`source` A reference to the tag object to find the owner count of.

function result The owner count of the tag object referenced in the `source` parameter.

DESCRIPTION

The `GXGetTagOwners` function returns the owner count of the referenced tag object. The owner count is the current number of references to the tag object.

ERRORS, WARNINGS, AND NOTICES**Errors**`tag_is_nil`**SEE ALSO**

Owner counts for tag objects are discussed in the section “Copying, Comparing, and Cloning Tag Objects” beginning on page 8-9, and in the section “Manipulating a Tag Object’s Owner Count” beginning on page 8-11.

To increment the owner count of a tag, use the `GXCloneTag` function, described on page 8-17. To decrement the owner count of a tag, use the `GXDisposeTag` function, described on page 8-14.

Directly Manipulating the Data in a Tag Object

This section describes the functions you use to directly manipulate the contents of a tag object in QuickDraw GX memory.

GXLockTag

You can use the `GXLockTag` function to load a tag object into QuickDraw GX memory and lock its contents into a fixed memory location.

```
void GXLockTag(gxTag target);
```

`target` A reference to the tag object to be loaded and locked.

DESCRIPTION

The `GXLockTag` function prevents a tag object from being relocated. To directly edit a tag’s contents, you must first call `GXLockTag`. You can then call `GXGetTagStructure` and edit the tag’s contents. After editing, you must call `GXUnlockTag` to release the tag for relocation.

SPECIAL CONSIDERATIONS

To avoid fragmenting the QuickDraw GX heap, you should call the `GXUnlockTag` function as soon as possible after calling `GXLockTag`.

You can nest calls to these direct-access routines, but be sure to call `GXUnlockTag` as many times as you call `GXLockTag`. Version 1.0 of QuickDraw GX prohibits more than 255 nested calls to `GXLockTag`.

Tag Objects

*ERRORS, WARNINGS, AND NOTICES***Errors**

out_of_memory
tag_is_nil

SEE ALSO

The GXUnlockTag function is described in the next section. The GXGetTagStructure function is described on page 8-23.

GXUnlockTag

You can use the GXUnlockTag function to allow QuickDraw GX to relocate, flatten, or unload a tag object.

```
void GXUnlockTag(gxTag target);
```

target A reference to the tag object to unlock.

DESCRIPTION

The GXUnlockTag function frees a previously locked tag object for relocation. To directly edit a tag's contents, you must first call GXLockTag. You can then call GXGetTagStructure and edit the tag's contents. After editing, you must call GXUnlockTag.

You cannot dispose of a tag that is locked. You must first call GXUnlockTag on a locked tag object before calling GXDisposeTag.

SPECIAL CONSIDERATIONS

To avoid fragmenting the QuickDraw GX heap, you should call the GXUnlockTag function as soon as possible after calling GXLockTag.

You can nest calls to these direct-access routines, but be sure to call GXUnlockTag as many times as you call GXLockTag.

*RESULT CODE***Errors**

tag_is_nil

Notices (debugging version)

tag_not_locked

SEE ALSO

The `GXLockTag` function is described in the previous section. The `GXGetTagStructure` function is described in the next section.

The `GXDisposeTag` function is described on page 8-14.

GXGetTagStructure

You can use the `GXGetTagStructure` function to get a pointer to the contents of a tag object.

```
void *GXGetTagStructure(gxTag source, long *length);
```

source A reference to the tag object whose contents you need access to.

length A pointer to a value. On return, contains the size in bytes of the contents of the tag object referenced in the `source` parameter.

function result A pointer to the contents of the source tag object.

DESCRIPTION

The `GXGetTagStructure` function returns a pointer to the contents of the tag object referenced in the `source` parameter. To directly edit a tag's contents, you must first call `GXLockTag`. You can then call `GXGetTagStructure` and edit the tag's contents. After editing, you must call `GXUnlockTag`.

The `GXGetTagStructure` function is different from the `GXGetTag` function in that it gives you direct access to the contents of a tag object in QuickDraw GX memory. The `GXGetTag` function returns a copy of the tag object's contents in a buffer that you have allocated in application memory.

To edit the contents of a tag object, you need to know its format and organization. `GXGetTagStructure` returns a pointer and a size only; it does not provide you with any information about the internal structure of the tag's contents.

This function does not provide access to tag type information.

SPECIAL CONSIDERATIONS

Note that using the `GXGetTagStructure` is different from calling `GXSetTag`, in that it does not allow you to change the size of the tag object.

This function is available for your convenience, in that you do not have to make a copy of the tag object's data, but is rarely needed. In most cases you can use the `GXGetTag` and `GXSetTag` functions to manipulate tag contents.

Tag Objects

ERRORS, WARNINGS, AND NOTICES

Errors

`out_of_memory`
`tag_is_nil`

Notices (debugging version)

`lockTag_called_as_side_effect`

SEE ALSO

The `GXLockTag` function is described on page 8-21. The `GXUnlockTag` function is described in the previous section.

The `GXGetTag` function is described on page 8-18. The `GXSetTag` function is described on page 8-19.

Summary of Tag Objects

C Summary

Functions

Creating and Manipulating Tag Objects

```

gxTag GXNewTag          (long tagType, long length, const void *data);
void GXDisposeTag       (gxTag target);
gxTag GXCopyToTag       (gxTag target, gxTag source);
boolean GXEqualTag      (gxTag one, gxTag two);
gxTag GXCloneTag        (gxTag source);

```

Manipulating Tag Object Properties

```

long GXGetTag           (gxTag source, long *tagType, void *data);
void GXSetTag           (gxTag target, long tagType, long length,
                        const void *data);
long GXGetTagOwners     (gxTag source);

```

Directly Manipulating the Data in a Tag Object

```

void GXLockTag          (gxTag target);
void GXUnlockTag        (gxTag target);
void *GXGetTagStructure (gxTag source, long *length);

```

