

## Transform Objects

This chapter describes transform objects and the functions you can use to manipulate them. Read this chapter if you need to clip parts of a shape for drawing, modify the position or dimensions of a shape, modify the view ports to which a shape is drawn, or hit-test a shape.

Before reading this chapter, you should be familiar with the information in the chapter “Introduction to QuickDraw GX” in this book. You should also be familiar with shape objects, as discussed in the chapter “Shape Objects” in this book.

For specific information about how transform objects affect bitmap and picture shapes, see *Inside Macintosh: QuickDraw GX Graphics*. For specific information about how transform objects affect typographic shapes, see *Inside Macintosh: QuickDraw GX Typography*. For information about the mathematical foundation of transform mappings, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*. For information about view ports, see the chapter “View-Related Objects” in this book.

This chapter introduces QuickDraw GX transform objects and describes their properties. It then shows how to use QuickDraw GX functions to

- create and manipulate transform objects
- manipulate transform object properties, including the clip, view port list, and hit-test parameters
- perform mapping operations to change the translation, rotation, scale, skew, or perspective of transform objects and shape objects

## About Transform Objects

---

A transform object exists to modify, or *transform*, the appearance or behavior of a shape. Each QuickDraw GX shape consists of a shape object, a style object, an ink object, and a transform object; the transform object specifies where the shape is drawn, how its appearance is transformed when drawn, and how the user can interact with the drawn shape. You can think of a transform object as a filter between the shape object and its drawing destination of one or more view ports.

QuickDraw GX identifies an individual transform object through a transform *reference*. To obtain information about a transform 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 transform object simply by comparing them for equality, and you can examine a reference to see if it is `nil`).

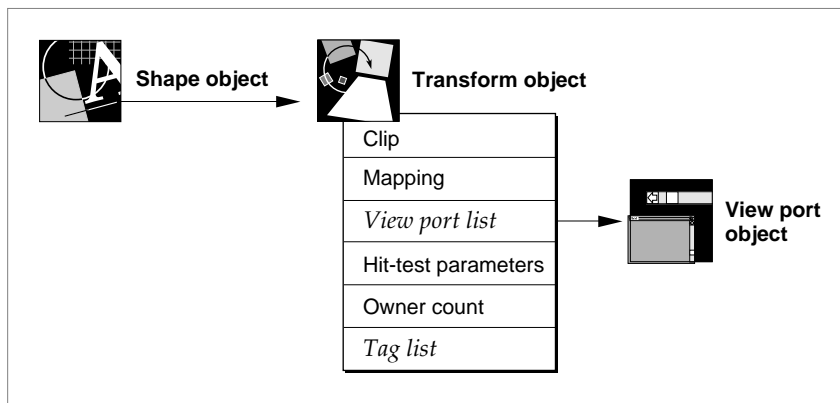
A shape (other than a picture shape) always refers to a single transform object. Several shapes, however, can refer to the same transform object. When they do, the transform object is shared and its transformations apply to all of those shapes. If you use a function that directly manipulates the transform object, the behavior of all shape objects associated with it changes.

## Transform Object Properties

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

Transform objects have six accessible properties, as shown in Figure 6-1. Note that, because a transform is an object and not a data structure, the order of the properties shown in Figure 6-1 is completely arbitrary. Properties in italics are references to other objects.

**Figure 6-1** The transform object and its properties



These are the six accessible properties in a transform object:

- ***Clip***. A reference to a specialized shape geometry that defines the visible area of the shape associated with this transform object. Only the parts of the shape that overlap with the clip remain visible when the shape is drawn. The transform clip is further described in the next section, “Clip.”
- ***Mapping***. A mathematical matrix that specifies the translation, scaling, rotation, skewing, and perspective of the shape associated with this transform object. The transform mapping is further described in the section “*Mapping*” on page 6-10 .
- ***View port list***. An array of references to the view ports that the shape associated with this transform object can be drawn to. The view port list is further described in the section “View Port List” on page 6-11.
- ***Hit-test parameters***. Two values that provide criteria for hit-testing the shape associated with this transform object. The hit-test parameters specify what parts of the shape are to be tested for, and how close to a part a hit point must be to be considered successful. The hit-test parameters are further described in the section “Hit-Test Parameters” on page 6-11.

## Transform Objects

- **Owner count.** The number of existing references to this transform object. General information about owner counts is in the chapter “Introduction to QuickDraw GX” in this book; the section “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16 describes when and how to examine and alter a transform object’s owner count.
- **Tag list.** A list of references to custom information about this transform object, stored in private data structures called tag objects. The chapter “Tag Objects” in this book describes tag objects in general and how you can use them to add custom information to objects.

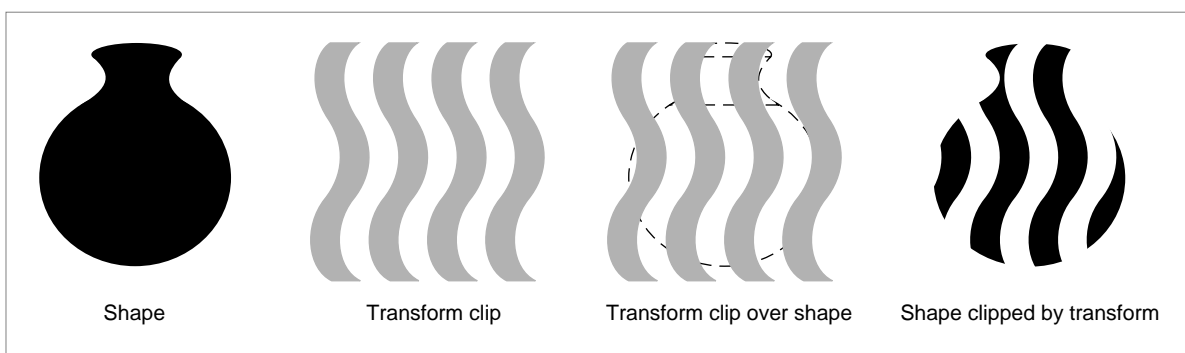
QuickDraw GX provides functions to manipulate each of these transform object properties.

## Clip

The *clip* property of a transform object is a specialized shape geometry that functions as a mask to restrict the visibility of a shape when it is displayed or printed. The clip shape is equivalent to a *primitive shape*, a shape (of any type) whose geometry and fill properties by themselves define the shape. In other words, a primitive shape does not use any information from a style object or transform object to determine its location, dimension, or even pen thickness; all dimensional information about a primitive shape is in the shape object itself.

The filled or framed parts of a transform’s clip define the areas in which the shape attached to that transform show through. Figure 6-2 shows the effect of using a transform object to clip a shape representing a vase. The vase shape references a transform object whose clip property defines a clip shape consisting of four filled paths. Only the parts of the vase that intersect the filled paths are allowed to show through.

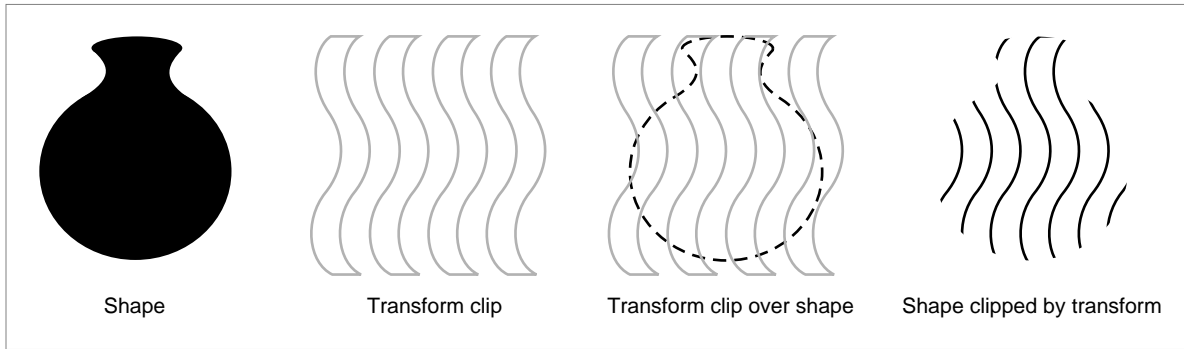
**Figure 6-2** A transform clip



## Transform Objects

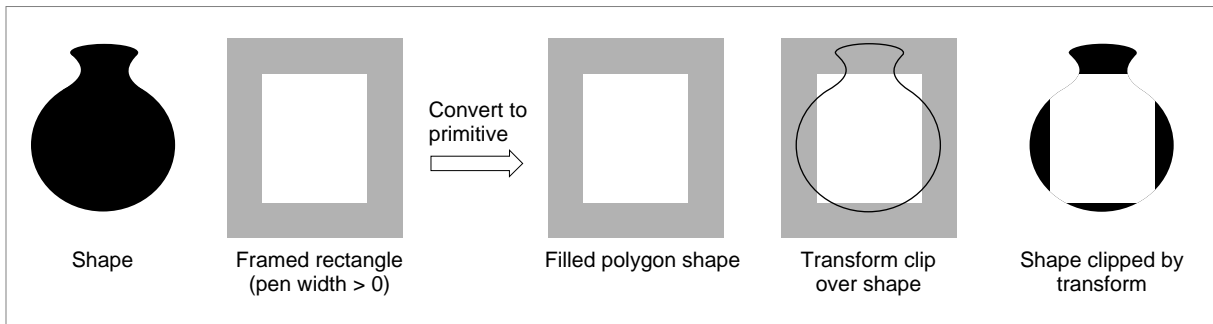
If the clip were a framed path shape instead of the filled path shape shown in Figure 6-2, only the parts of a shape that intersect the frame itself would be visible. And because the pen width is 0 for a primitive shape, the frame would be of hairline width only; the parts of the shape both outside and inside the hairline frame would be clipped out, as shown in Figure 6-3.

**Figure 6-3** A framed transform clip

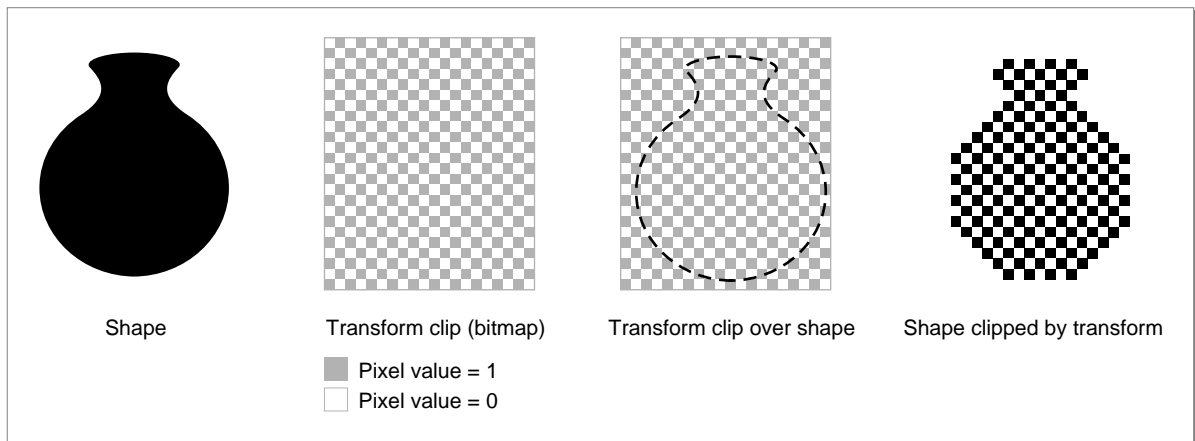


To use a framed shape with nonzero pen width as a clip shape, you first convert it to a primitive shape, at which point it becomes a filled shape in which the filled areas correspond to the pen width in the original framed shape. Figure 6-4 shows an example of converting a framed shape with a nonzero pen width into a transform clip shape.

**Figure 6-4** Converting a framed shape with a nonzero pen width into a transform clip

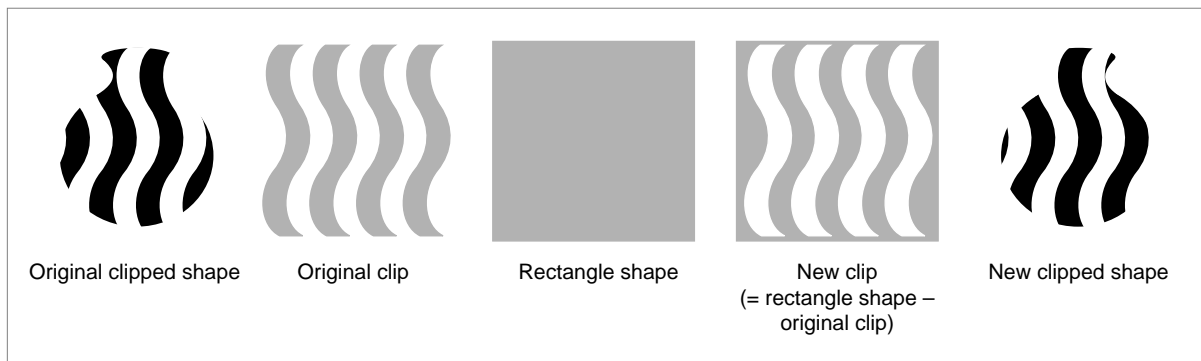


You can use a bitmap shape as a clip, but only if its pixel depth is 1—meaning that each pixel has a value of 0 or 1—and if its color profile is `nil`. When a transform clip is a bitmap, its individual pixels mask the shape that is drawn. A pixel with a value of 1 allows the shape geometry to show through the area covered by that pixel, and a pixel with a value of 0 clips out the part of the shape covered by that pixel. Figure 6-4 shows an example.

**Figure 6-5** Using a bitmap as a transform clip

For more information on bitmap shapes, see the bitmap shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. For more information on primitive shapes, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

QuickDraw GX provides functions that allow you to modify a transform clip by performing constructive geometry operations—such as union and intersection—between it and another shape. With these operations you can build a clip from one or more shapes. For example, you can start with the default clip shape, `gxFullShape`, which allows everything to show through, and use constructive geometry operations to restrict the visibility. Or you can start with an empty shape, `gxEmptyShape`, and use constructive geometry operations to increase visibility. Figure 6-6 shows one example (using reverse difference); the section “Getting, Setting, and Modifying the Transform Clip” beginning on page 6-20 describes the operations you can perform and the QuickDraw GX functions you use to modify a transform clip.

**Figure 6-6** Modifying a transform clip by subtracting it from another shape

## Transform Objects

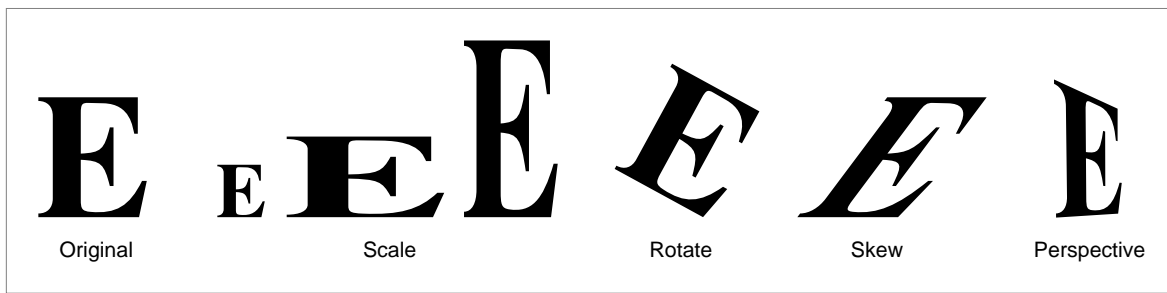
## Mapping

The *mapping* property is a  $3 \times 3$  matrix that specifies one or more transformations that a transform object performs on its associated shape. It is the transforming aspect of the mapping property that gives the transform object its name. You can use the transform mapping to perform the following operations on a shape:

- translation, which changes the position of the shape
- scaling, which shrinks or enlarges the shape horizontally or vertically or both
- rotation, which turns the shape about a fixed point
- skewing, which distorts the shape progressively along a single axis
- perspective, which distorts the shape to provide a three-dimensional appearance

Figure 6-7 shows examples of some of these transformations.

**Figure 6-7** Effects of the transform mapping



You can combine one or more of the possible transformations in a single mapping matrix. For example, you can specify 200 percent scaling and 30-degree rotation in the same mapping. The *identity mapping*, which is a matrix whose elements have the value 1.0 along the diagonal and 0.0 elsewhere, specifies no transformation. An identity mapping applied to a shape leaves it unchanged. The identity mapping is the default mapping for a transform.

One important advantage of having a mapping property separate from a shape's geometry is that you can change the visual appearance of a shape in many different ways and at many different times without ever changing the geometry of the shape itself. This minimizes the accumulation of errors, and also allows a set of identical shape geometries to result in many different appearances. QuickDraw GX provides functions with which you can easily modify the mapping of a transform object to perform translation, scaling, rotation, and skewing. See the section "Modifying the Transform Mapping" beginning on page 6-24 for more information and examples.

If you want to, however, you can also transform a shape by changing its geometry directly. Direct manipulation of shape geometry can be faster than modifying the transform object, and may be more appropriate when you want to change the fundamental nature of a shape. QuickDraw GX provides functions with which you

## Transform Objects

can directly modify the geometry of a shape object to perform translation, scaling, rotation, and skewing. See the section “Modifying Shape Geometry” beginning on page 6-26 for more information and examples.

For general information about the characteristics and capabilities of mappings, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## View Port List

The ***view port list*** property is an array of references to view port objects. The list specifies all of the view ports through which the shape associated with this transform object can be drawn. A transform object must have at least one view port reference in its view port list if any drawing is to occur. If it has more than one view port in the list, drawing occurs to all of its view ports simultaneously.

You may want to have several view ports in a list if you want the shapes associated with a transform object to display in several windows, or in different places in the same window. If you implement offscreen drawing, your transform object can reference both onscreen and offscreen view ports.

The same view port can be in the view port list more than once. For example, you may wish to draw the same shape several times, using a transfer mode that accounts for the color already on the view device. The view port list controls the order in which a shape is drawn: the shape is drawn to the first view port in the list before it is drawn to the second one in the list, and so on. You can use that information to control the order in which colors are manipulated by the transfer mode you have chosen. For information about transfer modes, see the chapter “Ink Objects” in this book.

Like transform objects, view port objects have their own clip and mapping properties that affect how a shape appears when drawn. So also do view device objects. Note that view ports do not correspond to view devices; for example, you don’t need to have multiple view ports in the view port list property just because the computer has several screens. View ports and view devices are described in the chapter “View-Related Objects” in this book.

For further information and examples of manipulating the view port list of a transform object, see “Manipulating the View Port List” beginning on page 6-28.

## Hit-Test Parameters

The ***hit-test parameters*** property of a transform object consists of two values that specify the criteria to be used for hit-testing shapes associated with this transform object. Hit-testing itself is introduced in the chapter “Introduction to QuickDraw GX” and described in more detail in the chapter “Shape Objects,” in this book.

The hit-testing functions `GXHitTestShape` and `GXHitTestPicture` use the hit-test parameters in the transform object. The hit-testing parameters consist of

- a mask that specifies the parts of a shape that are to be tested for a hit
- a value that specifies the tolerance, or distance from any of the parts, that a hit point can be and still be considered to represent a successful hit

## Transform Objects

**Shape-Parts Mask**

The parts of a shape object that you can consider in hit-testing are shown in Table 6-1. They are defined in the `gxShapeParts` enumeration.

**Table 6-1** Shape parts for hit-testing, from the `gxShapeParts` enumeration

Constant	Value	Explanation
<code>gxNoPart</code>	0	Not in any part of the shape. This value is returned by a hit-testing function if no shape part was successfully hit.
<code>gxBoundsPart</code>	0x0001	Anywhere within the bounding rectangle of the shape.
<code>gxGeometryPart</code>	0x0002	Anywhere within the geometry of the shape. If the shape is framed, this includes just the curves or lines that make up the contours; if the shape is filled, this includes all filled areas.
<code>gxPenPart</code>	0x0004	Anywhere in the pen swath. For example, if this shape's style object has its <code>gxCenterFrameStyle</code> attribute set, this includes anywhere within half the pen width on either side of all curves or lines that make up the contours.
<code>gxCornerPointPart</code>	0x0008	On any geometric (on-curve) point in the shape geometry
<code>gxControlPointPart</code>	0x0010	On any (off-curve) control point in the shape geometry.
<code>gxEdgePart</code>	0x0020	On the edge of the geometry; along any of the curves or lines that make up the contours.
<code>gxJoinPart</code>	0x0040	Within the geometry of a join that is part of the shape.
<code>gxStartCapPart</code>	0x0080	Within the geometry of a start cap that is part of the shape.
<code>gxEndCapPart</code>	0x0100	Within the geometry of an end cap that is part of the shape.
<code>gxDashPart</code>	0x0200	Within the geometry of a dash element that is part of the shape.
<code>gxPatternPart</code>	0x0400	Within the geometry of a pattern element that is part of the shape.
<code>gxGlyphBoundsPart</code>	0x0040	(Same value as <code>gxJoinPart</code> .) For a typographic shape, anywhere within the bounding rectangle of an individual glyph.



**Table 6-1** Shape parts for hit-testing, from the `gxShapeParts` enumeration (continued)

Constant	Value	Explanation
<code>gxGlyphFirstPart</code>	0x0080	(Same value as <code>gxStartCapPart</code> .) For a typographic shape with horizontal text, anywhere within the left half of a glyph, including its left side bearing; for vertical text, anywhere in the top half of the glyph. “Left half” or “top half” means half the advance width: half of the distance from the left margin of the left side bearing to the right margin of the right side bearing.
<code>gxGlyphLastPart</code>	0x0100	(Same value as <code>gxEndCapPart</code> .) For a typographic shape with horizontal text, anywhere within the right half of a glyph, including its right side bearing; for vertical text, anywhere in the bottom half of the glyph. “Right half” or “bottom half” means half the advance width: half of the distance from the right margin of the right side bearing to the left margin of the left side bearing.
<code>gxSideBearingPart</code>	0x0200	(Same value as <code>gxDashPart</code> .) For a typographic shape, within a glyph side bearing.
<code>gxAnyPart</code>	0x07FF	Any of the above parts. You can pass this value to a hit-testing function if you want it to test for all possible shape parts.

NOTE Points, control points, contours, joins, caps, dashes, patterns, and other components of geometric shape geometries are described in the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. The `gxCenterFrameStyle` attribute and other style attributes that apply to geometric shapes are described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*.

Glyph bounding rectangles, side bearings, and advance widths are described in the introductory chapter of *Inside Macintosh: QuickDraw GX Typography*.

**Tolerance**

The tolerance is a value that describes how close the tested point must be to a shape part before that part is considered to have been hit. Tolerance is specified in the shape object’s geometry space (except that style-object information is included when testing for pen, joins, caps, dashes, patterns, and typographic shapes). The tolerance is dimensionless (it has no metric, such as inches), and can have any fixed-point value in the range of –32,767.0 to approximately 32,768.0. A tolerance of 0 means that the hit point must exactly coincide with a shape part for a successful hit.

For more information about coordinate spaces, see the chapter “View-Related Objects” in this book.

## Transform Objects

**Setting Up the Parameters**

---

Before calling `GXHitTestShape` or `GXHitTestPicture`, you set up the shape-parts mask in the transform object to specify the shape parts you are interested in testing for. Note that values specifying join, cap, and dash parts in geometric shapes are used in typographic shapes to specify various glyph parts instead. Note also that you can specify no parts or all parts in the mask. You also specify a tolerance, which should be 0 if the hit point must exactly coincide with a shape part for a successful hit. See “Setting Up Hit-Test Parameters” beginning on page 6-30 for more information and examples.

The `GXHitTestShape` function is described in the chapter “Shape Objects” in this book, with additional information for typographic shapes and typographic shape parts in the typographic shapes chapter of *Inside Macintosh: QuickDraw GX Typography*. The `GXHitTestPicture` function is described in the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

**Default Transform Objects**

---

QuickDraw GX provides a default transform object for you. When you first create a shape, that shape’s transform reference is `nil`, which means that QuickDraw GX uses the default transform object as that shape’s transform. (This assumes that you have not modified the default shape object so that it references a specific transform; see the discussion of default shapes in the chapter “Shape Objects” in this book.)

Also, when you explicitly create a transform object, it is initially a copy of the default transform. These are the properties of the default transform object:

- A clip shape that is a full shape. No clipping occurs when QuickDraw GX draws the shape associated with this transform; the clip has no effect.
- A mapping that is the identity mapping. When drawing, QuickDraw GX does not change the position, scaling, skewing, rotation, or perspective of the shape associated with this transform; the mapping has no effect.
- A view port list that contains a single default view port that covers all screen view devices. See the chapter “View-Related Objects” in this book for more information on the default view port.
- Hit-test parameters that are
  - the default shape-parts mask, in which only the `gxBoundsPart` shape part is specified. QuickDraw GX considers the hit successful if the test point falls within the bounding rectangle of the shape associated with this transform.
  - the default hit-test tolerance, which is 0. The test point cannot be any distance outside of the bounding rectangle if the hit is to be considered successful.
- An owner count of 1.
- An empty tag list.

QuickDraw GX provides a function that allows you to reset a transform object to these default values at any time. See the section “Manipulating a Transform Object’s Owner Count” on page 6-19 for more information.

## Using Transform Objects

---

This section describes the basic transform-creation and transform-manipulation capabilities that QuickDraw GX provides. It describes how you can

- create and manipulate transform objects
- manipulate the common transform object properties
- use geometric operations to modify a transform's clip property
- transform a shape by modifying its transform object's mapping
- transform a shape by modifying its geometry
- manipulate a transform's view port list
- set up hit-testing parameters

### Creating and Manipulating Transform Objects

---

This section describes how you can create and interact with transform objects as whole entities—to create, dispose of, copy, compare, and clone them. Manipulating the individual properties of transform objects is described under “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16 and in subsequent sections.

#### Creating and Disposing of Transform Objects

---

QuickDraw GX provides the `GXNewTransform` function to allow you to create a new transform object. You can also create a new transform object by copying an existing one with the `GXCOPYToTransform` function. Once you have created a transform object, you can modify its properties using functions such as those described in the section “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16.

You need to explicitly create a transform object if you want a single nondefault transform to apply to several shapes. You can also indirectly cause the creation of a new transform object by modifying a transform property under certain conditions; see “Implicit Creation of Transform Objects” on page 6-18 for more information.

To delete your application's reference to a transform object, call the `GXDisposeTransform` function, which may or may not actually release the memory allocated for that transform object, depending on the transform's owner count. The function decreases the transform object's owner count by 1; if that brings the owner count to zero, the transform is completely deleted and its memory released. For more discussion of transform-object owner counts, see “Manipulating a Transform Object's Owner Count” on page 6-19.

## Transform Objects

Listing 6-1 is a code fragment that creates a transform object (`myTransform`) and assigns it to a shape object (`myRectangle`).

---

**Listing 6-1**      Creating and disposing of a transform object

```

gxTransform    myTransform;
gxShape        myRectangle;
myTransform =  GXNewTransform();
.
.  /* set the transform object's properties (not shown) */
.
myRectangle =  GXNewRectangle(gxRectangleType);
GXSetShapeTransform(myRectangle, myTransform);
GXDisposeTransform(myTransform);

```

Notice that the code disposes of the `myTransform` reference to the transform object immediately after it is assigned to the shape. The code no longer needs the reference, and this decreases the transform's owner count, allowing it to be deleted as soon as the shape no longer needs it. The proper place to call `GXDisposeTransform` is immediately after you have finished using a transform in your code, even if you know that another object's use prevents the transform from being deleted at that time.

The `GXNewTransform` function is described on page 6-33. The `GXDisposeTransform` function is described on page 6-34.

### Copying, Comparing, and Cloning Transform Objects

---

You can use the `GXCopyToTransform` function to copy information from one transform object to another or to create a new copy of a transform object.

You can test if two transform-object references refer to the same transform object by simply testing the references for equality. You can also compare two different transform objects for equality with the `GXEqualTransform` function. For two transform objects to be equal, their clips, mappings, view port lists, and hit-test parameters must have identical values, although their owner count and tag list do not need to be identical. Transform object copies created with the `GXCopyToTransform` function are always equal, by these criteria, to the transform from which they were copied.

The following code fragment creates a copy (`lineTransform`) of the transform object associated with the default line shape. It then scales the line shape by changing its transform mapping. Finally, it recopies the original transform back into `lineTransform` to restore the unscaled values.

## Transform Objects

```

gxTransform    lineTransform, savedTransform;
gxShape        defaultLine;
defaultLine = GXGetDefaultShape(gxLineType);
lineTransform = GXGetShapeTransform(defaultLine);
savedTransform = GXCopyToTransform(nil, lineTransform);
GXScaleTransform(lineTransform, ff(2), ff(2), 0, 0);
.
.  /* use the scaled transform (not shown) */
.
GXCopyToTransform(lineTransform, savedTransform);
GXDisposeTransform(savedTransform);

```

Note that the first call to `GXCopyToTransform` in the above code creates a new transform object, whereas the second call causes the contents of one transform to be copied into another.

In certain circumstances, you may want to copy a reference to a transform object without actually copying the object itself. For example, you may want two variables to refer to the same transform object, so that editing one of them affects both. Or, you may want to preserve a reference to a transform so that it cannot be inadvertently deleted. This is called cloning a transform; you can use the `GXCloneTransform` function to clone a transform object.

Functionally, `GXCloneTransform` does nothing more than increase the owner count of a transform. The code in Listing 6-2 clones a shape's original transform object to preserve it from being deleted, changes the shape's transform object temporarily to perform several operations (not shown in the example), and then restores the original transform. In this example, the original transform object is called `saved` and the one that is used for the operations is called `newXform`.

**Listing 6-2** Cloning a transform to prevent it from being deleted

```

gxTransform saved = GXGetShapeTransform(aShape);
GXCloneTransform(saved);
GXSetShapeTransform(aShape, newXform);
.
.  /* use the new transform (not shown) */
.
GXSetShapeTransform(aShape, saved);

```

The `saved` transform object must be cloned because, in the process of associating a new transform object with a shape, the `GXSetShapeTransform` function decrements the owner count of the previously associated transform object. Cloning prevents the `saved` object from being deleted because cloning increments the owner count, which prevents the owner count of the `saved` transform object from going down to 0.

## Transform Objects

For more information about cloning objects, see the chapter “Introduction to Objects” in this book. For more information on manipulating transform owner counts, see the section “Manipulating a Transform Object’s Owner Count” beginning on page 6-19 of this chapter.

The `GXCopyToTransform` function is described on page 6-35. The `GXCloneTransform` function is described on page 6-37. The `GXEqualTransform` function is described on page 6-36.

The `GXScaleTransform` function is described on page 6-60. The `GXSetShapeTransform` function is described in the chapter “Shape Objects” in this book.

## Implicit Creation of Transform Objects

---

QuickDraw GX provides two kinds of functions that modify transform properties:

- The first kind, such as `GXSetTransformClip` and `GXSetTransformMapping`, takes a transform reference as a parameter; it directly alters a property of the transform and thus affects all shapes that use that transform. No new transform object is created when you call this kind of function.
- The second kind, such as `GXSetShapeClip` and `GXSetShapeMapping`, takes a shape reference as a parameter; it alters a property of whatever transform object is used by that shape. To keep from inadvertently affecting other shapes that use the same transform, QuickDraw GX creates a copy of the transform object and modifies the copy if more than one shape object shares the transform.

In addition, if you call a function that normally affects only a shape’s geometry, such as `GXRotateShape`, and the shape’s `gxMapTransformShape` attribute is also set, the shape’s transform mapping is changed instead; in that case, if the transform is shared by more than one object, QuickDraw GX creates a copy and modifies the copy.

The `gxMapTransformShape` attribute is described in the chapter “Shape Objects” in this book. How it affects the functions described in this chapter is discussed in the section “Moving, Scaling, Rotating, and Skewing Shapes” beginning on page 6-23.

## Loading and Unloading Transform 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 transform object in memory, you can force QuickDraw GX to load it into memory. When your application no longer needs the transform object in a loaded state, you can instruct QuickDraw GX to unload it.

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

## Manipulating Transform Object Properties

---

This section describes how to manipulate the common object properties of transform objects: owner count and tag list. It also describes how to restore a transform object's properties to their default values.

To manipulate the clip of an transform, see the section “Getting, Setting, and Modifying the Transform Clip” beginning on page 6-20. To manipulate the mapping of a transform, see the section “Moving, Scaling, Rotating, and Skewing Shapes” beginning on page 6-23. To manipulate the view port list of a transform, see the section “Manipulating the View Port List” beginning on page 6-28. To manipulate the hit-test parameters of a transform, see the section “Setting Up Hit-Test Parameters” beginning on page 6-30.

For manipulating a transform object as a whole, see “Creating and Manipulating Transform Objects” beginning on page 6-15.

## Manipulating a Transform 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 transform object, QuickDraw GX sets the owner count of the new transform to 1. When you assign an existing transform object to a shape, QuickDraw GX increments the transform's owner count, corresponding to the new reference to the transform contained in the shape object.

For example, in Listing 6-1 on page 6-16, the call to `GXNewTransform` to create the transform `myTransform` sets its owner count to 1; the subsequent call to `GXSetShapeTransform` increments the owner count of `myTransform`, so it is 2. The call to `GXDisposeTransform` decrements the owner count of `myTransform`, making it 1 again. The transform is not deleted, which is appropriate because it is still used by the shape. If you were to call `GXSetShapeTransform` again to associate a different transform object with the shape, or call `GXDisposeShape` when the shape is no longer needed, the owner count of `myTransform` would decrement again, this time to 0, and it would be deleted.

As another example, the code in Listing 6-2 on page 6-17 clones a transform object before removing its reference from a shape. The cloning increments the transform's owner count, to ensure that the transform is not deleted when its owner count is decremented by the call to `GXSetShapeTransform` that removes it from the shape.

If you want to manage a transform's owner count directly, or if you want to know whether a transform object is shared, you can use the `GXGetTransformOwners` function to determine the owner count of a transform, and the `GXCloneTransform` and `GXDisposeTransform` functions to change the owner count of a transform. The `GXCloneTransform` function increments the transform's owner count, and the `GXDisposeTransform` function decrements the transform's owner count, freeing the memory used by the transform if the owner count goes to 0.

## Transform Objects

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 transform objects. Refer to that discussion if you find that transform objects you create have owner counts that are higher or lower than you expect.

The `GXGetTransformOwners` function is described on page 6-39.

### Getting and Setting a Transform Object’s Tag References

---

You can examine the list of references to tag objects currently associated with a transform object using the `GXGetTransformTags` function. Once you create a tag object, you can attach it to a transform object using the `GXSetTransformTags` function. You can attach as many tag objects as you like to a transform object.

Tag objects and the basic functions for manipulating them are described in the chapter “Tag Objects” in this book. That chapter also lists the common tag types defined and reserved by Apple Computer, Inc.

The `GXGetTransformTags` function is described on page 6-40. The `GXSetTransformTags` function is described on page 6-41.

### Resetting Default Transform Properties

---

If you explicitly create a new transform with the `GXNewTransform` function and then modify its properties, or if you indirectly modify the properties of a shared transform (by calling, for example, `GXSetShapeMapping`) and thereby cause QuickDraw GX to create a new transform, that new transform has nondefault properties. If you want to restore the default transform properties, you can call the `GXResetTransform` function. This function resets the transform’s clip, mapping, view port list, and hit-test parameters to their default values, but does not alter its owner count or tag list.

The `GXResetTransform` function is described on page 6-38.

### Getting, Setting, and Modifying the Transform Clip

---

The clip shape that you specify in a transform object controls the clipping of shapes associated with that transform. The transform clip must be a primitive shape; primitive shapes are described in the geometric operations chapter of *Inside Macintosh*:

*QuickDraw GX Graphics*. QuickDraw GX provides a pair of functions (`GXGetTransformClip`, `GXSetTransformClip`) that get and set the clip of a specified transform, and another pair (`GXGetShapeClip`, `GXSetShapeClip`) that get and set the clip of the transform associated with a specified shape.



## Transform Objects

QuickDraw GX also provides another set of functions with which you can easily modify a clip shape using constructive geometry. Table 6-2 shows the constructive geometry operations you can perform between a transform clip and another shape, in order to modify the clip shape.

**Table 6-2** Constructive geometry operations between transform clips and other shapes



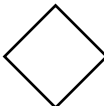

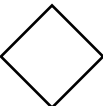
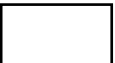

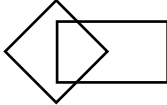

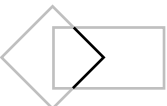
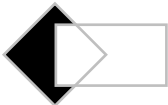
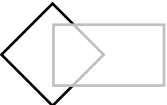
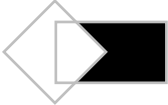
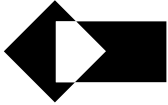
Function	Description
<code>GXUnionTransform</code>	Modifies the clip shape to be the union of it with another shape. Described on page 6-49.
<code>GXIntersectTransform</code>	Modifies the clip shape by intersecting it with another shape. Described on page 6-50.
<code>GXDifferenceTransform</code>	Modifies the clip shape by subtracting another shape from it. Described on page 6-51.
<code>GXReverseDifferenceTransform</code>	Modifies the clip shape by subtracting it from another shape. Described on page 6-52.
<code>GXExcludeTransform</code>	Modifies the clip shape by combining it with another shape in an exclusive-OR (XOR) operation. Described on page 6-53.

To use constructive geometry operations, the clip shape and the shape with which to operate must meet these criteria:

- The clip shape must be a primitive shape and cannot be a picture shape, text shape, or layout shape. (These criteria are automatically met if it is a clip shape.)
- The shape with which to operate cannot be a bitmap shape or picture shape. It should also be a primitive shape, because only its geometry and fill properties are used in the operation.
- If the clip shape's fill is even-odd fill or winding fill, or the inverse of these, the shape with which to operate must also be filled.
- If the clip shape is frame filled, a pen width of 0 is implied, indicating a hairline width to the clip frame. Hairlines are described in the geometric styles chapter of *Inside Macintosh: QuickDraw GX Graphics*.

Figure 6-8 shows several examples of the effects of these operations with a polygon clip combined with a rectangle shape. The figure also shows which combinations of fill types are allowed for each operation.

**Figure 6-8** Constructive geometry operations with a polygon clip and a rectangle shape

	Filled clip	Filled shape	Framed clip	Filled shape	Framed clip	Framed shape
						
<code>GXUnionTransform</code>			Error			
<code>GXIntersectTransform</code>					Error	
<code>GXDifferenceTransform</code>					Error	
<code>GXReverseDifferenceTransform</code>			Error		Error	
<code>GXExcludeTransform</code>			Error		Error	

**Note**

Figure 6-8 does not show a filled clip with a framed shape because this combination of shapes generates an error for any constructive geometry operation. ♦

The following example shows how to create a clip using a constructive geometry operation. The clip is first created as a path shape and assigned to the transform object with `GXSetTransformClip`. That clip is then unioned with another path shape, using `GXUnionTransform`. The geometries of the paths (`path1Geometry` and `path2Geometry`) are not shown.

## Transform Objects

```

gxShape clipShape, pathShape;
gxTransform myTransform;
.
.  /* get or create the transform (not shown) */
.
clipShape = GXNewPaths ((gxPaths *)path1Geometry);
GXSetTransformClip(myTransform, clipShape);
GXDisposeShape(clipShape);

pathShape = GXNewPaths ((gxPaths *)path2Geometry);
GXUnionTransform(myTransform, pathShape);
GXDisposeShape(pathShape);

```

Note that only the geometries of the two path shapes matter; style information is not considered. The `GXGetTransformClip` function is described on page 6-43. The `GXSetTransformClip` function is described on page 6-44.

## Moving, Scaling, Rotating, and Skewing Shapes

The mapping property of transform objects allows you to perform sophisticated transformations to your shape's geometries. By altering the values of a transform's mapping, you can move, scale, rotate, skew and create perspective effects on any shapes the transform applies to. However, determining the specific changes to the mapping matrix needed to achieve a desired transformational effect can involve complex calculations. As a convenience, QuickDraw GX provides several functions that perform the calculations necessary to achieve common transformations, without you having to know how the mapping matrix is altered.

The transformational functions that QuickDraw GX provides allow you to position, rotate, scale, and skew shapes. QuickDraw GX provides two kinds of such functions, one kind that operates on transform mappings, which is of the form `GXActionTransform`, and one kind that normally operates on shape geometries, which is of the form `GXActionShape`. If you use a function that operates on a transform's mapping, the mapping is changed and all shapes that refer to the transform are affected. If you use a function that normally operates on a shape geometry, there are two possible results:

- If the shape's `gxMapTransformShape` attribute is cleared, the shape's geometry is changed, as expected. Its transform mapping is unaffected.
- If the shape's `gxMapTransformShape` attribute is set, the function works exactly like a `GXActionTransform` function, changing the transform mapping instead of the shape geometry. An additional side effect is that, if the shape's transform object is shared with other shapes, QuickDraw GX creates a copy of the transform and modifies the copy, to avoid affecting the other shapes.

## Transform Objects

If you move a shape to an absolute location, the location applies to a specific anchor point in the shape's geometry and all other points in the geometry move in relation to this point. The point used depends on the kind of shape:

- For points, lines, and curves, the anchor point is the first point in the shape's geometry.
- For rectangles, polygons, paths, and bitmaps, the anchor point is the top-left corner of the bounding rectangle.
- For text, glyph, and layout shapes, the anchor point is the origin of the first glyph.
- Other shapes (empty shapes, full shapes, and pictures) cannot be moved.

However, remember that if the shape's `gxMapTransformShape` attribute is set, calling a function that moves the shape has no effect on the geometry; it modifies the transform mapping instead. In that case, moving the shape to an absolute location means only that its transform mapping adds that location to whatever location the geometry already specifies.

## Modifying the Transform Mapping

---

One way to transform a shape is by altering its transform object's mapping property. This section shows several examples of that kind of transformation.

For example, you can move a shape to a relative or absolute location by modifying its transform. The `GXMoveTransform` function modifies the transform's mapping to move a shape a specified distance from its current location in local coordinates. The `GXMoveTransformTo` function modifies the transform's mapping to move a shape to an absolute location in local coordinate space.

The following example causes all shapes associated with the `myTransform` transform object to move to the upper-left corner of the bounding rectangle, `bounds`, of the rectangle `aRectangle`:

```
gxRectangle aRectangle, bounds;
.
.  /* initialize the rectangle (not shown) */
.
GXGetShapeBounds(aRectangle, 0, &bounds);
GXMoveTransformTo(myTransform, bounds.left, bounds.top);
```

You can also modify a transform's mapping to rotate, scale, or skew a shape around a specified point. Listing 6-3 rotates a shape's transform mapping 90 degrees, and scales and skews the mapping. The shape's center is used as the point around which to rotate, scale, and skew the transform.

**Listing 6-3** Modifying a shape's transform with transform-mapping calls only

```

Fixed          hScale, vScale, xSkew, ySkew;
gxPoint        center;
gxShape        aShape;
gxTransform    myTransform;

.
.  /* initialize the shape and the rotate/scale/skew parameters */
.
/* find the shape's center */
GXGetShapeCenter(aShape, 0, &center);

/* get the transform, rotate it around shape's center */
myTransform = GXGetShapeTransform(aShape);
GXRotateTransform(myTransform, ff(90), center.x, center.y);

/* scale and skew the shape */
GXScaleTransform(myTransform, hScale, vScale, center.x, center.y);
GXSkewTransform(myTransform, xSkew, ySkew, center.x, center.y);

```

Listing 6-4 performs the same actions as Listing 6-3: rotating, scaling, and skewing a shape's transform mapping. Like Listing 6-3, this code also affects only the transform mapping associated with the shape. This is despite the fact that it makes some calls (GXScaleShape and GXSkewShape) that would normally affect the shape's geometry. Because the shape's gxMapTransformShape attribute is set before the geometry-altering calls are made, those functions are forced to affect the transform mapping instead of the shape's geometry.

**Listing 6-4** Modifying a shape's transform with transform-mapping and shape-geometry calls

```

Fixed          hScale, vScale, xSkew, ySkew;
gxPoint        center;
gxShape        aShape;
gxTransform    myTransform;

.
.  /* initialize the shape and the rotate/scale/skew parameters */
.
/* find the shape's center */
GXGetShapeCenter(aShape, 0, &center);

/* get the transform, rotate it around shape's center */
myTransform = GXGetShapeTransform(aShape);
GXRotateTransform(myTransform, ff(90), center.x, center.y);

```

## Transform Objects

```

/* set the gxMapTransformShape attribute */
GXSetShapeAttributes(aShape,
    GXGetShapeAttributes(aShape) | gxMapTransformShape);

/* scale and skew the shape (but it affects mapping instead) */
GXScaleShape(aShape, hScale, vScale, center.x, center.y);
GXSkewShape(aShape, xSkew, ySkew, center.x, center.y);

```

You can also perform these transformations, as well as perspective-modifying operations, by directly manipulating the matrix elements of a transform's mapping. You can use the functions `GXGetTransformMapping` or `GXGetShapeMapping` to obtain the mapping matrix, then modify the matrix as desired and reassign it to the transform with `GXSetTransformMapping` or `GXSetShapeMapping`. You can also create your own mapping matrix, and then multiply it (concatenate it) with the existing mapping of a transform object, using the functions `GXMapTransform` (or `GXMapShape`, if the shape's `gxMapTransformShape` attribute is set). For more information about matrix manipulation, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

The `GXGetTransformMapping` function is described on page 6-54; the

`GXSetTransformMapping` function is described on page 6-55.

The `GXGetShapeMapping` function is described on page 6-56; the

`GXSetShapeMapping` function is described on page 6-57.

The `GXMapTransform` function is described on page 6-64; the `GXMapShape` function is described on page 6-72.

The `GXMoveTransform` function is described on page 6-58. The `GXMoveTransformTo` function is described on page 6-59. The `GXScaleTransform` function is described on page 6-60. The `GXRotateTransform` function is described on page 6-62. The `GXSkewTransform` function is described on page 6-63.

## Modifying Shape Geometry

---

A second way to transform a shape is by altering its geometry property. This section shows several examples of that kind of transformation.

You can move a shape to a relative or absolute location by modifying the shape's geometry instead of its transform mapping. The `GXMoveShape` function modifies the geometry to move a shape a specified distance from its current location in local coordinates. The `GXMoveShapeTo` function modifies the geometry to move a shape to an absolute location in local coordinate space. In either case, the geometry is altered only if the shape's `gxMapTransformShape` attribute is cleared; otherwise, the functions work just like `GXMoveTransform` and `GXMoveTransformTo`, and alter the mapping of the transform object attached to the shape.

## Transform Objects

The following example causes the shape `myShape` to move to the upper-left corner of the bounding rectangle, `bounds`, of the rectangle `aRectangle`:

```
gxRectangle aRectangle, bounds;
.
. /* initialize the rectangle (not shown) */
.
GXGetShapeBounds(aRectangle, 0, &bounds);
GXMoveShapeTo(myShape, bounds.left, bounds.top);
```

Listing 6-5 performs the same actions as Listing 6-3 and Listing 6-4: rotating, scaling, and skewing a shape. However, unlike either previous listing, this code alters the geometry of the shape itself. To ensure that the operations do not affect the shape's transform mapping, the code clears the shape's `gxMapTransformShape` attribute before making the geometry-altering calls.

**Listing 6-5** Modifying a shape's geometry with shape-geometry calls

```
Fixed          hScale, vScale, xSkew, ySkew;
gxPoint        center;
gxShape        aShape;
.
. /* initialize the shape and the rotate/scale/skew parameters */
.
/* find the shape's center */
GXGetShapeCenter(aShape, 0, &center);

/* clear the gxMapTransformShape attribute */
GXSetShapeAttributes(aShape, gxNoAttributes);

/* rotate shape around its center (affects geometry this time) */
GXRotateShape(myShape, ff(90), center.x, center.y);

/* scale and skew the shape (affects geometry this time) */
GXScaleShape(aShape, hScale, vScale, center.x, center.y);
GXSkewShape(aShape, xSkew, ySkew, center.x, center.y);
```

**Note**

Rotation of a shape's geometry can change the shape's type. For example, a rectangle may turn into a polygon when rotated. For more information, see the geometric shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*. ♦

## Transform Objects

You can also perform these operations, as well as perspective-modifying operations, by applying a mapping directly to a shape's geometry. You can create your own mapping matrix, and then apply it to a shape object using the `GXMapShape` function if the shape's `gxMapTransformShape` attribute is cleared; if the attribute is set, this function affects the shape's transform mapping instead. For more information about matrix manipulation, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

The `GXMapShape` function is described on page 6-72.

The `GXMoveShape` function is described on page 6-66. The `GXMoveShapeTo` function is described on page 6-67. The `GXScaleShape` function is described on page 6-68. The `GXRotateShape` function is described on page 6-70. The `GXSkewShape` function is described on page 6-71.

## Manipulating the View Port List

---

The view port list property of the transform object specifies all the view ports to which shapes associated with the transform are drawn. QuickDraw GX provides a pair of functions (`GXGetTransformViewPorts` and `GXSetTransformViewPorts`) that get and set the view port list of a specified transform, and another pair (`GXGetShapeViewPorts` and `GXSetShapeViewPorts`) that get and set the view port list of the transform associated with a specified shape. View ports are described in the chapter “View-Related Objects” in this book.

When you create a window, you create one or more view ports. If you want the shapes that you subsequently create to be drawn in that window, you place references to one or more of those view ports in the view port list of the shapes' transform object.

You may also want to alter a view port list of an existing transform object. For example, you might temporarily create a pane or a separate window that shows a zoomed view of a currently displayed shape. As another example, you might want to draw an object both onscreen and offscreen simultaneously.

Listing 6-6 is a partial listing of a function that adds a new view port (`newPort`) to the view port list of the transform object `myTransform`. The function calls the `GXGetTransformViewPorts` function twice, first to determine the number of view ports already in the list in order to allocate memory for it, and second to retrieve the list itself. Before adding the new view port, the function first checks the list and, if the view port is already in the list, does not add it. The function assigns the new view port to the last element in the list, and then calls `GXSetTransformViewPorts` to reassign the list to the transform. Finally, the code disposes of the view port list.



## Transform Objects

**Listing 6-6** Getting and setting view ports

---

```

gxViewPort      *ports, *portPtr;
gxViewPort      newPort;
short           portCount, count;
gxTransform     myTransform;
.
.  /* get the transform, set up the new view port (not shown) */
.
/* first, call to see how big the current view port list is */
portCount = GXGetTransformViewPorts(myTransform, nil);

/* accounting for new view port, allocate memory for the list */
portCount++;
ports = (gxViewPort *) NewPtr(portCount * sizeof(gxViewPort));

/* get the current list into memory */
GXGetTransformViewPorts(myTransform, ports);

/* check if the view port is already in the list */
portPtr = ports;
count = portCount;
while (--count > 0)
{
    /* if port is already in transform, release memory and leave */
    if (*portPtr++ == newPort)
    {
        DisposPtr((void *) ports);
        return;
    }
}
/* put view port in transform */
*portPtr = newPort;
GXSetTransformViewPorts(myTransform, portCount, ports);

/* clean up and leave */
DisposePtr((void *)ports);
return;

```

## Transform Objects

The `GXGetTransformViewPorts` function is described on page 6-73;  
 the `GXSetTransformViewPorts` function is described on page 6-74. The  
`GXGetShapeViewPorts` function is described on page 6-75;  
 the `GXSetShapeViewPorts` function is described on page 6-76.

## Setting Up Hit-Test Parameters

---

QuickDraw GX provides a pair of functions (`GXGetTransformHitTest`, `GXSetTransformHitTest`) that get and set the hit-test parameters of a specified transform, and another pair (`GXGetShapeHitTest`, `GXSetShapeHitTest`) that get and set the hit-test parameters of the transform associated with a specified shape.

The hit-test parameters are used by the functions `GXHitTestShape` and `GXHitTestPicture`. Before calling either function, you set up the shape-parts mask and define a tolerance, and assign them both to the transform object of the shape you are going to hit-test. The shape-parts mask consists of values from the `gxShapeParts` enumeration; see Table 6-1 on page 6-12 for descriptions of the individual values.

The `GXHitTestShape` and `GXHitTestPicture` functions return, in addition to an indication of which shape parts were hit during a hit-test, a distance from the hit point to one of the hit parts. If only one shape part was hit, the distance is the distance from the hit point to the nearest point on the hit part. But if more than one part was hit (for example, if a hit corresponded to both the bounding rectangle and the shape geometry), the distance returned is the distance to the first shape part—in order of processing by the function—that was hit. The order in which shape parts are processed is the order in which they appear in the `gxShapeParts` enumeration. Thus, if both bounding rectangle and geometry are tested for, and if both are hit, the distance returned is the distance to the bounding rectangle. You can use the processing order to set up the shape-parts mask to make sure that `GXHitTestShape` and `GXHitTestPicture` return the exact distance information you need.

The following example sets the shape-parts mask (`mask`) to include both the geometry and the corner points of the shape `aShape`. It also sets the tolerance to 0, meaning that if a hit point is any distance outside of the shape geometry or corner points, it is not considered a hit.

```
gxShapePart    mask = gxGeometryPart | gxCornerPointPart;
GXSetShapeHitTest(aShape, mask, 0);
```

## Transform Objects

For more information about shape parts and tolerance as a transform object property, see the section “Hit-Test Parameters” on page 6-11. For information about hit-testing with `GXHitTestShape`, see the chapter “Shape Objects” in this book. For information about hit-testing with `GXHitTestPicture`, see the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

The `GXGetTransformHitTest` function is described on page 6-78; the `GXSetTransformHitTest` function is described on page 6-79. The `GXGetShapeHitTest` function is described on page 6-80; the `GXSetShapeHitTest` function is described on page 6-81.

## Transform Objects Reference

---

This section provides reference information to the constants, data types, and functions that allow you to create and manipulate transform objects and alter their properties. It includes

- definitions of the constants and data types, including enumerations, that are specific to transform objects
- descriptions of the QuickDraw GX functions that operate on transform objects
- descriptions of the QuickDraw GX transformation functions that operate either on shape geometries or on transform mappings, depending on the state of the `gxMapTransformShape` attribute

### Constants and Data Types

---

This section describes the data types that you use to obtain and provide information about transform objects.

### The Transform Object

---

QuickDraw GX provides you with access to an individual transform object through a transform reference:

```
typedef struct gxPrivateTransformRecord *gxTransform;
```

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

## Shape Parts for Hit-Testing

---

Each transform object specifies the parts of a shape on which hit-testing is performed. The choices are specified by the `gxShapeParts` enumeration. For determining distance to a hit part, the hit-testing functions evaluate shape parts in the order shown in the enumeration.

```
enum gxShapeParts {          /* (in order of evaluation) */
    gxNoPart                 = 0,
    gxBoundsPart             = 0x0001,
    gxGeometryPart           = 0x0002,
    gxPenPart                 = 0x0004,
    gxCornerPointPart        = 0x0008,
    gxControlPointPart       = 0x0010,
    gxEdgePart               = 0x0020,
    gxJoinPart               = 0x0040,
    gxStartCapPart           = 0x0080,
    gxEndCapPart             = 0x0100,
    gxDashPart               = 0x0200,
    gxPatternPart            = 0x0400,
    gxGlyphBoundsPart        = gxJoinPart,
    gxGlyphFirstPart         = gxStartCapPart,
    gxGlyphLastPart          = gxEndCapPart,
    gxSideBearingPart        = gxDashPart,
    gxAnyPart                 = gxBoundsPart | gxGeometryPart |
        gxPenPart | gxCornerPointPart | gxControlPointPart |
        gxEdgePart | gxJoinPart | gxStartCapPart |
        gxEndCapPart | gxDashPart | gxPatternPart
};
```

```
typedef long gxShapePart;
```

The individual shape parts are described in Table 6-1 on page 6-12.

## Functions

---

This section describes the QuickDraw GX functions you can use to

- create and manipulate a transform object
- manipulate the common object properties of a transform object
- get and set the clip shape of a transform object
- perform geometric operations on a transform clip
- get and set the mapping matrix of a transform object

## Transform Objects

- apply transformation operations to a transform's mapping
- apply transformation operations directly to a shape's geometry
- get and set the view port list of a transform object
- get and set the hit-test parameters of a transform object

## Creating and Manipulating Transform Objects

---

This section describes the functions that manipulate transforms as objects in memory. With the functions in this section, you can create and dispose of a transform object, and copy, compare, and clone transform objects.

To associate a transform object with a QuickDraw GX shape, use the `GXGetShapeTransform` and `GXSetShapeTransform` functions, described in the chapter “Shape Objects” in this book.

### ***GXNewTransform***

---

You can use the `GXNewTransform` function to create a new transform object with default properties.

```
gxTransform GXNewTransform(void);
```

*function result* A reference to the newly created transform object.

#### **DESCRIPTION**

The `GXNewTransform` function creates a transform object with an owner count of 1. All other properties of the transform are set to their default values:

- A clip that is a full shape.
- A mapping that is the identity mapping.
- A shape-parts mask specifying `gxBoundsPart` only, and a tolerance of 0.
- A view port list containing a single view port covering all onscreen view devices.
- The owner count is 1.
- The tag list is empty.

#### **SPECIAL CONSIDERATIONS**

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

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory

## SEE ALSO

For an example of the use of this function, see Listing 6-1 on page 6-16.

Default transform properties are described in the section “Default Transform Objects” beginning on page 6-14. For general information on the properties of transform objects, see “Transform Object Properties” beginning on page 6-6.

To dispose of a transform object, use the `GXDisposeTransform` function, which is described in the next section.

To create a transform object that is identical to an existing one, use the `GXCopyToTransform` function, described on page 6-35.

***GXDisposeTransform***

---

You can use the `GXDisposeTransform` function to release a reference to a transform object.

```
void GXDisposeTransform(gxTransform target);
```

target      The transform object to dispose of.

## DESCRIPTION

The `GXDisposeTransform` function decrements the owner count of the transform object specified by the `target` parameter, and releases any memory used by the transform if the owner count goes to 0.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

transform\_is\_nil

**Warnings**

cannot\_dispose\_default\_transform      (debugging version)

**SEE ALSO**

For an example of the use of this function, see Listing 6-1 on page 6-16.

Owner counts for transform objects are discussed in the section “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16, and in the section “Manipulating a Transform Object’s Owner Count” beginning on page 6-19.

To examine the owner count of a transform, use the `GXGetTransformOwners` function, described on page 6-39. To increment the owner count of a transform object, use the `GXCloneTransform` function, which is described on page 6-37.

## ***GXCopyToTransform***

---

You can use the `GXCopyToTransform` function to create a copy of an existing transform object.

```
gxTransform GXCopyToTransform (gxTransform target,
                               gxTransform source);
```

**target**        A reference to the transform object to copy the source contents into. If you specify `nil` for this parameter, the `GXCopyToTransform` function creates a new transform object.

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

**DESCRIPTION**

The `GXCopyToTransform` function copies the contents of an existing transform object to another, or it creates a new transform object and copies the contents of an existing transform object to it. The function copies the clip, mapping, hit-test parameters, tag list and view port list (but not the owner count) of the source transform object into the target transform object. It clones, but does not copy, the tag objects in the tag list.

If you specify `nil` for the `target` parameter, the `GXCopyToTransform` function creates a new transform object and copies the properties of the source transform, including the tag list, to the new transform.

You can use this function to create a copy of a transform object and then modify it without changing the original.

**SPECIAL CONSIDERATIONS**

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

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
transform\_is\_nil

## SEE ALSO

For an example of the use of this function, see page 6-17.

To create a new transform object with default values, use the `GXNewTransform` function, described on page 6-33.

To compare transform objects for equality, use the `GXEqualTransform` function, described in the next section.

***GXEqualTransform***

---

You can use the `GXEqualTransform` function to determine if two transform objects are equal.

```
boolean GXEqualTransform(gxTransform one, gxTransform two);
```

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

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

*function result*   true if the transform specified by the `one` parameter is equal to the transform specified by the `two` parameter; otherwise false.

## DESCRIPTION

The `GXEqualTransform` function determines whether the transform object referenced by the `one` parameter is equal to the transform object referenced by the `two` parameter.

For two transform objects to be equal, they must have identical clip shape geometries, mappings, hit-test parameters, and view port lists. Their owner count and tag list need not be identical.

## SPECIAL CONSIDERATIONS

Note that for two clips to be identical means more than having identical dimensions. For example, a polygon clip might have the same dimensions as a path or rectangle, but shapes with different shape types are never identical. You can call the `GXSimplifyShape` function to convert the clips to their simplest form.



## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
transform\_is\_nil

## SEE ALSO

To make a copy of a transform object that is equal by the criteria of this function, use the `GXCopyToTransform` function, described on page 6-35.

The `GXSimplifyShape` function is described in the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

***GXCloneTransform***

---

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

```
gxTransform GXCloneTransform(gxTransform source);
```

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

*function result*   A copy of the reference to the source transform object.

## DESCRIPTION

The `GXCloneTransform` function increments the owner count of the transform referenced by the `source` parameter. You typically use this function when you want to create another reference to an existing transform rather than creating a distinct copy of the transform.

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

## ERRORS, WARNINGS, AND NOTICES

**Errors**

transform\_is\_nil

## Transform Objects

**SEE ALSO**

For an example of the use of this function, see Listing 6-2 on page 6-17.

Owner counts for transform objects are discussed in the section “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16, and in the section “Manipulating a Transform Object’s Owner Count” beginning on page 6-19.

To examine the owner count of a transform, use the `GXGetTransformOwners` function, described on page 6-39. To decrement the owner count of a transform, use the `GXDisposeTransform` function, described on page 6-34.

## Manipulating Transform Object Properties

---

This section describes the functions that manipulate the object properties of transforms. The functions described in this section allow you to

- reset a transform object’s properties to their default values
- manipulate the common object properties of a transform: owner count and tag list

### ***GXResetTransform***

---

You can use the `GXResetTransform` function to reset the properties of a transform object to their default values.

```
void GXResetTransform(gxTransform target);
```

`target`      A reference to the transform object whose properties you want to reset.

**DESCRIPTION**

The `GXResetTransform` function resets the following properties of the target transform to the following default values:

- The clip is a full shape.
- The mapping is the identity mapping.
- The shape-parts mask specifies `gxBoundsPart`, and the tolerance is 0.
- The view port list contains a single view port covering all screen view devices.

This function does not change the target transform’s owner count or tag list.

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out\_of\_memory  
transform\_is\_nil

## SEE ALSO

Default transform properties are described in the section “Default Transform Objects” beginning on page 6-14. For general information on the properties of transform objects, see “Transform Object Properties” beginning on page 6-6.

***GXGetTransformOwners***

---

You can use the `GXGetTransformOwners` function to determine the number of references to a particular transform object.

```
long GXGetTransformOwners(gxTransform source);
```

`source`      A reference to the transform object whose owner count you want to find.

*function result*   The owner count of the source transform object.

## DESCRIPTION

The `GXGetTransformOwners` function returns a value indicating the number of current references to the source shape.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

transform\_is\_nil

## SEE ALSO

Owner counts for transform objects are discussed in the section “Copying, Comparing, and Cloning Transform Objects” beginning on page 6-16, and in the section “Manipulating a Transform Object’s Owner Count” beginning on page 6-19.

To increment the owner count of a transform object, use the `GXCloneTransform` function, described on page 6-37. To release a reference to a transform object, use the `GXDisposeTransform` function, described on page 6-34.

## *GXGetTransformTags*

---

You can use the `GXGetTransformTags` function to examine one or more of the tag objects associated with a transform object.

```
long GXGetTransformTags(gxTransform source, long tagType,
                        long index, long count, gxTag items[]);
```

<code>source</code>	A reference to the transform object whose tag list you want to examine.
<code>tagType</code>	The type of tag object to search for. A value of 0 indicates that you want to look for all tag types.
<code>index</code>	The (1-based) index of the first such tag reference to return.
<code>count</code>	The number of tag references to return.
<code>items</code>	An array to hold the returned tag references.

*function result* The number of tag references found that match the criteria specified by the input parameters.

### **DESCRIPTION**

The `GXGetTransformTags` function searches the tag list of the source transform object for references to tag objects with the tag type specified by the `tagType` parameter. If you specify 0 for this parameter, the function searches for all tag types.

You can use the `index` and the `count` parameters to specify which tag references of the appropriate type the `GXGetInkTags` function should return. The `index` parameter indicates the first tag reference to return and the `count` parameter indicates how many tag references to return. The `index` parameter must be greater than 0. The `count` parameter must be greater than 0 or equal to the `gxSelectToEnd` constant (-1), which indicates that all tag references (starting with the tag reference indicated by the `index` parameter) should be returned.

The function result is the number of tag references found that fit the criteria. If you pass a value other than `nil` for the `items` parameter, the `GXGetInkTags` function returns in the `items` parameter the tag references that were found.

Typically, you call this function once with a `nil` value for the `items` parameter to determine the number of matching tags. Then you allocate an appropriately sized tag reference array and call the function a second time to obtain references to the matching tags.

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

```

out_of_memory
transform_is_nil
index_is_less_than_one    (debugging version)
count_is_less_than_one    (debugging version)

```

**Warnings**

```

index_out_of_range
count_out_of_range

```

## SEE ALSO

Tag objects are discussed in the chapter “Tag Objects” in this book.

To change the set of tags associated with a transform, use the `GXSetTransformTags` function, described next.

***GXSetTransformTags***

---

You can use the `GXSetTransformTags` function to add, remove, or replace tag objects associated with a transform object.

```

void GXSetTransformTags(gxTransform target, long tagType,
                        long index, long oldCount, long newCount,
                        const gxTag items[]);

```

<code>target</code>	A reference to the transform object whose tag list you want to alter.
<code>tagType</code>	The type of tag objects to replace. A value of 0 indicates that you want to replace tags of all types.
<code>index</code>	The (1-based) index of the first tag reference (of the appropriate type) to replace.
<code>oldCount</code>	The number of tag references to replace. A value of 0 specifies that you want to insert tag references before the tag reference indicated by the <code>index</code> parameter, rather than replace tag references. A value of -1 (the <code>gxSelectToEnd</code> constant) indicates that all tag references of the requested type (starting with the reference indicated by the <code>index</code> parameter), should be replaced.
<code>newCount</code>	The number of tag references to insert. A value of 0 indicates that there are no tag references to insert; the existing tag references that match the selection criteria are removed from the target transform’s tag list and disposed of.
<code>items</code>	An array of tag references to insert into the transform’s tag list.

## Transform Objects

**DESCRIPTION**

The `GXSetTransformTags` function allows you add tag references to a transform object's tag list, to remove tag references from the list, or to replace tag references in the list with new tag references. In any of these three cases, the `target` parameter specifies the ink object to be modified, the `newCount` parameter specifies the number of tag references to add, and the `items` parameter provides the new tag references.

- To add tag references, set the `oldCount` parameter to 0. Use the `tagType` and the `index` parameters to specify where to add the new tag references. (For example, if you specify `nil` for the `tagType` parameter and 1 for the `index` parameter, this function inserts the new tag references before the current tag references. If you specify a value other than `nil` for the `tagType` parameter and a value of 2 for the `index` parameter, the function inserts the new tag references before the second tag reference with a tag type matching the `tagType` parameter.)
- To remove tag references, set the `newCount` parameter to 0 and the `items` parameter to `nil`. You can use the `index` and the `oldCount` parameters to specify which tag references (of the specified type) should be removed. The `index` parameter indicates the first tag reference (of the specified type) to remove and the `oldCount` parameter indicates how many tag references (of the specified type) to remove.
- To replace tag references, use the `tagType`, `index`, and `oldCount` parameters to indicate which tag references to replace, and use the `newCount` and `items` parameters to specify the new tag references to add. If `newCount` is greater than `oldCount`, the extra tag references are placed immediately adjacent to the last tag reference replaced.

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

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>tag_is_nil</code>	
<code>parameter_is_nil</code>	(debugging version)
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_zero</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)

**Warnings**

<code>index_out_of_range</code>
<code>count_out_of_range</code>

**Notices (debugging version)**

<code>tag_already_set</code>
------------------------------

**SEE ALSO**

Tag objects are discussed in the chapter “Tag Objects” in this book.

To examine the set of tags associated with a transform, use the `GXGetTransformTags` function, described in the previous section.

## Getting and Setting the Clip

---

This section describes the functions that allow you to manipulate the clip property of a transform object. The clip property is a reference to a shape object. See “Clip” on page 6-7 for more information.

### ***GXGetTransformClip***

---

You can use the `GXGetTransformClip` function to retrieve the clip property of a transform object.

```
gxShape GXGetTransformClip(gxTransform source);
```

**source**            A reference to the transform object whose clip shape you want to determine.

*function result*   A reference to a newly created shape object that is a copy of the source transform’s clip.

#### **DESCRIPTION**

The `GXGetTransformClip` function creates a new shape object, copies into it the geometry of the shape referenced in the clip property of the source transform, and returns a reference to the new shape as the function result.

Note that the returned shape object is a copy; you can alter it without affecting the clip property of the source transform. If you call this function and alter the clip shape it returns, you can then assign that changed clip shape back to the transform object by calling the `GXSetTransformClip` function.

#### **SPECIAL CONSIDERATIONS**

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

#### **ERRORS, WARNINGS, AND NOTICES**

##### **Errors**

`out_of_memory`  
`transform_is_nil`  
`parameter_is_nil`      (debugging version)

## Transform Objects

## SEE ALSO

For information about the clip property of transform objects, see “Clip” on page 6-7.

To change the information in the clip property of a transform object, use the `GXSetTransformClip` function, described next.

If you want to manipulate the clip property of a transform associated with a specific shape, you can use the `GXGetShapeClip` function, described on page 6-45, or the `GXSetShapeClip` function, described on page 6-46.

## ***GXSetTransformClip***

---

You can use the `GXSetTransformClip` function to change a transform object’s clip property.

```
void GXSetTransformClip(gxTransform target, gxShape clip);
```

`target`        A reference to the transform object whose clip shape you want to change.

`clip`         A reference to a shape object containing the new clip shape information.

## DESCRIPTION

The `GXSetTransformClip` function copies information from the shape object referenced by the `clip` parameter into the clip property of the transform object referenced by the `target` parameter. You can specify `nil` for the `clip` parameter, in which case this function sets the clip property of the target transform to a full clip (no transform clipping takes place).

The new clip shape, which you specify using the `clip` parameter, may be a geometric shape, a bitmap shape, or a glyph shape. It may not be a picture, text, or layout shape.

- If you specify a geometric shape, it must be in primitive form—that is, all the stylistic information about the shape must be incorporated into the shape’s geometry—because this function copies only the geometry-related information from the shape you specify. It does not copy the information contained in the shape’s style. You can convert a shape to its primitive form using the `GXPrimitiveShape` function, which is described in *Inside Macintosh: QuickDraw GX Graphics*. You can also specify an empty or full shape for a clip.
- If you specify a bitmap shape, it must have a pixel size of 1 and its color profile reference must be `nil`. In the bitmap, pixel values of 0 obscure drawing; pixel values of 1 do not restrict visibility.
- If you specify a glyph shape, this function uses information from the glyph shape’s style object as well as its style list to determine the size, form, and position of the glyph outlines; those outlines are then used to clip drawing. The style list cannot have `nil` entries. A style object referenced by the glyph shape cannot be complex—that is, it cannot have a cap, join, dash, pattern, text face, font variation, tag list, or any of the properties used only by layout shapes.



## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out_of_memory	
transform_is_nil	
shape_is_nil	
shapeFill_not_allowed	(debugging version)
colorProfile_must_be_nil	(debugging version)
bitmap_pixel_size_must_be_1	(debugging version)
empty_shape_not_allowed	(debugging version)
ignorePlatformShape_not_allowed	(debugging version)
nil_style_in_glyph_not_allowed	(debugging version)
complex_glyph_style_not_allowed	(debugging version)
illegal_type_for_shape	(debugging version)

**Warnings**

tags_in_shape_ignored	(debugging version)
-----------------------	---------------------

**Notices (debugging version)**

clip_already_set	
------------------	--

## SEE ALSO

For an example of the use of this function, see page 6-23.

For information about the clip property of transform objects, see “Clip” on page 6-7.

For information about primitive shapes, geometric shapes and bitmap shapes, see *Inside Macintosh: QuickDraw GX Graphics*. For information about glyph shapes, see *Inside Macintosh: QuickDraw GX Typography*.

To examine the clip shape of a transform object, use the `GXGetTransformClip` function, described in the previous section.

If you want to manipulate the clip property of a transform associated with a specific shape, you can use the `GXGetShapeClip` function, described next, or the `GXSetShapeClip` function, described on page 6-46.

***GXGetShapeClip***

You can use the `GXGetShapeClip` function to retrieve the clip property of a transform object associated with a specified shape.

```
gxShape GXGetShapeClip(gxShape source);
```

**source**      A reference to the shape whose transform object you want to examine the clip property of.

**function result**   A reference to a newly created shape encapsulating information copied from the clip property of the source shape’s transform object.

## Transform Objects

**DESCRIPTION**

The `GXGetShapeClip` function creates a new shape object, copies into it geometry information from the clip property of the source shape's transform object, and returns a reference to the new shape as the function result.

Note that the returned shape object is a copy; you can alter it without affecting the clip property of the source shape's transform. If you call this function and alter the clip shape it returns, you can then assign that changed clip shape back to the shape's transform object by calling the `GXSetShapeClip` function.

**SPECIAL CONSIDERATIONS**

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

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

`out_of_memory`  
`shape_is_nil`

**SEE ALSO**

For information about the clip property of transform objects, see "Clip" on page 6-7.

To alter the clip property of a transform object associated with a particular shape, use the `GXSetShapeClip` function, described in the next section.

If you want to manipulate the clip property of a particular transform object, you can use the `GXGetTransformClip` function, described on page 6-43, or the `GXSetTransformClip` function, described on page 6-44.

***GXSetShapeClip***

---

You can use the `GXSetShapeClip` function to change the clip property of a transform object associated with a specified shape.

```
void GXSetShapeClip(gxShape target, gxShape clip);
```

**target**      A reference to the shape whose transform object you want to change the clip shape of.

**clip**        A reference to a shape object containing the new clip shape information.

## Transform Objects

## DESCRIPTION

The `GXSetShapeClip` function copies information from the shape object referenced by the `clip` parameter into the `clip` property of the transform object associated with the shape referenced by the `target` parameter.

Calling this function is almost equivalent to

```
GXSetTransformClip(GXGetShapeTransform(myShape), theClip);
```

except that, if the source shape's transform object is shared with other shapes, `GXSetShapeClip` creates a new copy of the transform object, attaches it to the source shape, and changes the `clip` of the copy. That way, calling this function does not produce side effects on other shapes.

You can specify `nil` for the `clip` parameter, in which case this function sets the `clip` property of the target shape's transform to a full clip (no transform clipping takes place).

The new clip shape, which you specify using the `clip` parameter, may be a geometric shape, a bitmap shape, or a glyph shape. It may not be a picture, text, or layout shape.

- If you specify a geometric shape, it must be in primitive form—that is, all the stylistic information about the shape must be incorporated into the shape's geometry—because this function copies only the geometry-related information from the shape you specify. It does not copy the information contained in the shape's style. You can convert a shape to its primitive form using the `GXPrimitiveShape` function, which is described in *Inside Macintosh: QuickDraw GX Graphics*.
- If you specify a bitmap shape, it must have a pixel size of 1 and its color profile reference must be `nil`. In the bitmap, pixel values of 0 obscure drawing; pixel values of 1 do not restrict visibility.
- If you specify a glyph shape, this function uses information from the glyph shape's style object as well as its style list to determine the size, form, and position of the glyph outlines; those outlines are then used to clip drawing. The style list cannot have `nil` entries. A style object referenced by the glyph shape cannot be complex—that is, it cannot have a cap, join, dash, pattern, text face, font variation, tag list, or any of the properties used only by layout shapes.

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

out_of_memory	
transform_is_nil	
shape_is_nil	
shapeFill_not_allowed	(debugging version)
colorProfile_must_be_nil	(debugging version)
bitmap_pixel_size_must_be_1	(debugging version)
empty_shape_not_allowed	(debugging version)
ignorePlatformShape_not_allowed	(debugging version)
nil_style_in_glyph_not_allowed	(debugging version)
complex_glyph_style_not_allowed	(debugging version)
illegal_type_for_shape	(debugging version)

**Warnings**

tags_in_shape_ignored	(debugging version)
-----------------------	---------------------

**Notices (debugging version)**

clip_already_set
------------------

## SEE ALSO

To retrieve the clip property of a transform object associated with a particular shape, use the `GXGetShapeClip` function, described in the previous section.

To assign a clip directly to a transform object, use the `GXSetTransformClip` function, described on page 6-44.

For information about the clip property of transform objects, see “Clip” on page 6-7.

For information about primitive shapes, geometric shapes and bitmap shapes, see *Inside Macintosh: QuickDraw GX Graphics*. For information about glyph shapes, see *Inside Macintosh: QuickDraw GX Typography*.

## Performing Geometric Operations on Transform Clips

QuickDraw GX provides a number of functions that allow you to perform constructive geometry operations on the clip shapes of transform objects. Each of these functions replaces the clip property of a transform object with the result of an operation combining the original clip geometry with the geometry of another shape. The functions are

- `GXUnionTransform`, which combines the transform clip with a shape geometry
- `GXIntersectTransform`, which intersects the transform clip with a shape geometry
- `GXDifferenceTransform`, which subtracts a shape geometry from the transform clip
- `GXReverseDifferenceTransform`, which subtracts the transform clip from a shape geometry
- The `GXExcludeTransform`, which performs an exclusive-OR operation with the transform clip and a shape geometry

## *GXUnionTransform*

---

You can use the `GXUnionTransform` function to calculate the union of the geometry of the clip shape in a transform object with the geometry of a specified shape object, and then replace the transform's clip geometry with the resulting geometry.

```
void GXUnionTransform(gxTransform target, gxShape operand);
```

<code>target</code>	A reference to the transform object containing the clip property you want to modify.
<code>operand</code>	A reference to the shape containing the geometry you want to combine with the target transform's clip.

### ERRORS, WARNINGS, AND NOTICES

#### Errors

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>shapeFill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_may_not_be_a_picture</code>	(debugging version)

#### Warnings

<code>character_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

### SEE ALSO

For an example of the use of this function, see page 6-23.

For an illustration of the effects of the `GXUnionTransform` function on the transform clip, see Figure 6-8 on page 6-22. For a general description of constructive geometry operations, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

***GXIntersectTransform***

---

You can use the `GXIntersectTransform` function to calculate the intersection of the geometry of the clip shape in a transform object with the geometry of a specified shape object, and then replace the transform's clip geometry with the resulting geometry.

```
void GXIntersectTransform(gxTransform target, gxShape operand);
```

<code>target</code>	A reference to the transform object containing the clip property you want to modify.
<code>operand</code>	A reference to the shape containing the geometry you want to intersect with the target transform's clip.

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

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>shapeFill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_may_not_be_a_picture</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

**SEE ALSO**

For an illustration of the effects of the `GXIntersectTransform` function on the transform clip, see Figure 6-8 on page 6-22. For a general description of constructive geometry operations, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

***GXDifferenceTransform***

---

You can use the `GXDifferenceTransform` function to subtract the geometry of a specified shape object from the geometry of the clip shape in a transform object, and then replace the transform's clip property with the resulting geometry.

```
void GXDifferenceTransform(gxTransform target, gxShape operand);
```

**target**            A reference to the transform object containing the clip property you want to modify.

**operand**          A reference to the shape containing the geometry you want to subtract from the target transform's clip.

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

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>shapeFill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_may_not_be_a_picture</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

**SEE ALSO**

For an illustration of the effects of the `GXDifferenceTransform` function on the transform clip, see Figure 6-8 on page 6-22. For a general description of constructive geometry operations, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

***GXReverseDifferenceTransform***

---

You can use the `GXReverseDifferenceTransform` function to subtract the geometry of the clip shape in a transform object from the geometry of a specified shape object, and then replace the transform's clip property with the resulting geometry.

```
void GXReverseDifferenceTransform(gxTransform target,
                                gxShape operand);
```

<code>target</code>	A reference to the transform object containing the clip property you want to modify.
<code>operand</code>	A reference to the shape containing the geometry from which you want to subtract the target transform's clip.

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

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>shapeFill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_may_not_be_a_picture</code>	(debugging version)

**Warnings**

<code>character_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

**SEE ALSO**

For an illustration of the effects of the `GXReverseDifferenceTransform` function on the transform clip, see Figure 6-8 on page 6-22. For a general description of constructive geometry operations, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.



## *GXExcludeTransform*

---

You can use the `GXExcludeTransform` function to perform an exclusive-OR operation between the geometry of the clip shape in a transform object and the geometry of a specified shape object, and then replace the transform's clip property with the resulting geometry.

```
void GXExcludeTransform(gxTransform target, gxShape operand);
```

**target**            A reference to the transform object containing the clip property you want to modify.

**operand**           A reference to the shape containing the geometry you want to combine with the target transform's clip in an exclusive-OR operation.

### ERRORS, WARNINGS, AND NOTICES

#### Errors

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>shape_is_nil</code>	
<code>number_of_contours_exceeds_implementation_limit</code>	
<code>number_of_points_exceeds_implementation_limit</code>	
<code>size_of_path_exceeds_implementation_limit</code>	
<code>size_of_polygon_exceeds_implementation_limit</code>	
<code>shapeFill_not_allowed</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>clip_to_frame_shape_unimplemented</code>	(debugging version)
<code>shape_may_not_be_a_bitmap</code>	(debugging version)
<code>shape_may_not_be_a_picture</code>	(debugging version)

#### Warnings

<code>character_substitution_took_place</code>	
<code>unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve</code>	(debugging version)

### SEE ALSO

For an illustration of the effects of the `GXExcludeTransform` function on the transform clip, see Figure 6-8 on page 6-22. For a general description of constructive geometry operations, see the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.

## Getting and Setting the Mapping

---

This section describes the functions you can use to manipulate a transform object's mapping matrix.

## ***GXGetTransformMapping***

---

You can use the `GXGetTransformMapping` function to retrieve the mapping property of a transform object.

```
gxMapping *GXGetTransformMapping(gxTransform source,
                                  gxMapping *map);
```

`source`      A reference to the transform object whose mapping you want to examine.

`map`          A pointer to a mapping structure. On return, the structure contains the mapping matrix of the source transform.

*function result* A pointer to the mapping property of the source transform. (This value is the same as the value returned in the `map` parameter.)

### **DESCRIPTION**

The `GXGetTransformMapping` function copies the mapping matrix information from the mapping property of the source transform object into the mapping structure pointed to by the `map` parameter. The function also returns as its function result a pointer to this mapping structure.

Note that the returned mapping is a copy; you can alter it without affecting the mapping property of the source transform. If you call this function and alter the mapping that it returns, you can then assign that changed mapping back to the transform object by calling the `GXSetTransformMapping` function.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`out_of_memory`  
`transform_is_nil`  
`parameter_is_nil`      (debugging version)

### **SEE ALSO**

For information about the mapping property of the transform object, see the section “Mapping” beginning on page 6-10. For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## ***GXSetTransformMapping***

---

You can use the `GXSetTransformMapping` function to assign a mapping to a transform object.

```
void GXSetTransformMapping(gxTransform target,
                           const gxMapping *map);
```

**target**            A reference to the transform object you want to assign the mapping to.

**map**                A pointer to a mapping structure containing the matrix you want to assign as the mapping property of the target transform.

### ***DESCRIPTION***

The `GXSetTransformMapping` function copies information from the mapping structure pointed to by the `map` parameter into the mapping property of the transform object referenced by the `target` parameter.

You can specify `nil` for the `map` parameter, in which case this function sets the mapping property of the target transform to the identity mapping. (An identity mapping has no transforming effect on shape geometries that it is applied to.)

### ***SPECIAL CONSIDERATIONS***

You may provide any values for the elements of the mapping structure pointed to by the `map` parameter, with one exception: the lower-right element of this matrix (element `[2][2]`) may not be 0.

### ***ERRORS, WARNINGS, AND NOTICES***

#### **Errors**

`out_of_memory`  
`transform_is_nil`

#### **Notices (debugging version)**

`mapping_unaffected`

### ***SEE ALSO***

For information about the mapping property of the transform object, see the section “Mapping” beginning on page 6-10. For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## ***GXGetShapeMapping***

---

You can use the `GXGetShapeMapping` function to retrieve the mapping property of the transform object associated with a specified shape.

```
gxMapping *GXGetShapeMapping(gxShape source, gxMapping *map);
```

**source**      A reference to the shape whose transform object contains the mapping property you want to examine.

**map**          A pointer to the mapping structure. On return, the structure contains the mapping matrix of the source shape's transform.

*function result* A pointer to the mapping property of the source shape's transform. (This value is the same as the value returned in the `map` parameter.)

### **DESCRIPTION**

The `GXGetShapeMapping` function copies the mapping matrix information from the mapping property of the source shape's transform object into the mapping structure pointed to by the `map` parameter. The function also returns as its function result a pointer to this mapping structure.

Note that the returned mapping is a copy; you can alter it without affecting the mapping property of the source shape's transform. If you call this function and alter the mapping that it returns, you can then assign that changed mapping back to the shape's transform object by calling the `GXSetShapeMapping` function.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`out_of_memory`  
`shape_is_nil`  
`parameter_is_nil`      (debugging version)

### **SEE ALSO**

For information about the mapping property of the transform object, see the section "Mapping" beginning on page 6-10. For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## *GXSetShapeMapping*

---

You can use the `GXSetShapeMapping` function to assign a new mapping to the transform object associated with a specified shape.

```
void GXSetShapeMapping(gxShape target, const gxMapping *map);
```

<code>target</code>	A reference to the shape whose transform you want to assign the mapping to.
<code>map</code>	A pointer to a mapping structure containing the matrix you want to assign as the mapping property of the target shape's transform.

### *DESCRIPTION*

The `GXSetShapeMapping` function copies information from the mapping structure pointed to by the `map` parameter into the mapping property of the transform object associated with the shape referenced by the `target` parameter.

Calling this function is almost equivalent to

```
GXSetTransformMapping(GXGetShapeTransform(myShape), theMapping);
```

except that, if the source shape's transform object is shared with other shapes, `GXSetShapeMapping` creates a new copy of the transform object, attaches it to the source shape, and changes the mapping of the copy. That way, calling this function does not produce side effects on other shapes.

You can specify `nil` for the `map` parameter, in which case this function sets the mapping property of the target shape's transform to the identity matrix. (An identity mapping has no transforming effect on shape geometries that it is applied to.)

### *SPECIAL CONSIDERATIONS*

You can provide any values for the elements of the mapping structure pointed to by the `map` parameter, with one exception: the lower-right element of this matrix (element `[2][2]`) may not be 0.

### *ERRORS, WARNINGS, AND NOTICES*

#### **Errors**

`out_of_memory`  
`shape_is_nil`

#### **Notices (debugging version)**

`mapping_unaffected`

## Transform Objects

## SEE ALSO

To assign a mapping directly to a transform object, use the `GXSetTransformMapping` function, described on page 6-55.

For information about the mapping property of the transform object, see the section “Mapping” beginning on page 6-10. For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

## Transforming Shapes by Modifying Transform Mappings

---

The mapping property of transform objects allows you to perform sophisticated transformations to your shape’s geometries. To get and set the mapping property, use the functions described in the section “Getting and Setting the Mapping” beginning on page 6-53.

The functions described in this section perform the calculations necessary to achieve common matrix transformations, without you having to modify the mapping matrix directly:

- `GXMoveTransform` alters a transform’s mapping to move a shape by a specified horizontal and vertical offset.
- `GXMoveTransformTo` alters a transform’s mapping to move a shape to a specified position.
- `GXScaleTransform` alters a transform’s mapping to scale a shape by specified horizontal and vertical factors around a specified origin.
- `GXRotateTransform` alters a transform’s mapping to rotate a shape by a specified number of degrees around a specified origin.
- `GXSkewTransform` alters a transform’s mapping to skew a shape by specified horizontal and vertical factors around a specified origin.
- `GXMapTransform` concatenates (using matrix multiplication) a specified mapping matrix to the mapping matrix contained in a transform’s mapping property.

QuickDraw GX provides a corresponding set of functions that you can use to apply these common transformations directly to the geometry of a shape object, rather than to its transform mapping. They are described in the section “Transforming Shapes by Modifying Shape Geometries” beginning on page 6-65.

### ***GXMoveTransform***

---

You can use the `GXMoveTransform` function to alter the mapping property of a transform object so that it moves its associated shape by a specified horizontal and vertical distance.

```
void GXMoveTransform(gxTransform target, Fixed deltaX,
                    Fixed deltaY);
```

## Transform Objects

<code>target</code>	A reference to the transform object whose mapping property you want to alter.
<code>deltaX</code>	The horizontal distance.
<code>deltaY</code>	The vertical distance.

**DESCRIPTION**

The `GXMoveTransform` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. When applied to a shape, the new matrix performs the same mapping transformations on the shape as the original matrix, except that the new matrix also moves the shape horizontally by the distance specified in the `deltaX` parameter and vertically by the distance specified in the `deltaY` parameter.

The distances are specified in geometry space.

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

`out_of_memory`  
`transform_is_nil`

**Warnings**

`move_transform_out_of_range`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To move a shape by altering its geometry, use the `GXMoveShape` function, described on page 6-66.

***GXMoveTransformTo***

You can use the `GXMoveTransformTo` function to alter the mapping property of a transform object so that it moves its associated shape to a specified position.

```
void GXMoveTransformTo(gxTransform target, Fixed x, Fixed y);
```

<code>target</code>	A reference to the transform object whose mapping property you want to alter.
<code>x</code>	The horizontal coordinate of the desired position.
<code>y</code>	The vertical coordinate of the desired position.

## Transform Objects

**DESCRIPTION**

The `GXMoveTransformTo` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. When applied to a shape, the new mapping matrix performs the same mapping transformations on the shape as the original matrix, except that the new matrix moves the shape to the position specified by the `x` and `y` parameters.

The horizontal and vertical coordinates are specified in geometry space. However, the position they specify relates only to the translation values in the mapping itself; the values in a shape's geometry are added to these values to determine the shape's final position in local space.

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

`out_of_memory`  
`transform_is_nil`

**Warnings**

`move_transform_out_of_range`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

For an example of the use of this function, see page 6-24.

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To move a shape to a specified position by altering its geometry, use the `GXMoveShapeTo` function, described on page 6-67.

***GXScaleTransform***

---

You can use the `GXScaleTransform` function to alter the mapping property of a transform object so that it scales its associated shape by specified horizontal and vertical factors about a specified origin.

```
void GXScaleTransform(gxTransform target, Fixed hScale,
                     Fixed vScale, Fixed xOffset, Fixed yOffset);
```



## Transform Objects

<code>target</code>	A reference to the transform object whose mapping property you want to alter.
<code>hScale</code>	The horizontal scaling factor.
<code>vScale</code>	The vertical scaling factor.
<code>xOffset</code>	The horizontal coordinate of the origin to scale about.
<code>yOffset</code>	The vertical coordinate of the origin to scale about.

**DESCRIPTION**

The `GXScaleTransform` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. When applied to a shape, the new mapping matrix performs the same mapping transformations on the shape as the original matrix, but the new matrix also scales the shape horizontally by the factor indicated by the `hScale` parameter and vertically by the factor indicated by the `vScale` parameter. The new matrix scales the shape about the origin specified by the `xOffset` and `yOffset` parameters. (The origin is the point whose coordinates do not change as a result of the scaling operation.)

A value of `ff(1)` for the `hScale` or `vScale` parameter indicates no change of scale in the corresponding direction.

The coordinates of the origin are specified in local space.

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

`out_of_memory`  
`transform_is_nil`

**Warnings**

`scale_transform_out_of_range`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

For examples of the use of this function, see page 6-17 and Listing 6-3 on page 6-25.

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To scale a shape by altering its geometry, use the `GXScaleShape` function, described on page 6-68.

## ***GXRotateTransform***

---

You can use the `GXRotateTransform` function to alter the mapping property of a transform object so that it rotates its associated shape a specified number of degrees around a specified origin.

```
void GXRotateTransform(gxTransform target, Fixed degrees,
                      Fixed xOffset, Fixed yOffset);
```

<code>target</code>	A reference to the transform object whose mapping property you want to alter.
<code>degrees</code>	The amount to rotate.
<code>xOffset</code>	The horizontal coordinate of the origin to rotate around.
<code>yOffset</code>	The vertical coordinate of the origin to rotate around.

### **DESCRIPTION**

The `GXRotateTransform` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. When applied to a shape, the new mapping matrix performs the same mapping transformations on the shape as the original matrix, but the new matrix also rotates the shape by the number of degrees specified in the `degrees` parameter around the origin specified by the `xOffset` and `yOffset` parameters.

The coordinates of the origin are specified in local space.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`out_of_memory`  
`transform_is_nil`

#### **Warnings**

`rotate_transform_out_of_range`

#### **Notices (debugging version)**

`mapping_unaffected`

### **SEE ALSO**

For an example of the use of this function, see Listing 6-3 on page 6-25.

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To rotate a shape by altering its geometry, use the `GXRotateShape` function, described on page 6-70.

## *GXSkewTransform*

---

You can use the `GXSkewTransform` function to alter the mapping property of a transform object so that it skews its associated shape about a specified origin by specified horizontal and vertical factors.

```
void GXSkewTransform(gxTransform target, Fixed xSkew,
                    Fixed ySkew, Fixed xOffset, Fixed yOffset);
```

<code>target</code>	A reference to the transform object whose mapping property you want to alter.
<code>hSkew</code>	The amount to skew in the horizontal direction.
<code>vSkew</code>	The amount to skew in the vertical direction.
<code>xOffset</code>	The horizontal coordinate of the origin to skew about.
<code>yOffset</code>	The vertical coordinate of the origin to skew about.

### *DESCRIPTION*

The `GXSkewTransform` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. When applied to a shape, the new mapping matrix performs the same mapping transformations on the shape as the original matrix, but the new matrix also skews the shape in the horizontal direction by the factor indicated by the `hSkew` parameter, and in the vertical direction by the factor indicated by the `vSkew` parameter. The new matrix skews the shape about the origin specified by the `xOffset` and `yOffset` parameters. (The origin is the point whose coordinates do not change as a result of the scaling operation.)

The skew factors are expressed as a proportional amount of shift in one direction with distance in the perpendicular direction. A value of 0 for the `hSkew` or `vSkew` parameter indicates no skewing in the corresponding direction.

The coordinates of the origin are specified in local space.

### *ERRORS, WARNINGS, AND NOTICES*

#### **Errors**

`out_of_memory`  
`transform_is_nil`

#### **Warnings**

`skew_transform_out_of_range`

#### **Notices (debugging version)**

`mapping_unaffected`

## Transform Objects

**SEE ALSO**

For an example of the use of this function, see Listing 6-3 on page 6-25.

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To skew a shape by altering its geometry, use the `GXSkewShape` function, described on page 6-71.

***GXMapTransform***

---

You can use the `GXMapTransform` function to apply a separate mapping matrix to the mapping of a transform object, so that its associated shape is additionally transformed according to the specifications of the matrix.

```
void GXMapTransform(gxTransform target, const gxMapping *map);
```

**target**            A reference to the transform object whose mapping property you want to alter.

**map**                A pointer to a mapping structure containing the information you want to incorporate into the target transform's mapping.

**DESCRIPTION**

The `GXMapTransform` function calculates a new mapping matrix for the transform object referenced by the `target` parameter. It does so by concatenating the mappings (performing matrix multiplication). When applied to a shape, the transform's new mapping matrix performs the same transformations as the transform's original matrix as well as the transformations indicated by the matrix pointed to by the `map` parameter.

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

`out_of_memory`  
`transform_is_nil`

**Warnings**

`map_transform_out_of_range`

**Notices (debugging version)**

`mapping_unaffected`

## SEE ALSO

For information about mapping matrices in general, see the mathematics chapter of *Inside Macintosh: QuickDraw GX Environment and Utilities*.

To use a mapping matrix to alter a shape's geometry, use the `GXMapShape` function, described on page 6-72.

## Transforming Shapes by Modifying Shape Geometries

---

The functions described in this section perform the calculations necessary to achieve common matrix transformations of shapes. They are equivalent to the functions that modify transform mappings as described in the previous section, "Transforming Shapes by Modifying Transform Mappings" beginning on page 6-58; however, the functions in this section can perform their transformations by directly modifying a shape's geometry rather than altering its transform's mapping:

- `GXMoveShape` alters a shape's geometry to move it by a specified horizontal and vertical offset.
  - `GXMoveShapeTo` alters a shape's geometry to move it to a specified position.
  - `GXScaleShape` alters a shape's geometry to scale it by specified horizontal and vertical factors around a specified origin.
  - `GXRotateShape` alters a shape's geometry to rotate it by a specified number of degrees around a specified origin.
  - `GXSkewShape` alters a shape's geometry to skew it by specified horizontal and vertical factors around a specified origin.
  - `GXMapShape` alters a shape's geometry by applying a specified mapping matrix to it.
- (Note also that the function `GXSetShapeBounds` works in the same manner as the other functions listed here; it modifies a shape's geometry to effect a scaling operation. However, `GXSetShapeBounds` is described in the geometric operations chapter of *Inside Macintosh: QuickDraw GX Graphics*.)

When applied to a shape object, each of these functions performs its transformation in one of two ways:

- If the shape's `gxMapTransformShape` attribute is cleared, these functions apply their mapping operations directly to the points of the shape object's geometry. In this case, the shape's transform mapping is unaffected.
- If the shape's `gxMapTransformShape` attribute is set, these functions apply their mapping operations by altering the mapping property of the shape's transform object, in the manner of the transform-altering functions described in the previous section. In this case, the shape's geometry is unaffected.

## *GXMoveShape*

---

You can use the `GXMoveShape` function to move a shape by a specified horizontal and vertical distance.

```
void GXMoveShape(gxShape target, Fixed deltaX, Fixed deltaY);
```

<code>target</code>	A reference to the shape you want to move.
<code>deltaX</code>	The horizontal distance to move the shape.
<code>deltaY</code>	The vertical distance to move the shape.

### *DESCRIPTION*

The `GXMoveShape` function changes the position of the shape referenced by the `target` parameter horizontally by the distance specified in the `deltaX` parameter and vertically by the distance specified in the `deltaY` parameter.

This function moves the target shape by the specified offsets in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function recalculates the control points of the shape's geometry to effect the move.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXMoveTransform` function; it recalculates the mapping matrix of the target shape's transform object to effect the move. If the target shape shares this transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set. Also, if the shape is a text, glyph, or layout shape and it contains no characters, this function has no effect.

The distances are specified in geometry coordinates.

### *ERRORS, WARNINGS, AND NOTICES*

#### **Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>shape_access_not_allowed</code>	(debugging version)

#### **Warnings**

<code>move_shape_out_of_range</code>
<code>graphic_type_cannot_be_moved</code>

#### **Notices (debugging version)**

<code>mapping_unaffected</code>
---------------------------------

**SEE ALSO**

To move a shape by altering the mapping property of its transform object, you can also use the `GXMoveTransform` function, described on page 6-58.

***GXMoveShapeTo***

---

You can use the `GXMoveShapeTo` function to move a shape to a specified position.

```
void GXMoveShapeTo(gxShape target, Fixed x, Fixed y);
```

<code>target</code>	A reference to the shape you want to move.
<code>x</code>	The horizontal coordinate of the position to move the shape to.
<code>y</code>	The vertical coordinate of the position to move the shape to.

**DESCRIPTION**

The `GXMoveShapeTo` function moves the shape referenced by the `target` parameter to the position specified by the `x` and `y` parameters. The position corresponds to a specific point in the shape's geometry:

- For point, line, and curve shapes, the point (`x`, `y`) corresponds to the first point in the shape's geometry.
- For rectangle, polygon, path, and bitmap shapes, the point (`x`, `y`) corresponds to the top-left corner of the bounding rectangle.
- For text, glyph, and layout shapes, the point (`x`, `y`) corresponds to the origin of the first glyph. If the shape contains no characters, this function has no effect.
- Other shapes (empty shapes, full shapes, and pictures) cannot be moved.

This function relocates the target shape in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function recalculates the control points of the shape's geometry to effect the move.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXMoveTransformTo` function; it recalculates the mapping matrix of the target shape's transform object to effect the move. If the target shape shares this transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set.

The horizontal and vertical coordinates are specified in geometry space.

## Transform Objects

**SPECIAL CONSIDERATIONS**

This function does not necessarily move the target shape to the position in local space specified by the `x` and `y` parameters. Furthermore, if the shape's `gxMapTransformShape` attribute is set, this function does not necessarily move the shape to the same position it would if the `gxMapTransformShape` attribute were cleared:

- With the attribute cleared, this function modifies shape geometry so that, in geometry space, the shape is at the position specified in the `x` and `y` parameters. However, the function ignores the transform mapping, so the shape's resultant position in local space will be at (`x`, `y`) only if the transform mapping specifies no translation.
- With the attribute set, this function ignores shape geometry and sets the translation values in the shape's transform mapping to reflect the `x` and `y` parameters. Thus the shape's resultant position in local space will be at (`x`, `y`) only if its position in geometry space is at (0.0, 0.0).

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

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)

**Warnings**

`move_shape_out_of_range`  
`graphic_type_cannot_be_moved`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

For an example of the use of this function, see page 6-27.

To move a shape to a specified position by altering the mapping property of its transform object, you can also use the `GXMoveTransformTo` function, described on page 6-59.

***GXScaleShape***

---

You can use the `GXScaleShape` function to scale a shape by specified horizontal and vertical factors about a specified origin.

```
void GXScaleShape(gxShape target, Fixed hScale, Fixed vScale,
                  Fixed xOffset, Fixed yOffset);
```

`target`        A reference to the shape you want to scale.

`hScale`        The horizontal scaling factor.



## Transform Objects

<code>vScale</code>	The vertical scaling factor.
<code>xOffset</code>	The horizontal coordinate of the origin to scale the shape about.
<code>yOffset</code>	The vertical coordinate of the origin to scale the shape about.

**DESCRIPTION**

The `GXScaleShape` function scales the shape referenced by the `target` parameter horizontally by the factor specified in the `hScale` parameter and vertically by the factor specified in the `vScale` parameter. The scaling is centered about the origin specified in the `xOffset` and `yOffset` parameters. (The origin is the point whose coordinates do not change as a result of the scaling operation.)

This function scales the target shape in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function recalculates the control points of the shape's geometry to effect the scaling.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXScaleTransform` function; it recalculates the mapping matrix of the target shape's transform object to effect the scaling. If the target shape shares this transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set.

The coordinates of the origin are specified in geometry space.

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

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)

**Warnings**

`scale_shape_out_of_range`  
`graphic_type_cannot_be_moved`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

For examples of the use of this function, see Listing 6-4 on page 6-25 and Listing 6-5 on page 6-27.

To scale a shape by altering the mapping property of its transform object, you can also use the `GXScaleTransform` function, described on page 6-60.

## ***GXRotateShape***

---

You can use the `GXRotateShape` function to rotate a shape around a specified origin.

```
void GXRotateShape(gxShape target, Fixed degrees,
                  Fixed xOffset, Fixed yOffset);
```

<code>target</code>	A reference to the shape you want to rotate.
<code>degrees</code>	The number of degrees to rotate the shape.
<code>xOffset</code>	The horizontal coordinate of the origin to rotate the shape around.
<code>yOffset</code>	The vertical coordinate of the origin to rotate the shape around.

### ***DESCRIPTION***

The `GXRotateShape` function rotates the shape referenced by the `target` parameter by the number of degrees specified in the `degrees` parameter around the origin specified by the `xOffset` and `yOffset` parameters.

This function rotates the target shape in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function recalculates the control points of the shape's geometry to effect the rotation.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXRotateTransform` function; it recalculates the mapping matrix of the target shape's transform object to effect the rotation. If the target shape shares this transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set.

The coordinates of the origin are specified in geometry space.

### ***ERRORS, WARNINGS, AND NOTICES***

#### **Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>shape_access_not_allowed</code>	(debugging version)

#### **Warnings**

<code>rotate_shape_out_of_range</code>
<code>graphic_type_cannot_be_moved</code>

#### **Notices (debugging version)**

<code>mapping_unaffected</code>
---------------------------------

**SEE ALSO**

For an example of the use of this function, see Listing 6-4 on page 6-25.

To rotate a shape by altering the mapping property of its transform object, you can also use the `GXRotateTransform` function, described on page 6-62.

***GXSkewShape***

---

You can use the `GXSkewShape` function to skew a shape about a specified origin by specified horizontal and vertical factors.

```
void GXSkewShape(gxShape target, Fixed xSkew, Fixed ySkew,
                 Fixed xOffset, Fixed yOffset);
```

<code>target</code>	A reference to the shape you want to skew.
<code>hSkew</code>	The amount to skew the shape horizontally.
<code>vSkew</code>	The amount to skew the shape vertically.
<code>xOffset</code>	The horizontal coordinate of the origin to skew the shape about.
<code>yOffset</code>	The vertical coordinate of the origin to skew the shape about.

**DESCRIPTION**

The `GXSkewShape` function skews the shape referenced by the `target` parameter horizontally by the factor specified in the `hSkew` parameter and vertically by the factor specified in the `vSkew` parameter. The skewing is centered about the origin specified in the `xOffset` and `yOffset` parameters. (The origin is the point whose coordinates do not change as a result of the skewing operation.)

The skew factors are expressed as a proportional amount of shift in one direction with distance in the perpendicular direction. A value of 0 for the `hSkew` or `vSkew` parameter indicates no skewing in the corresponding direction.

This function skews the target shape in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function recalculates the control points of the shape's geometry to effect the skewing.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXSkewTransform` function; it recalculates the mapping matrix of the target shape's transform object to effect the skewing. If the target shape shares this transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set.

The coordinates of the origin are specified in geometry space.

## Transform Objects

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)

**Warnings**

`skew_shape_out_of_range`  
`graphic_type_cannot_be_moved`

**Notices (debugging version)**

`mapping_unaffected`

## SEE ALSO

For examples of the use of this function, see Listing 6-4 on page 6-25 and Listing 6-5 on page 6-27.

To skew a shape by altering the mapping property of its transform object, you can also use the `GXSkewTransform` function, described on page 6-63.

***GXMapShape***

You can use the `GXMapShape` function to apply an arbitrary mapping matrix to a shape.

```
void GXMapShape(gxShape target, const gxMapping *map);
```

<code>target</code>	A reference to the shape you want to apply the mapping to.
<code>map</code>	A pointer to a mapping structure containing the matrix you want to apply to the target shape.

## DESCRIPTION

The `GXMapShape` function applies to the target shape the mapping transformations represented by the mapping matrix pointed to by the `map` parameter.

This function applies the mapping in one of two ways:

- If the target shape's `gxMapTransformShape` attribute is cleared, the function applies the mapping matrix directly to the points of the shape's geometry.
- If the target shape's `gxMapTransformShape` attribute is set, this function is identical to the `GXMapTransform` function; it concatenates the target shape's transform mapping with the mapping matrix pointed to by the `map` parameter. If the target shape shares its transform object with other shapes, QuickDraw GX makes a copy of the transform object, associates the copy with the target shape, and makes changes to the copy.

## Transform Objects

The target shape can be any shape type. However, if the target shape is an empty shape, a full shape, or a picture shape, this function has no effect unless the shape's `gxMapTransformShape` attribute is set.

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

`out_of_memory`  
`shape_is_nil`  
`shape_access_not_allowed` (debugging version)

**Warnings**

`map_shape_out_of_range`  
`graphic_type_cannot_be_moved`

**Notices (debugging version)**

`mapping_unaffected`

**SEE ALSO**

To apply a mapping matrix to the mapping property of a transform object, you can also use the `GXMapTransform` function, described on page 6-64.

**Getting and Setting the View Port List**

---

This section describes the functions you can use to manipulate the view port list property of a transform object. For information about the view port list property, see “View Port List” on page 6-11. For general information about view ports, see the chapter “View-Related Objects” in this book.

***GXGetTransformViewPorts***

---

You can use the `GXGetTransformViewPorts` function to retrieve the view port list of a transform object.

```
long GXGetTransformViewPorts(gxTransform source,
                             gxViewport list[]);
```

<code>source</code>	A reference to the transform object whose view port list you want to examine.
<code>list</code>	An array of view port references. On return, the array contains the view port list of the source transform.

*function result* The number of view ports in the source transform's view port list.

## Transform Objects

**DESCRIPTION**

The `GXGetTransformViewPorts` function copies the list of view port references in the source transform into the array referenced by the `list` parameter, and returns as the function result the total number of view port references in the list.

If you pass `nil` for the `list` parameter, the function returns the number of view ports as the function result, but does not return the references to the view ports. Thus you normally call this function twice: once to determine the size of view port array to allocate, and once to retrieve the array itself.

**SPECIAL CONSIDERATIONS**

This function returns all view port references in the source transform's view port list, including any invalid ones (for view ports that have been disposed of).

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

`out_of_memory`  
`transform_is_nil`

**Notices (debugging version)**

`transform_references_disposed_viewPort`

**SEE ALSO**

For an example of the use of this function, see Listing 6-6 on page 6-29.

To assign a view port list to a transform object, use the `GXSetTransformViewPorts` function, described next.

To retrieve the view port list of the transform object associated with a specified shape, use the `GXGetShapeViewPorts` function, described on page 6-75.

***GXSetTransformViewPorts***

---

You can use the `GXSetTransformViewPorts` function to assign a view port list to a transform object.

```
void GXSetTransformViewPorts(gxTransform target, long count,
                             const gxViewPort list[]);
```

<code>target</code>	A reference to the transform object to which you want to assign the view port list.
<code>count</code>	The number of entries in the new view port list; the size of the <code>list</code> array.
<code>list</code>	The new view port list; an array of references to the view ports you want to associate with the source transform.

## Transform Objects

**DESCRIPTION**

The `GXSetTransformViewPorts` function replaces the view port list of the transform object referenced by the `target` parameter with the view port list contained in the `list` parameter. The `count` parameter specifies the number of view ports in the new list.

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

`out_of_memory`  
`transform_is_nil`  
`invalid_viewPort_reference`  
`parameter_out_of_range` (debugging version)

**Notices (debugging version)**

`transform_viewPorts_already_set`

**SEE ALSO**

For an example of the use of this function, see Listing 6-6 on page 6-29.

To retrieve the view port list of a transform object, use the `GXGetTransformViewPorts` function, described in the previous section.

To assign a view port list to the transform object associated with a specified shape, use the `GXSetShapeViewPorts` function, described on page 6-76.

***GXGetShapeViewPorts***

---

You can use the `GXGetShapeViewPorts` function to retrieve the view port list of the transform object associated with a specific shape.

```
long GXGetShapeViewPorts(gxShape source,
                        gxViewPort list[]);
```

<code>source</code>	A reference to the shape whose transform object contains the view port list you want to examine.
<code>list</code>	An array of view port references. On return, the array contains the view port list of the source shape's transform.

*function result* The number of references in the view port list of the source shape's transform.

## Transform Objects

**DESCRIPTION**

The `GXGetShapeViewPorts` function copies references to the view ports associated with the source shape's transform into the array referenced by the `list` parameter, and returns as the function result the total number of view port references in the list.

If you pass `nil` for the `list` parameter, the function returns the number of view ports as the function result, but does not return the references to the view ports. Thus you normally call this function twice: once to determine the size of view port array to allocate, and once to retrieve the array itself.

**SPECIAL CONSIDERATIONS**

This function returns all view port references in the source transform's view port list, including any invalid ones (for view ports that have been disposed of).

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

`out_of_memory`  
`shape_is_nil`

**Notices (debugging version)**

`transform_references_disposed_viewPort`

**SEE ALSO**

To assign a view port list to the transform object associated with a specified shape, use the `GXSetShapeViewPorts` function, described next.

To retrieve the view port list of a transform object, use the `GXGetTransformViewPorts` function, described on page 6-73.

***GXSetShapeViewPorts***

---

You can use the `GXSetShapeViewPorts` function to assign a view port list to the transform object associated with a particular shape.

```
void GXSetShapeViewPorts(gxShape target, long count,
                        const gxViewport list[]);
```

<code>target</code>	A reference to the shape object whose transform's view port list you want to replace.
<code>count</code>	The number of view port references in the new view port list; the size of the <code>list</code> array.
<code>list</code>	The new view port list; an array of references to the view ports you want to associate with the source shape's transform.



## Transform Objects

**DESCRIPTION**

The `GXSetShapeViewPorts` function replaces the view port list of the transform object associated with the shape referenced by the `target` parameter with the view port list specified by the `list` parameter. The `count` parameter specifies the number of view ports in the new list.

If the source shape shares its transform object with other shapes, this function first copies the transform, associates the copy with the source shape, and then makes changes to the copy.

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

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>invalid_viewPort_reference</code>	
<code>parameter_out_of_range</code>	(debugging version)

**Notices (debugging version)**

`transform_viewPorts_already_set`

**SEE ALSO**

To retrieve the view port list from the transform object associated with a specified shape, use the `GXGetShapeViewPorts` function, described in the previous section.

To assign a view port list directly to a transform object, use the `GXSetTransformViewPorts` function, described on page 6-74.

## Getting and Setting the Hit-Test Parameters

---

This section describes the functions you can use to manipulate the hit-test parameters property of a transform object. For general information on the hit-test parameters property, see the section “Hit-Test Parameters” beginning on page 6-11.

For information on hit-testing and using the `GXHitTestShape` or `GXHitTestPicture` functions, see the chapter “Shape Objects” in this book. For additional information on `GXHitTestPicture`, see the picture shapes chapter of *Inside Macintosh: QuickDraw GX Graphics*.

***GXGetTransformHitTest***

---

You can use the `GXGetTransformHitTest` function to retrieve the hit-test parameters of a transform object.

```
gxShapePart GXGetTransformHitTest(gxTransform source,
                                   Fixed *tolerance);
```

**source**        A reference to the transform object whose hit-test parameters you want to examine.

**tolerance**    A pointer to a `Fixed` value. On return, the value specifies the hit-test tolerance of the source transform.

*function result*   The shape-parts mask of the source transform, specifying the shape parts to be tested for.

**DESCRIPTION**

The `GXGetTransformHitTest` function returns the contents of the hit-test parameters property of the transform object referenced by the `source` parameter. The shape-parts mask is returned as the function result and the hit-test tolerance is returned by the `tolerance` parameter.

Hit-test tolerance is specified in geometry units. You can specify `nil` for the `tolerance` parameter, in which case the hit-test tolerance is not returned.

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

`out_of_memory`  
`transform_is_nil`

**SEE ALSO**

For information about the hit-test parameters property, see “Hit-Test Parameters” beginning on page 6-11. To interpret the values in the shape-parts mask, see Table 6-1 on page 6-12.

To assign hit-test parameters to a transform object, use the `GXSetTransformHitTest` function, described next.

To retrieve the hit-test parameters of the transform associated with a specified shape, use the `GXGetShapeHitTest` function, described on page 6-80.

## ***GXSetTransformHitTest***

---

You can use the `GXSetTransformHitTest` function to assign new hit-test parameters to a transform object.

```
void GXSetTransformHitTest(gxTransform target, gxShapePart mask,
                          Fixed tolerance);
```

<code>target</code>	A reference to the transform object whose hit-test parameters you want to assign.
<code>mask</code>	The shape-parts mask to assign to the target transform.
<code>tolerance</code>	The hit-test tolerance to assign to the target transform. It is measured in geometry units, and can be 0 or any positive number.

### ***DESCRIPTION***

The `GXSetTransformHitTest` function assigns the shape-parts mask contained in the `mask` parameter and the hit-test tolerance contained in the `tolerance` parameter to the transform object referenced by the `target` parameter. The tolerance value cannot be negative.

### ***ERRORS, WARNINGS, AND NOTICES***

#### **Errors**

<code>out_of_memory</code>	
<code>transform_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>tolerance_out_of_range</code>	(debugging version)

### ***SEE ALSO***

For information about the hit-test parameters property, see “Hit-Test Parameters” beginning on page 6-11. To interpret the values in the shape-parts mask, see Table 6-1 on page 6-12.

To retrieve the hit-test parameters of a transform object, use the `GXGetTransformHitTest` function, described in the previous section.

To assign hit-test parameters to the transform associated with a specified shape, use the `GXSetShapeHitTest` function, described on page 6-81.

## ***GXGetShapeHitTest***

---

You can use the `GXGetShapeHitTest` function to retrieve the hit-test parameters of the transform object associated with a particular shape.

```
gxShapePart GXGetShapeHitTest(gxShape source,
                              Fixed *tolerance);
```

**source**        A reference to the shape whose transform object contains the hit-test parameters you want to examine.

**tolerance**    A pointer to a `Fixed` value. On return, the value specifies the hit-test tolerance of the source shape's transform.

*function result* The shape-parts mask of the source shape's transform.

### **DESCRIPTION**

The `GXGetShapeHitTest` function returns the hit-test parameters from the transform object associated with the shape referenced by the `source` parameter. The shape-parts mask is returned as the function result and the hit-test tolerance is returned by the `tolerance` parameter.

Hit-test tolerance is specified in geometry units. You can specify `nil` for the `tolerance` parameter, in which case the hit-test tolerance is not returned.

### **ERRORS, WARNINGS, AND NOTICES**

#### **Errors**

`out_of_memory`  
`shape_is_nil`

### **SEE ALSO**

For information about the hit-test parameters property, see "Hit-Test Parameters" beginning on page 6-11. To interpret the values in the shape-parts mask, see Table 6-1 on page 6-12.

To assign hit-test parameters to the transform associated with a specified shape, use the `GXSetShapeHitTest` function, described next.

To retrieve the hit-test parameters of a specified transform object, use the `GXGetTransformHitTest` function, described on page 6-78.

## ***GXSetShapeHitTest***

---

You can use the `GXSetShapeHitTest` function to assign hit-test parameters to the transform object associated with a particular shape.

```
void GXSetShapeHitTest(gxShape target, gxShapePart mask,
                      Fixed tolerance);
```

<code>target</code>	A reference to the shape associated with the transform object whose hit-test parameters you want to assign.
<code>mask</code>	The shape-parts mask to assign to the transform.
<code>tolerance</code>	The hit-test tolerance to assign to the transform. It is measured in geometry units, and can be 0 or any positive number.

### ***DESCRIPTION***

The `GXSetShapeHitTest` function assigns the shape-parts mask contained in the `mask` parameter and the hit-test tolerance contained in the `tolerance` parameter to the transform object associated with the shape referenced by the `target` parameter. The tolerance value cannot be negative.

If the target shape shares its transform object with other shapes, this function first makes a copy of the transform object, associates it with the target shape, and then assigns the new hit-test parameters to the copy.

### ***ERRORS, WARNINGS, AND NOTICES***

#### **Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>tolerance_out_of_range</code>	(debugging version)

### ***SEE ALSO***

For an example of the use of this function, see page 6-30.

For information about the hit-test parameters property, see “Hit-Test Parameters” beginning on page 6-11. To interpret the values in the shape-parts mask, see Table 6-1 on page 6-12.

To retrieve the hit-test parameters from the transform associated with a specified shape, use the `GXGetShapeHitTest` function, described in the previous section.

To assign hit-test parameters directly to a transform object, use the `GXSetTransformHitTest` function, described on page 6-79.

## Summary of Transform Objects

---

### Constants and Data Types

---

#### *The Transform Object*

```
typedef struct gxPrivateTransformRecord *gxTransform;
```

#### *Shape Parts for Hit-Testing*

```
enum gxShapeParts {          /* (in order of evaluation) */
    gxNoPart                 = 0,
    gxBoundsPart             = 0x0001,
    gxGeometryPart           = 0x0002,
    gxPenPart                 = 0x0004,
    gxCornerPointPart         = 0x0008,
    gxControlPointPart        = 0x0010,
    gxEdgePart                = 0x0020,
    gxJoinPart                = 0x0040,
    gxStartCapPart            = 0x0080,
    gxEndCapPart              = 0x0100,
    gxDashPart                = 0x0200,
    gxPatternPart             = 0x0400,
    gxGlyphBoundsPart         = gxJoinPart,
    gxGlyphFirstPart          = gxStartCapPart,
    gxGlyphLastPart           = gxEndCapPart,
    gxSideBearingPart         = gxDashPart,
    gxAnyPart                 = gxBoundsPart | gxGeometryPart |
        gxPenPart | gxCornerPointPart | gxControlPointPart |
        gxEdgePart | gxJoinPart | gxStartCapPart |
        gxEndCapPart | gxDashPart | gxPatternPart
} ;

typedef long gxShapePart;
```

## Functions

---

### *Creating and Manipulating Transform Objects*

```

gxTransform GXNewTransform    (void);
void GXDisposeTransform      (gxTransform target);
void GXCopyToTransform       (gxTransform target, gxTransform source);
boolean GXEqualTransform     (gxTransform one, gxTransform two);
gxTransform GXCloneTransform (gxTransform source);

```

### *Manipulating Transform Object Properties*

```

void GXResetTransform      (gxTransform target);
long GXGetTransformOwners  (gxTransform source);
long GXGetTransformTags   (gxTransform source, long tagType,
                           long index, long count, gxTag items[]);
void GXSetTransformTags    (gxTransform target, long tagType,
                           long index, long oldCount, long newCount,
                           const gxTag items[]);

```

### *Getting and Setting a Transform's Clip*

```

void GXSetTransformClip    (gxTransform target, gxShape clip);
gxShape GXGetTransformClip (gxTransform source);
gxShape GXGetShapeClip     (gxShape source);
void GXSetShapeClip        (gxShape target, gxShape clip);

```

### *Performing Geometric Arithmetic on Transform Clips*

```

void GXUnionTransform      (gxTransform target, gxShape operand);
void GXIntersectTransform  (gxTransform target, gxShape operand);
void GXDifferenceTransform (gxTransform target, gxShape operand);
void GXReverseDifferenceTransform
                           (gxTransform target, gxShape operand);
void GXExcludeTransform    (gxTransform target, gxShape operand);

```

### *Getting and Setting a Transform's Mapping*

```

gxMapping *GXGetTransformMapping
                           (gxTransform source, gxMapping *map);
void GXSetTransformMapping (gxTransform target const gxMapping *map);
gxMapping *GXGetShapeMapping(gxShape source, gxMapping *map);
void GXSetShapeMapping     (gxShape target, const gxMapping *map);

```

## Transform Objects

***Transforming Shapes by Modifying Transform Mappings***

```

void GXMoveTransform      (gxTransform target, Fixed deltaX,
                          Fixed deltaY);

void GXMoveTransformTo    (gxTransform target, Fixed x, Fixed y);

void GXScaleTransform     (gxTransform target, Fixed hScale,
                          Fixed vScale, Fixed xOffset, Fixed yOffset);

void GXRotateTransform    (gxTransform target, Fixed degrees,
                          Fixed xOffset, Fixed yOffset);

void GXSkewTransform      (gxTransform target, Fixed xSkew,
                          Fixed ySkew, Fixed xOffset, Fixed yOffset);

void GXMapTransform       (gxTransform target, const gxMapping *map);

```

***Transforming Shapes by Modifying Shape Geometries***

```

void GXMoveShape          (gxShape target, Fixed deltaX, Fixed deltaY);

void GXMoveShapeTo        (gxShape target, Fixed x, Fixed y);

void GXScaleShape         (gxShape target, Fixed hScale, Fixed vScale,
                          Fixed xOffset, Fixed yOffset);

void GXRotateShape        (gxShape target, Fixed degrees,
                          Fixed xOffset, Fixed yOffset);

void GXSkewShape          (gxShape target, Fixed xSkew, Fixed ySkew,
                          Fixed xOffset, Fixed yOffset);

void GXMapShape           (gxShape target, const gxMapping *map);

```

***Getting and Setting a Transform's View Ports***

```

long GXGetTransformViewPorts (gxTransform source, gxViewport list[]);

void GXSetTransformViewPorts (gxTransform target, long count,
                              const gxViewport list[]);

long GXGetShapeViewPorts    (gxShape source, gxViewport list[]);

void GXSetShapeViewPorts    (gxShape target, long count,
                              const gxViewport list[]);

```

***Getting and Setting a Transform's Hit-Test Parameters***

```

gxShapePart GXGetTransformHitTest
                                (gxTransform source, Fixed *tolerance);

void GXSetTransformHitTest      (gxTransform target, gxShapePart mask,
                                Fixed tolerance);

gxShapePart GXGetShapeHitTest   (gxShape source, Fixed *tolerance);

void GXSetShapeHitTest          (gxShape target, gxShapePart mask,
                                Fixed tolerance);

```