

## Picture Shapes

This chapter describes picture shapes and the functions you use to manipulate them. It also discusses the functions described in other chapters that you can apply to picture shapes.

In particular, this chapter shows you how you can create and draw picture shapes; edit a picture shape's list of items; override style, ink, and transform information for items in a picture; create picture hierarchies; and hit-test picture shapes.

You should be familiar with the information in the chapter "Shape Objects" of *Inside Macintosh: QuickDraw GX Objects* before you read this chapter, and you will probably want to be familiar with the information in the chapter "Transform Objects" of that book. You might also want to be familiar with the other shape types, which are described in Chapter 2, "Geometric Shapes," and Chapter 5, "Bitmap Shapes," of this book, as well as in the chapter "Typographic Shapes" of *Inside Macintosh: QuickDraw GX Typography*.

## About Picture Shapes

A **picture shape** represents a collection of other shapes. For example, you could create a scroll bar using a picture shape:

- You could create separate polygon shapes to represent the scroll box, the gray area, and the two scroll arrows.
- You could then collect these individual polygon shapes into a single picture shape to represent the entire scroll bar.

Using picture shapes, you can create complex graphics, create shapes with both graphic and typographic content, combine multiple bitmaps into a single shape, create groups of shapes, create shape layers, group shapes into pages to prepare for printing, and so on.

Like any QuickDraw GX shape, a picture shape is represented in memory by a shape object, a style object, an ink object, and a transform object. A shape object representing a picture shape contains the same properties as a shape object representing a geometric or a typographic shape: owner count, tag list, shape type, shape fill, geometry, and so on.

Since picture shapes contain other shapes, they don't make much use of their shape fill property, although you can specify a no-fill shape fill if you don't want the picture to appear when drawn.

Picture shapes also don't make much use of their associated style object, since each shape in the picture has its own style object.

Pictures shapes also don't make much use of their ink objects for the same reasons.

Picture shapes do make full use of their transform objects, however. For example, you can scale, skew, rotate, and clip picture shapes as a whole, as well as separately for each individual shape in the picture. This process is described in more detail in the section "Transform Concatenation" beginning on page 6-19.

Picture shapes differ from other types of shapes primarily in the content of their geometries. A picture shape's geometry contains a list of picture items. Each **picture item** contains a reference to another shape.

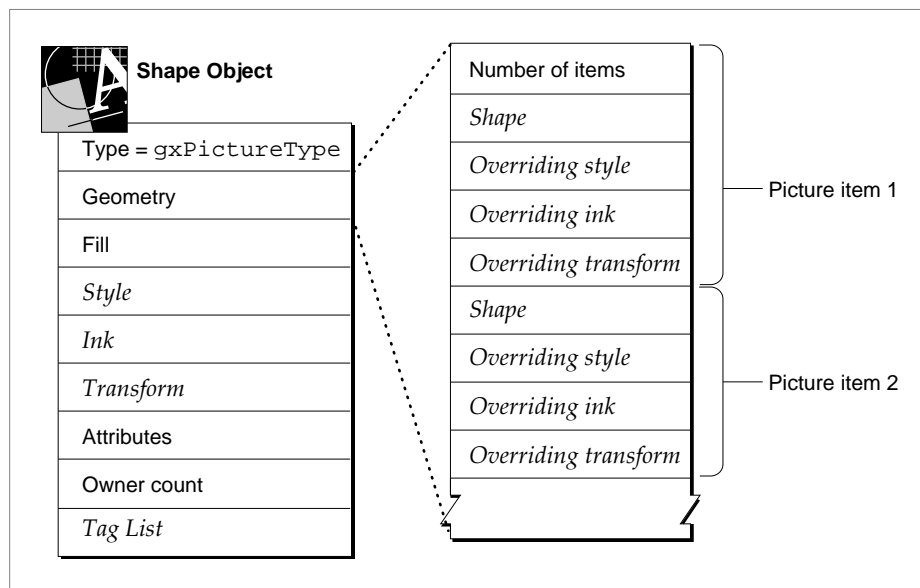
## Picture Shapes

Although each of the shapes in a picture has its own style, ink, and transform object, picture shapes allow you to provide an overriding style, ink, and transform object for each of these shapes. QuickDraw GX uses this overriding information only when drawing the picture. Even after you insert a shape into a picture, you can still draw the original shape using its original style, ink, and transform object.

Overriding objects are described in the next section “Overriding Styles, Inks, and Transforms” beginning on page 6-8.

Figure 6-1 shows a graphic representation of a picture shape and a picture geometry.

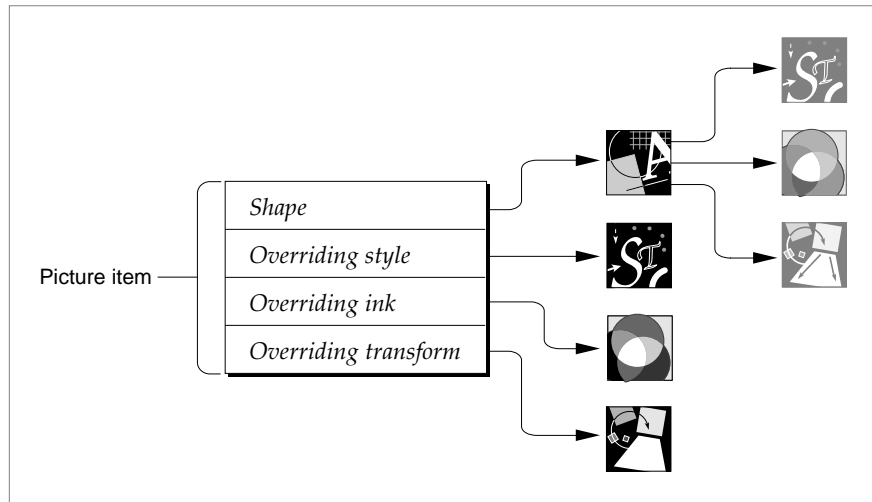
**Figure 6-1** A picture shape



## Picture Shapes

Figure 6-2 shows a single picture item. This item contains a reference to a shape object, which contains a reference to its associated style, ink, and transform objects. These objects are shown in grey, because the picture item also contains references to an overriding style, ink, and transform object for the shape.

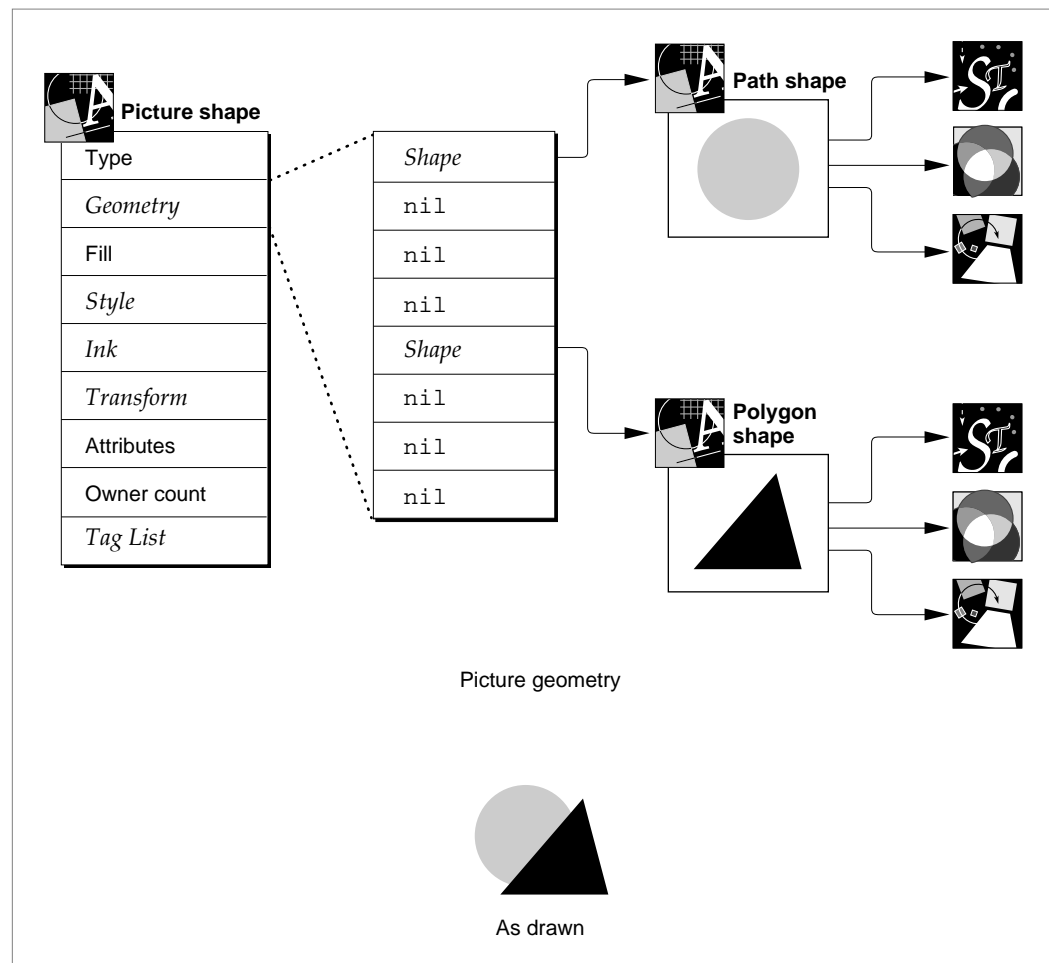
**Figure 6-2** A picture item



## Picture Shapes

Figure 6-3 shows an example of a picture shape with a geometry that contains two picture items. Each item contains a reference to a shape, but neither item contains a reference to an overriding style, ink, or transform object. Therefore, when QuickDraw GX draws this picture, it draws each shape in the picture using the style, ink, and transform information originally associated with the shape, as shown at the bottom of Figure 6-3.

**Figure 6-3** A picture geometry with two items

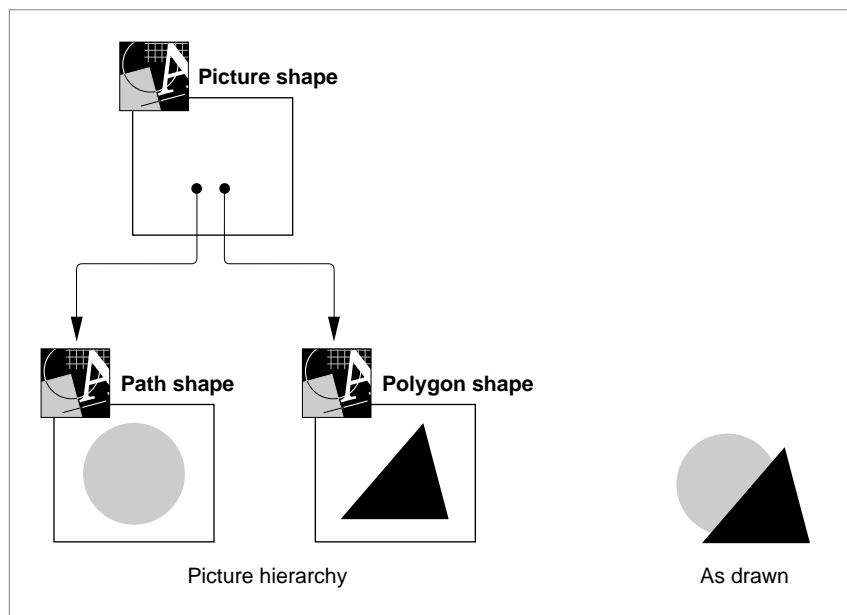


## Picture Shapes

Notice that QuickDraw GX draws the shapes in a picture in the order the references to them appear in the picture geometry: from back to front.

Figure 6-3 shows the entire shape object and picture geometry for the picture shape. Figure 6-4 shows a condensed view of the same picture. This chapter uses condensed views of picture shapes when drawing picture hierarchies, which are described in “Picture Hierarchies” beginning on page 6-18.

**Figure 6-4** Condensed view of picture with two items

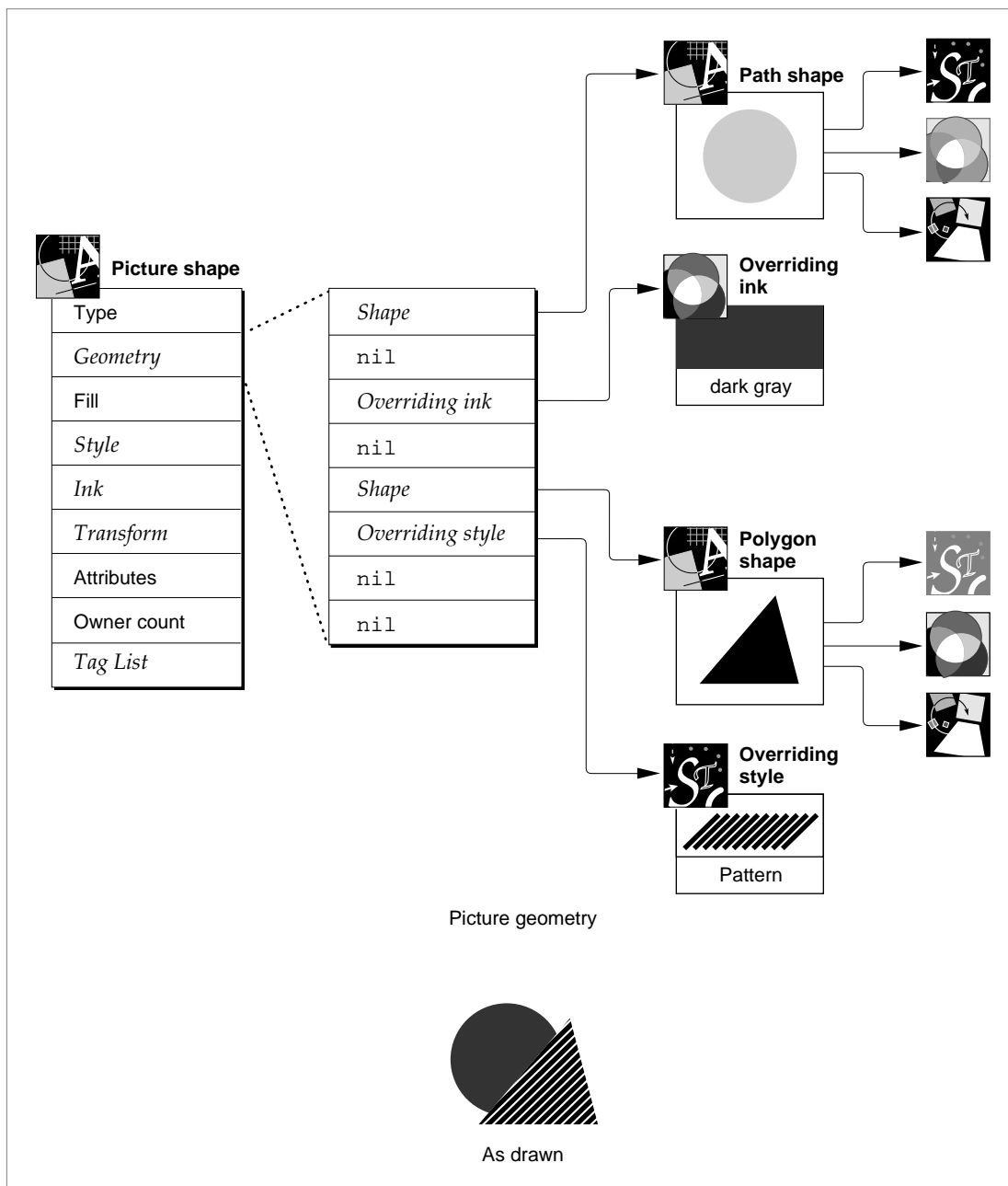


## Overriding Styles, Inks, and Transforms

---

QuickDraw GX allows you to specify an **overriding style, ink, or transform** object for any item in a picture. If an item has an overriding style, ink, or transform object, QuickDraw GX uses the information in the overriding object rather than the information in the original style, ink, or transform when drawing that item of the picture shape.

Figure 6-5 shows the picture from Figure 6-4 with overriding information added. In this figure, the first picture item has an overriding ink, which specifies a dark gray color. The second picture item has an overriding style, which specifies a pattern.

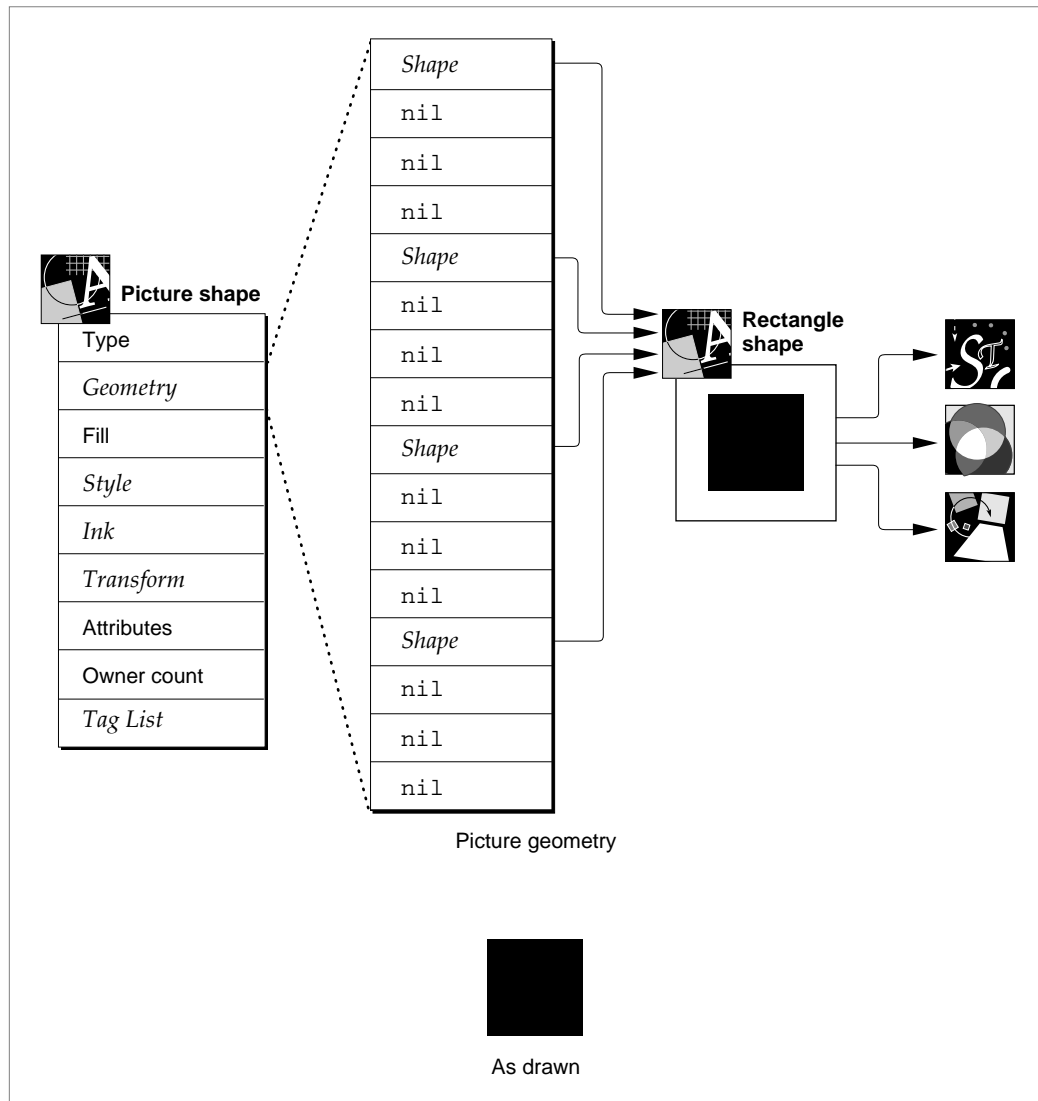
**Figure 6-5** A picture shape with overrides

When QuickDraw GX draws the picture represented in Figure 6-5, it draws the first picture item using the information in the overriding ink object, rather than the information in the ink object originally associated with the first item. Similarly, when it draws the second shape, it uses the information in the overriding style rather than the information in the original style.

## Multiple References

QuickDraw GX allows multiple items in a picture to reference the same shape. Figure 6-6 shows an example of a picture shape containing four items. In this example, each item references the same shape: a black rectangle.

**Figure 6-6** A picture containing multiple references to the same shape

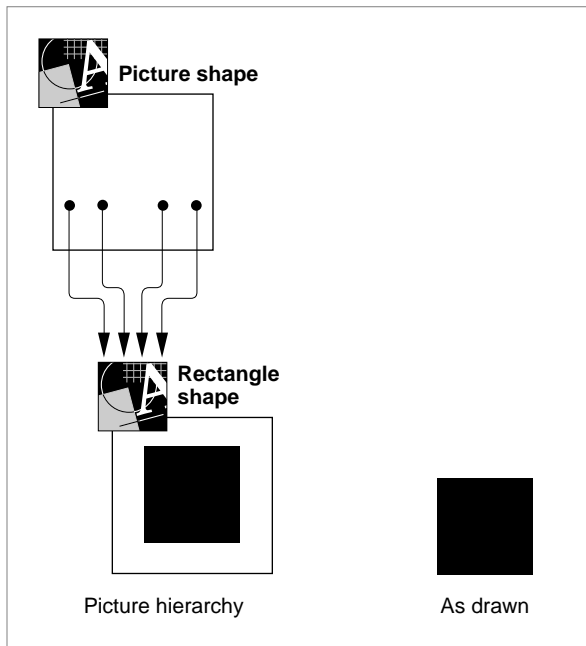




## Picture Shapes

Figure 6-7 shows the condensed view of the picture from Figure 6-6.

**Figure 6-7** A condensed view of a picture with multiple references

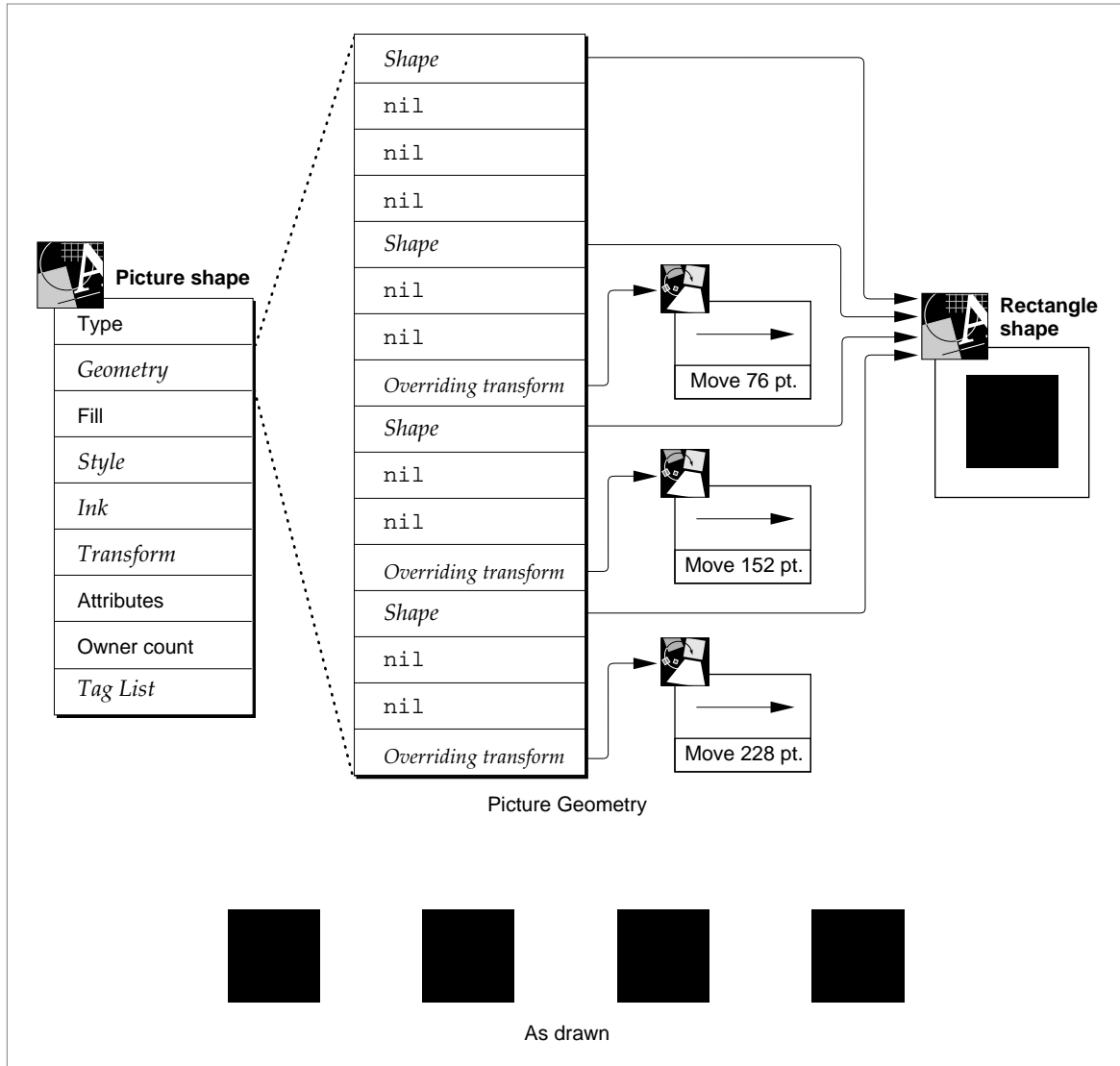


Although the picture shape shown in Figure 6-7 contains four references to the black rectangle, only one black rectangle appears when the picture is drawn. You might expect the rectangle to be drawn four times; however, it only appears once because the rectangle is redrawn in the same location four times.

## Picture Shapes

Having multiple references to the same shape becomes more useful when you add overriding information. For example, if you add overriding transforms to three of the items in the picture shape from Figure 6-6, all four items appear when the picture is drawn, as shown in Figure 6-8.

**Figure 6-8** Multiple references with overriding transforms



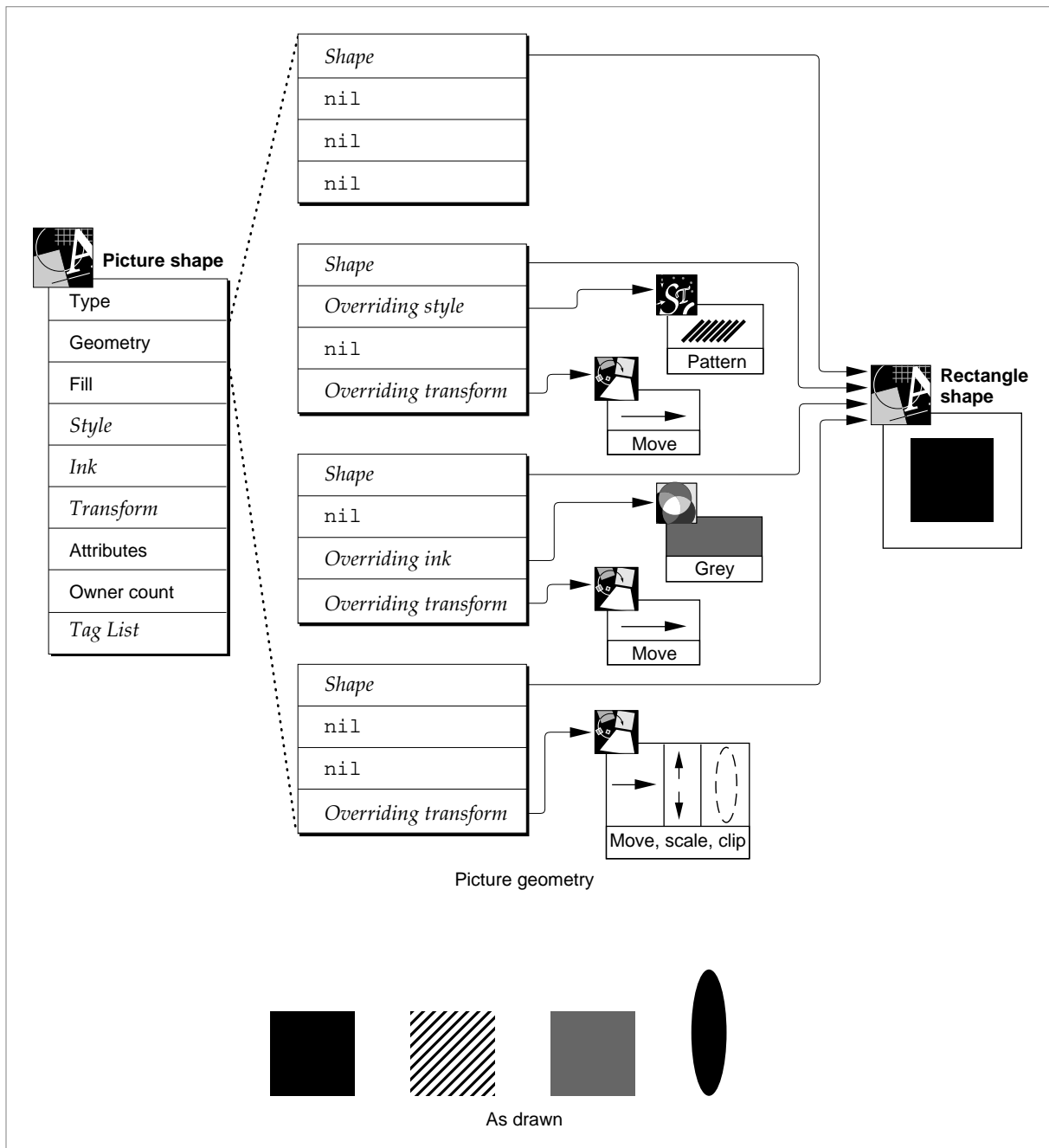
## Picture Shapes

The picture shape in Figure 6-8 contains four items each referencing the same black rectangle shape. However, the second, third, and fourth items contain overriding transforms. When drawing this picture shape, QuickDraw GX applies the original transform when drawing the first item, and applies the overriding transforms when drawing the second, third, and fourth items. In this way, the four items appear separate when the picture is drawn, even though all four items reference the same shape.

## Picture Shapes

You can use overriding styles and inks to make multiple references even more powerful. In Figure 6-9, the second item has an overriding style as well as an overriding transform, the third item has an overriding ink as well as an overriding transform, and the fourth item has an overriding transform that not only moves, but scales and clips as well.

**Figure 6-9** Multiple references with overriding styles, inks, and transforms



## Picture Shapes

Although each item in the picture shape shown in Figure 6-9 references the same black rectangle, the use of overriding styles, inks, and transforms creates substantial variations in the items of the picture as drawn.

For more examples of multiple references and overriding styles, inks, and transforms, see “Adding Multiple References” beginning on page 6-40.

## Unique Items Shape Attribute

One of the shape attributes provided by QuickDraw GX is the **unique items attribute**. This attribute affects the way shapes are added to a picture:

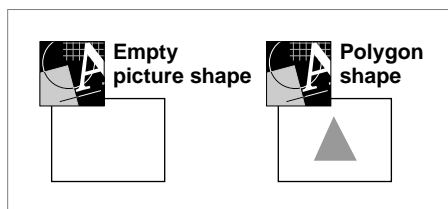
- If a picture shape does not have the unique items attribute set, QuickDraw GX adds shapes to the picture by reference.
- If a picture shape does have the unique items attribute set, QuickDraw GX adds shapes to the picture by copying the shapes and adding a reference to the copy.

Although you may clear the unique items attribute for a picture at any time, you may set the unique items attribute only when a picture is empty—that is, only when the picture contains no items.

You set or clear the unique items attribute using the `GXGetShapeAttributes` function, which is described in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

Figure 6-10 depicts an empty picture shape and a polygon shape. The following two figures use these shapes to illustrate the effect of the unique items attribute.

**Figure 6-10** An empty picture shape and a polygon shape

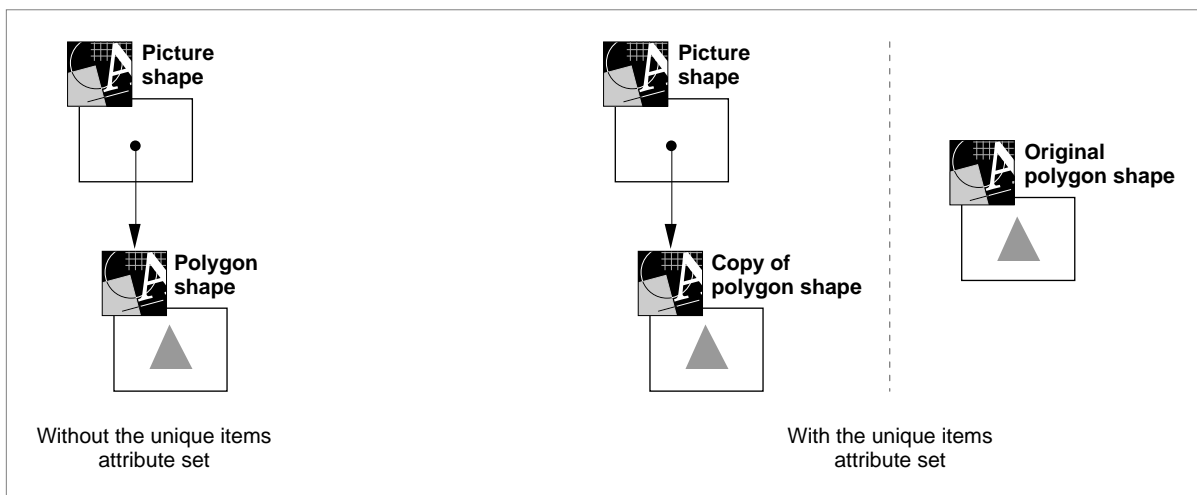


## Picture Shapes

Figure 6-11 shows the result of adding the polygon shape to the empty picture shape. In the left half of this figure, the picture shape does not have the unique items attribute set. In this case, adding the polygon shape to the empty picture simply adds a reference to the polygon shape to the geometry of the picture shape and increases the owner count of the polygon shape.

In the right half of this figure, the picture shape has the unique items attribute set. In this case, adding the polygon shape to the empty picture creates a deep copy of the polygon shape—including all objects referenced by the polygon shape—and adds the copy to the geometry of the picture shape. The original polygon shape is unchanged.

**Figure 6-11** Adding a polygon shape to a picture shape



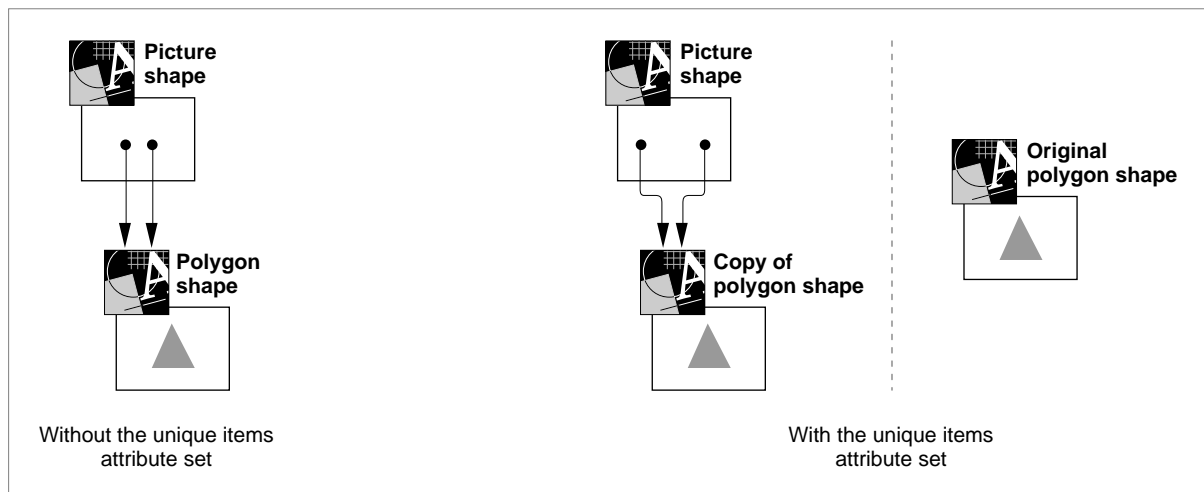
## Picture Shapes

Figure 6-12 shows the result of adding the same polygon shape to the empty picture shape twice.

In the left half of this figure, the picture shape does not have the unique items attribute set. In this case, the first time the polygon shape is added to the empty picture, a reference to the polygon shape is added to the geometry of the picture shape and the owner count of the polygon shape is incremented. The second time the polygon is added to the picture, another reference to the polygon is added to the picture geometry and the owner count of the polygon is incremented again.

In the right half of this figure, the picture shape has the unique items attribute set. In this case, the first time the polygon shape is added to the empty picture, QuickDraw GX creates a deep copy of the polygon shape—including all objects referenced by the polygon shape—and adds a reference to the copy to the geometry of the picture shape. The original polygon shape is unchanged. The second time the polygon is added to the picture, QuickDraw GX notices that the polygon has already been added to the picture and has not been changed. Therefore, to avoid making a second deep, QuickDraw GX simply adds to the picture geometry another reference to the first deep copy.

**Figure 6-12** Adding a shape to a picture twice



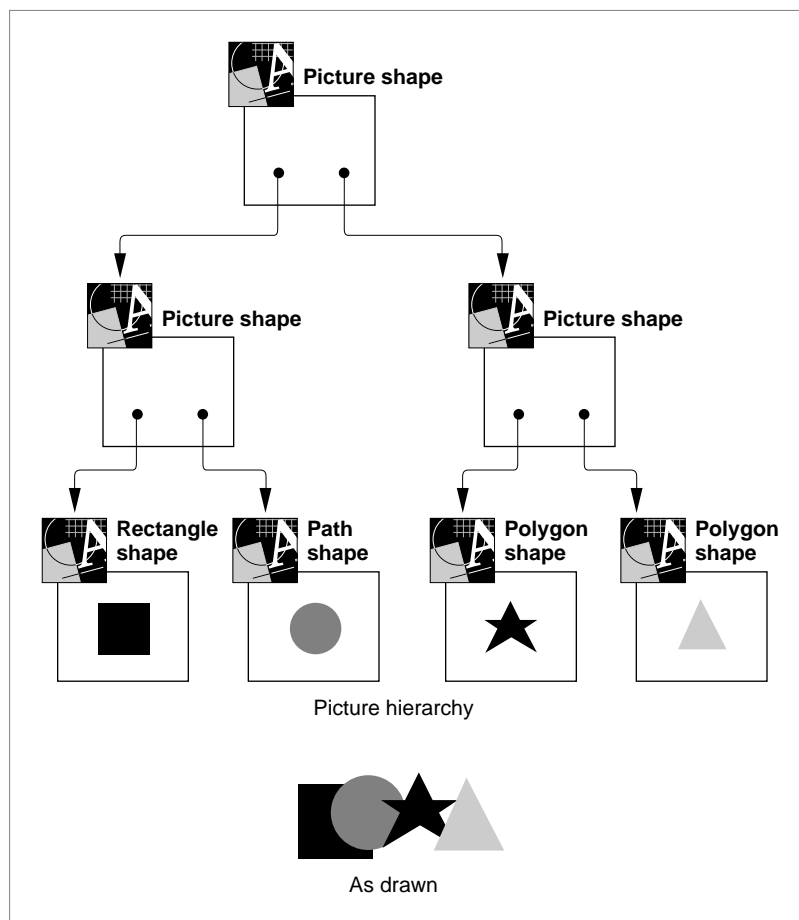
For more examples involving the unique items attribute, see “Adding Items With the Unique Items Attribute Set” beginning on page 6-43.

## Picture Hierarchies

Each item of a picture shape contains a reference to another shape. These shapes can be of any type, including other picture shapes. When a picture shape contains references to other picture shapes, you have a **picture hierarchy**. Figure 6-13 shows a picture hierarchy.

Figure 6-13 depicts a picture shape with two items. Each item references another picture shape, each of which also has two items. This figure shows the condensed view of the picture hierarchy.

**Figure 6-13** A condensed view of a picture hierarchy





## Picture Shapes

Each item in a picture hierarchy has a **level**. The two items belonging to the topmost picture shape—which are picture shapes themselves—have a level of 1. Items belonging to pictures that have a level of 1 have a level of 2, and so on. In the picture hierarchy shown in Figure 6-13, the four geometric shapes all have a level of 2.

## Transform Concatenation

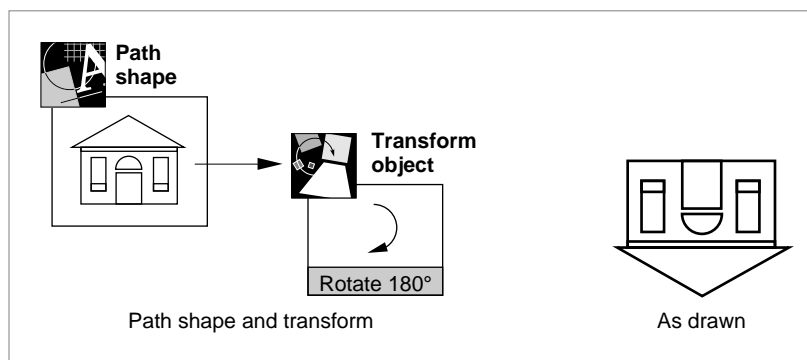
Each item in a picture shape has its own transform object and possibly an overriding transform as well. When QuickDraw GX draws a picture shape, it maps and clips each item according to the mapping and clipping information stored in that item's transform object (or the information in the item's overriding transform, if it has one).

After applying mappings and clippings to the individual items of a picture, QuickDraw GX applies a mapping and clipping to the entire picture, as indicated by the transform object associated with the picture shape. In this way, each item in the picture can go through two transformations: an individual transformation as indicated by the item's individual transform (or overriding transform), and a group transformation as indicated by the picture shape's transform. This process is called **transform concatenation**.

If a picture shape contains a picture hierarchy, QuickDraw GX repeats this concatenation process from the individual shapes at the lowest level of the hierarchy all the way up to the picture shape at the highest level of the hierarchy.

As an example, Figure 6-14 shows a path shape representing a house. This path shape has a transform that rotates it 180 degrees.

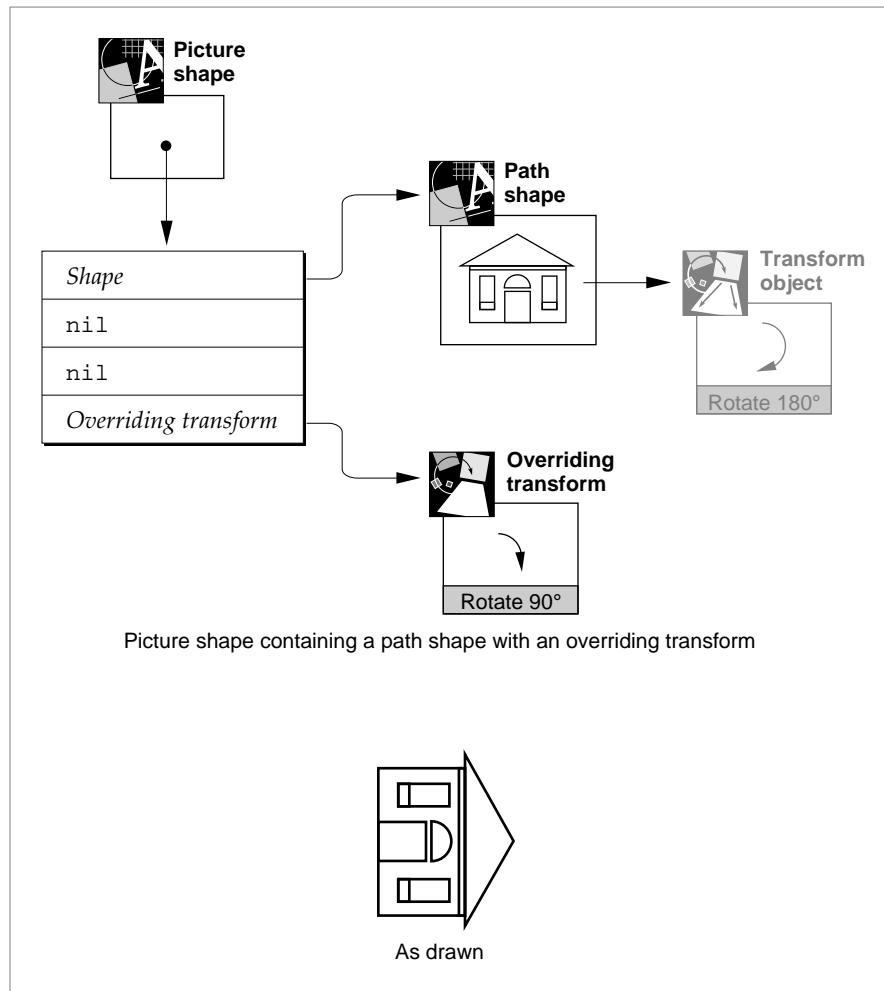
**Figure 6-14** A path shape and its transform



## Picture Shapes

Figure 6-15 shows the same path shape, but in this figure the path shape has been added to a picture shape as the picture's only item. This item includes an overriding transform. When drawing this picture, QuickDraw GX ignores the original transform, and rotates every item in the path shape clockwise by 90 degrees, as specified in the overriding transform.

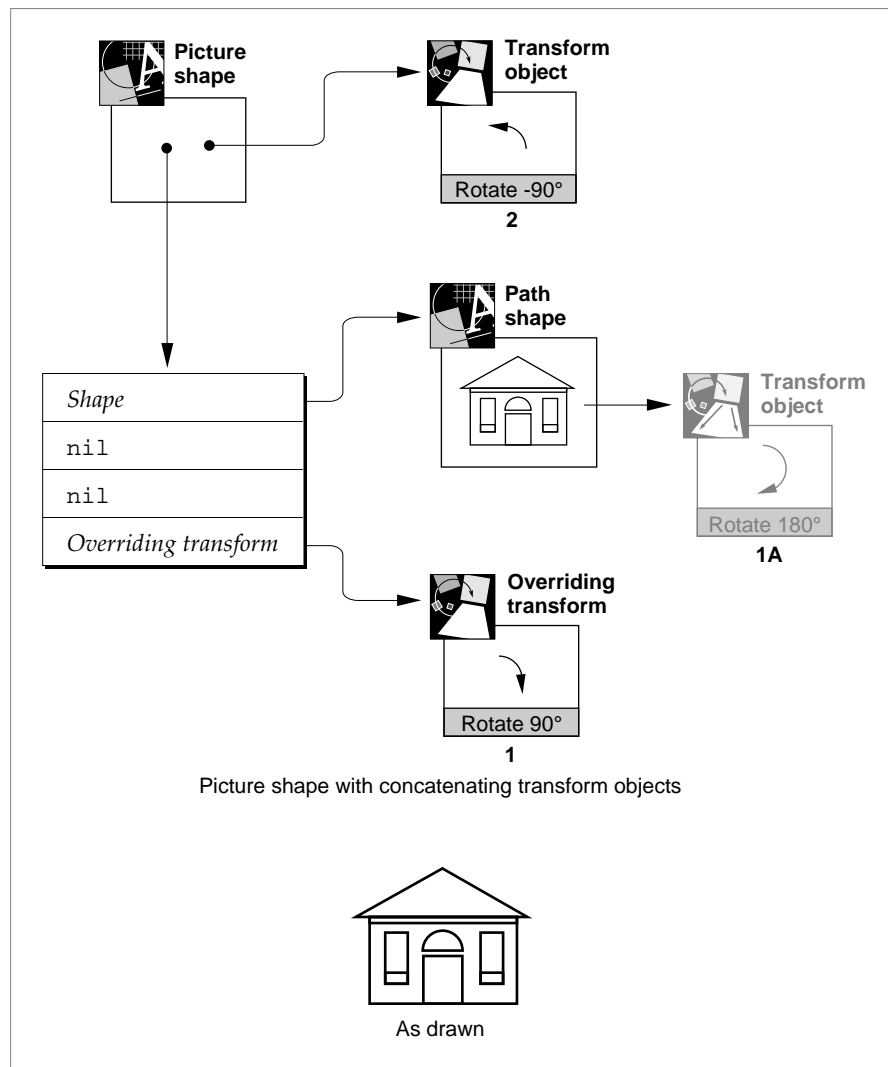
**Figure 6-15** A picture with an overriding transform



## Picture Shapes

Figure 6-16 shows the same picture shape as Figure 6-15. In Figure 6-16, however, the picture shape at the top of the picture hierarchy has its own transform object that specifies that the entire picture should be rotated counterclockwise by 90 degrees. QuickDraw GX concatenates the overriding transform of the path shape (labeled 1 in the picture) with the transform of the top of the picture hierarchy (labeled 2 in the picture), and draws the house at its original orientation. The original transform of the path shape (labeled 1A) is ignored because of the overriding transform.

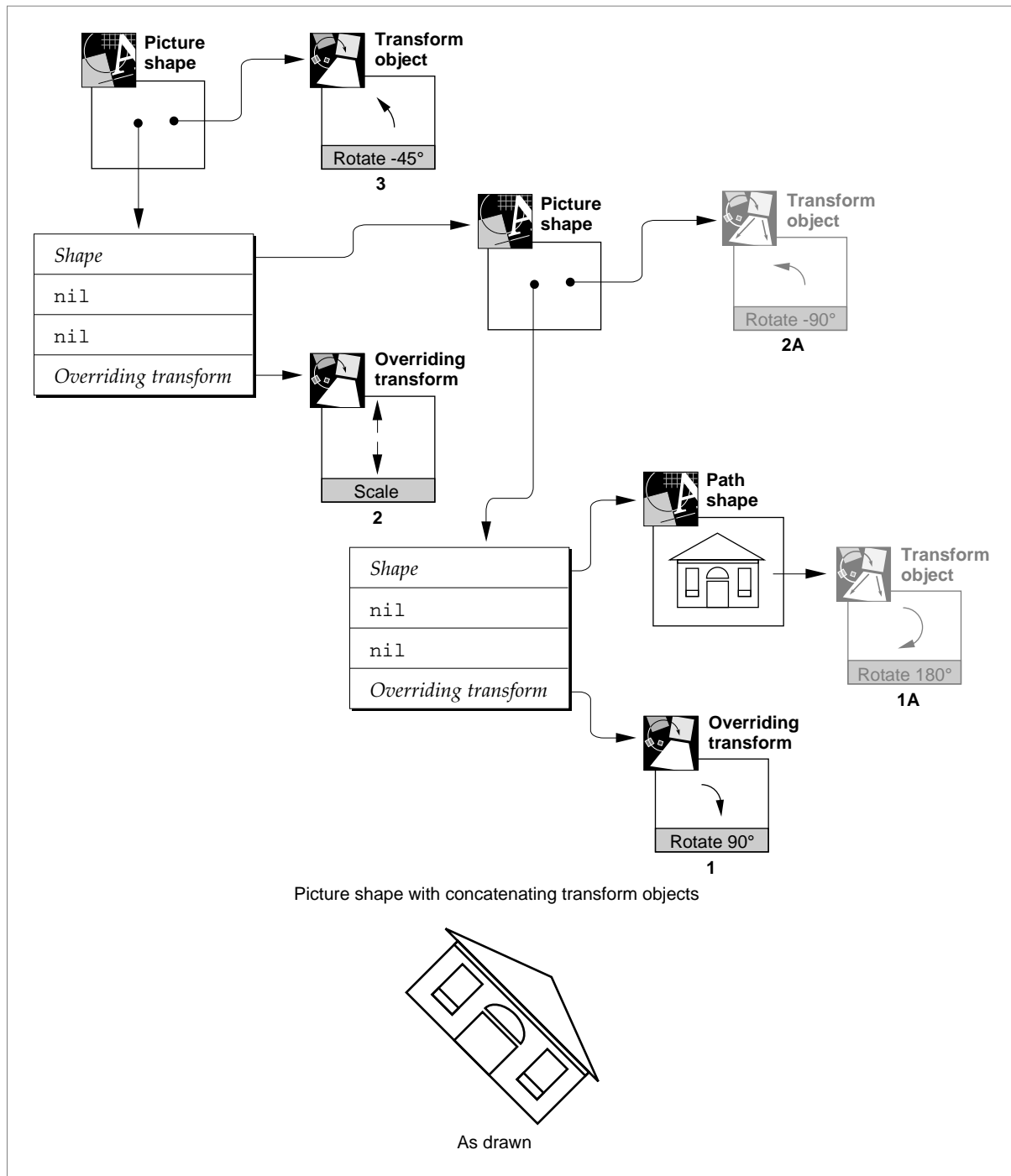
**Figure 6-16** Simple transform concatenation



## Picture Shapes

Figure 6-17 shows an even more complex example of transform concatenation. This figure shows the same picture from Figure 6-16, but in Figure 6-17 this picture has been added as an item to another picture.

To draw this picture, QuickDraw GX uses the overriding transform (labeled 1) of the original path shape, which rotates it 90 degrees to the right. Then QuickDraw GX uses the overriding transform (labeled 2) associated with the picture that contains the path shape, which scales the picture by a factor of 2. Finally, QuickDraw GX uses the transform object (labeled 3) of the picture at the top of the hierarchy, which rotates the picture 45 degrees to the right. The result is shown at the bottom of Figure 6-17.

**Figure 6-17** Intricate transform concatenation

You can find more examples of transform concatenation in “Creating Picture Hierarchies” beginning on page 6-44.

## About Hit-Testing Picture Shapes

---

When the user clicks the mouse, your application receives the information from the Macintosh Toolbox about where the mouse click occurred. By sending this information to the `GXHitTestPicture` function, you can find out which item in a picture was hit. This process is called hit-testing a picture shape.

When hit-testing a picture shape, QuickDraw GX searches through the shapes contained in the picture until it finds the shape that was hit by the hit-test point. As QuickDraw GX searches through the shapes in the picture, it

- hit-tests the shape, using the hit-test information in that shape's transform object (or overriding transform object, if the shape has one) to determine if the shape was hit or not
- determines whether the hit shapes satisfy criteria that you specify

QuickDraw GX returns information about the first item that was hit and satisfies the criteria.

Since more than one shape in a picture can be hit during a single hit-test, you provide QuickDraw GX with extra selection criteria when hit-testing a picture. Specifically, you specify a depth and a level:

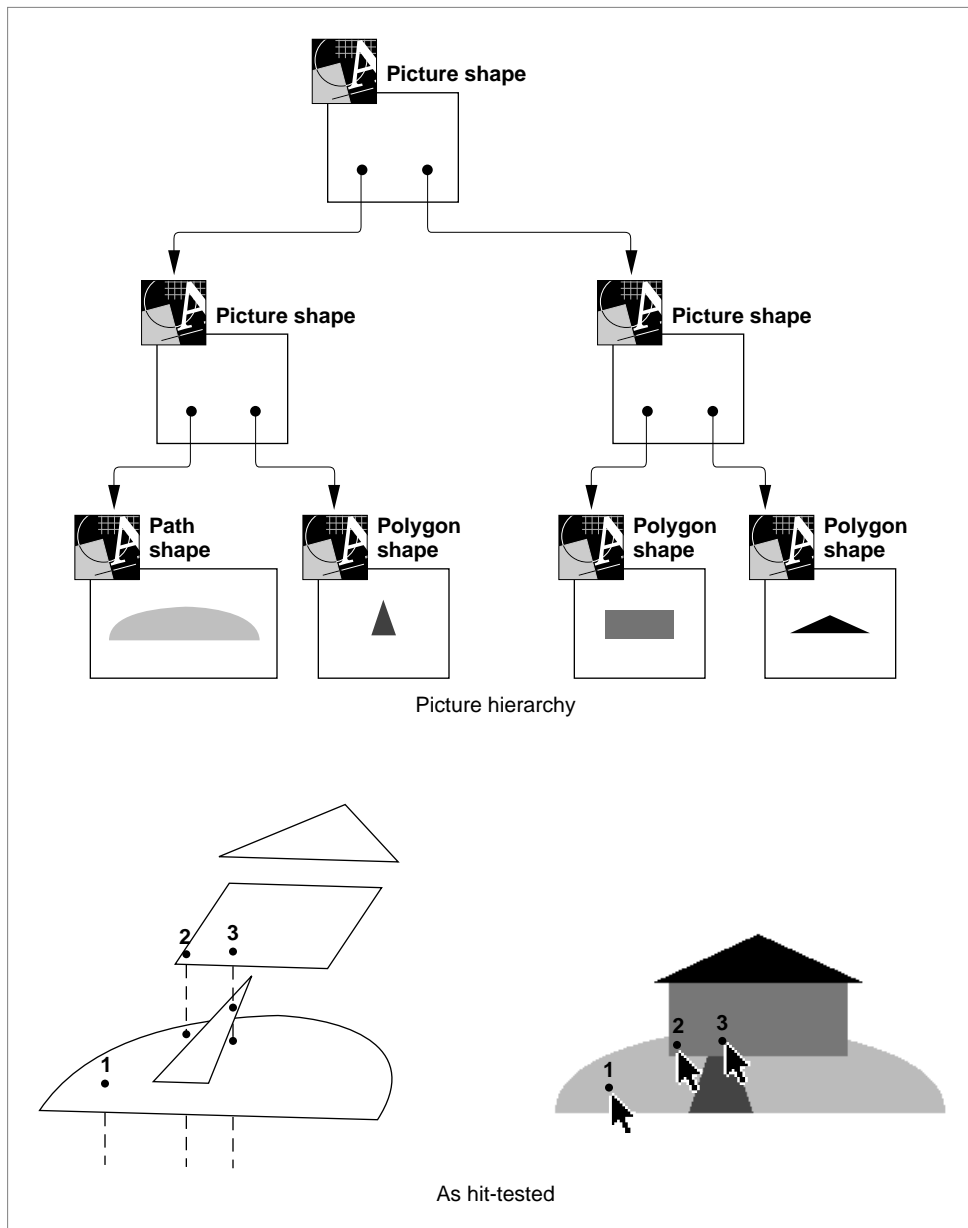
- Pictures frequently contain shapes that overlap when drawn. Therefore, it is possible that the test point hits multiple shapes. For example, if the picture contains two shapes, one on top of the other, the test point might hit both of them. You can control which of these shapes QuickDraw GX selects as the hit shape by specifying a shape **depth**. In this example, specifying a shape depth of 1 would indicate that QuickDraw GX should select the shape that was drawn on top as the hit shape. Specifying a shape depth of 2 would indicate that QuickDraw GX should select the shape that was drawn underneath as the hit shape.
- In a picture hierarchy, each shape can be contained by a picture shape, which in turn can be contained by another picture shape, and so on. If the hit shape has a level of 3, for example, you can specify that QuickDraw GX return a reference to the hit shape by specifying a level of 3. You can specify that QuickDraw GX return a reference to the picture that contains the hit shape by specifying a level of 2. You can specify that QuickDraw GX return a reference to the picture that contains the picture that contains the hit shape by specifying a level of 1.

## Picture Shapes

Figure 6-18 shows an example. The picture shape shown in this figure has two items, each of which are pictures. Each of these pictures has two items itself, making a total of four shapes that have a level of 2 in this hierarchy.

This figure shows the picture as drawn, and three sample hit-test points.

**Figure 6-18** A picture shape and hit-test points



## Picture Shapes

The first sample hit-test point hits only one shape: the lawn path shape. If you specified a depth of 1, QuickDraw GX would select this shape as the hit shape. The shape returned by QuickDraw GX, however, depends on what you specify for the level. If you specify 2, the lawn path shape would be returned. If you specified 1, however, the picture that contains the lawn path shape would be returned.

The second sample hit-test point hits two shapes: at depth 1, it hits the house rectangle; at depth 2, it hits the lawn shape. You determine which is the hit shape by specifying a depth of 1 or 2. You also specify whether QuickDraw GX returns the hit shape (by specifying level 2) or the picture that contains the hit shape (by specifying level 1).

The third hit-test point hits three shapes: at depth 1, it hits the house rectangle; at depth 2, it hits the walkway polygon; at depth 3, it hits the lawn shape. Again, you can determine which of these shapes is the hit shape (by specifying the depth) and whether the hit shape or the picture that contains it is returned (by specifying the level).

For programming examples of hit-testing picture shapes, see “Hit-Testing Pictures” beginning on page 6-46.

## Using Picture Shapes

---

This section shows you how to create, draw, edit, and hit-test picture shapes. In particular, this section shows you how to

- create and draw pictures
- add items to a picture
- remove and replace items in a picture
- provide overriding styles, inks, and transforms for the items in a picture
- add multiple copies of a shape to a picture
- copy objects when adding them to a picture
- create hierarchies of pictures
- hit-test pictures

Although the geometry of a picture shape does not contain geometric points, a picture shape can contain shapes whose geometries do contain geometric points. For this reason, some of the sample functions in this section need to specify geometric points, which are made up of two fixed-point numbers. To convert integers to fixed-point numbers when specifying geometric points, QuickDraw GX provides the `GXIntToFixed` macro:

```
#define GXIntToFixed(a) ((Fixed)(a) << 16)
```

QuickDraw GX also provides the `ff` macro as a convenient alias:

```
#define ff(a) GXIntToFixed(a)
```



## Picture Shapes

The sample functions throughout this section use the `ff` macro when converting an integer constant to a fixed-point constant.

## Creating and Drawing Picture Shapes

QuickDraw GX provides a number of methods to create and draw pictures. In general, you can

- define the items of the picture and draw them without creating a picture shape
- define the items of the picture, incorporate them into a picture shape, and draw the picture shape

You can use the `GXDrawPicture` function to draw pictures using the first method. You send five parameters to this function: a count of how many shapes are in the picture, an array of references to the shapes you want drawn, and arrays of references to the overriding styles, inks, and transforms for those shapes. (See “Using Overriding Styles, Inks, and Transforms” beginning on page 6-38 for examples of overriding styles, inks, and transforms.) The `GXDrawPicture` function creates a temporary picture shape using the information in the arrays you provide, draws the picture shape, and then disposes of the temporary picture shape.

The `GXDrawPicture` function is convenient if you have a set of shapes (and overriding styles, inks, and transforms) that you want to draw only one time.

QuickDraw GX also provides a number of ways for you to create a more permanent picture shape—one that you can edit and draw repeatedly. To create a picture shape, you can

- create an empty picture shape using the `GXNewShape` function and add items to the picture all at once using the `GXSetPicture` function
- create an empty picture shape using the `GXNewShape` function and add items to the picture individually using the `GXSetPictureParts` function or the `AddToPicture` library function
- create a picture with an initial set of items using the `GXNewPicture` function

In any of these three cases, you draw the picture shape using the `GXDrawShape` function.

The `GXSetPicture` function allows you to replace the entire geometry of a picture with a new set of items. For more information, see “Getting and Setting Picture Geometries” beginning on page 6-31.

The `GXSetPictureParts` function provides more sophisticated editing of a picture shape’s item list. For more information, see “Adding Items to a Picture” beginning on page 6-32, and “Removing and Replacing Items in a Picture” beginning on page 6-35.

The `GXNewPicture` function is similar to the `GXDrawPicture` function in that it requires four arrays as parameters: arrays of references to the shapes, the overriding styles, the overriding inks, and the overriding transforms that make up the items of the picture shape. However, unlike the `GXDrawPicture` function, the `GXNewPicture` function creates a picture shape and returns a reference to it to your application. You can use this reference to draw the picture using the `GXDrawShape` function.

## Picture Shapes

Listing 6-1 shows how to draw a picture of a house comprising three shapes: a rectangle for the house itself, another rectangle for the door, and a triangle for the roof. The sample function shown in this listing creates three shapes and draws them using the `GXDrawPicture` function.

---

**Listing 6-1**      Creating a simple picture of a house

```
static gxShape DrawHousePicture(void)
{
    const gxRectangle houseGeometry = {ff(90), ff(80),
                                       ff(200), ff(125)};

    const gxRectangle doorGeometry = {ff(155), ff(95),
                                       ff(170), ff(125)};

    const long roofGeometry[] = {1, /* number of contours */
                                 3, /* number of points */
                                 ff(80), ff(80),
                                 ff(145), ff(50),
                                 ff(210), ff(80)};

    gxShape houseRectangle;
    gxShape roofPolygon;
    gxShape doorRectangle;

    gxShape partsOfHouse[3];

    houseRectangle = GXNewRectangle(&houseGeometry);
    SetShapeCommonColor(houseRectangle, gxGray);

    roofPolygon = GXNewPolygons((gxPolygons *) roofGeometry);

    doorRectangle = GXNewRectangle(&doorGeometry);

    partsOfHouse[0] = houseRectangle;
    partsOfHouse[1] = roofPolygon;
    partsOfHouse[2] = doorRectangle;

    GXDrawPicture(3, partsOfHouse, nil, nil, nil);
}
```

## Picture Shapes

```

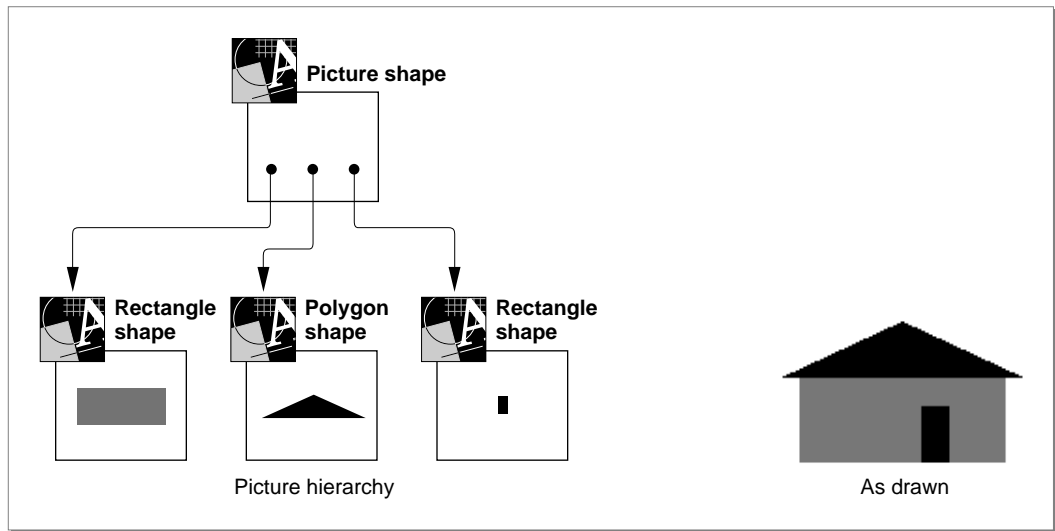
GXDisposeShape(houseRectangle);
GXDisposeShape(roofPolygon);
GXDisposeShape(doorRectangle);

};

```

The results of this sample function are shown in Figure 6-19.

**Figure 6-19** A picture of a house with a roof and a door



The call to the `GXDrawPicture` function in this example creates a temporary picture shape, draws it, and then disposes of it. This function does not return a reference to the picture shape, and so your code never has access to this shape. To create a more permanent picture shape, one that your code can reference, you must first declare a shape reference variable:

```
gxShape housePicture;
```

Then you can replace the call to the `GXDrawPicture` function with calls to the `GXNewPicture` function and the `GXDrawShape` function:

```
housePicture = GXNewPicture(3, partsOfHouse, nil, nil, nil);

GXDrawShape(housePicture);
```

The resulting picture looks the same as the picture drawn by the `GXDrawPicture` function, which is shown in Figure 6-19, but in this case the picture shape exists until you explicitly dispose of it.

## Picture Shapes

You can dispose of the picture shape using the `GXDisposeShape` function:

```
GXDisposeShape(housePicture);
```

In this example, disposing of the house picture also disposes of the three geometric shapes referenced by the house picture. That is, disposing of the house picture releases one of the references to each of the geometric shapes. However, before you dispose of the house picture, each of these shapes has an owner count of 2. (The owner count of each shape starts at 1 when you create it, and the call to the `GXNewPicture` function increments the owner count of each of the shapes.) Therefore, when you dispose of the house picture, the owner count of each of the geometric shapes decrements to 1. To free the memory used by these shapes, you must still dispose of them individually—just as you created them:

```
GXDisposeShape(houseRectangle);
GXDisposeShape(roofPolygon);
GXDisposeShape(doorRectangle);
```

Notice that you can dispose of these three geometric shapes before you dispose of the house picture, as shown in Listing 6-2.

---

**Listing 6-2**     Disposing of shapes contained in a picture before disposing of the picture

```
housePicture = GXNewPicture(3, partsOfHouse, nil, nil, nil);

GXDisposeShape(houseRectangle);
GXDisposeShape(roofPolygon);
GXDisposeShape(doorRectangle);

GXDrawPicture(housePicture);

GXDisposeShape(housePicture);
```

In this example, disposing of the three geometric shapes decrements their owner count by 1, but does not free their memory because the house picture shape still contains a reference to each of the three shapes. Only when the house picture is disposed of is the memory occupied by these three geometric shapes freed.

For information about the `GXDrawPicture` function, see page 6-67. For information about the `GXNewPicture` function, see page 6-57.

## Getting and Setting Picture Geometries

QuickDraw GX provides the `GXGetPicture` function and the `GXSetPicture` function to allow you to examine and replace the entire geometry of a picture shape.

The `GXGetPicture` function returns as its function result the number of items in the picture, and optionally returns an array of references to the shapes referenced by the picture's items, as well as arrays of references to the picture items' overriding styles, inks, and transforms. Typically, you call this function twice. The first time, you determine the number of items in the picture. Then you use that number to allocate enough memory to hold the arrays of references. Finally, you call the function a second time to copy references from the items of the picture into your arrays.

The `GXSetPicture` function allows you to replace the geometry of a picture with a new set of items. This function increments the owner counts of the new shapes, overriding styles, overriding inks, and overriding transforms and disposes of the original shapes, overriding styles, overriding inks, and overriding transforms.

Listing 6-3 gives an example of the `GXGetPicture` function. This example, which builds on the example from Listing 6-1 on page 6-28, edits the picture of the house by moving the location of the door.

**Listing 6-3**      Extracting and editing items from a picture

```
gxShape   *extractedShapes;

long      numberOfItems;
.
.
.
numberOfItems = GXGetPicture(housePicture, nil, nil, nil, nil);
extractedShapes = (gxShape *)
                  NewPtr(numberOfItems * sizeof(gxShape));
GXGetPicture(housePicture, extractedShapes, nil, nil, nil);

GXMoveShape(extractedShapes[2], ff(-40), 0);

GXDrawShape(housePicture);
```

## Picture Shapes

The code in Listing 6-3 includes two new variable declarations: a pointer to shape references and a long integer. The code in this listing calls the `GXGetPicture` function to determine the number of items in the house picture, uses that number to allocate enough memory to store the appropriate number of shape references, and then calls the `GXGetPicture` function a second time to copy the shape references from the items of the picture into the array of shape references. The sample code then uses the `GXMoveShape` function to move the third shape in the picture 40 grid points to the left. Notice that the `extractedShapes` array does not contain copies of the shapes in the picture; instead, it contains copies of references to the shapes in the picture. The references in the `extractedShapes` array reference the actual shapes in the picture. Therefore, moving the shape referenced by the third item in the `extractedShapes` array actually affects the house picture, as shown in Figure 6-20.

**Figure 6-20** A picture of a house with a relocated door



For more information about the `GXGetPicture` and `GXSetPicture` functions, see page 6-59 through page 6-63.

## Adding Items to a Picture

Once you have created a picture shape, you can add more items to it using one of these methods:

- You can use the `GXGetPicture` function to obtain arrays of references to the shapes, overriding styles, overriding inks, and overriding transforms that make up the items of a picture. You can then add new references to these arrays and use the `GXSetPicture` function to replace the original items with the information in the edited arrays.
- You can use the `GXSetPictureParts` function to insert any number of new items directly into a picture shape. With this function, you can insert the new items anywhere in the existing item list.
- You can use the `AddToPicture` library function to insert a single new item at the end of a picture shape's item list.

## Picture Shapes

Listing 6-4 and Listing 6-5 show how to use the `GXSetPictureParts` function to add three new items to the house picture defined in Listing 6-1 on page 6-28. Listing 6-4 defines three new shapes to include in the picture, and Listing 6-5 uses the `GXSetPictureParts` function to insert the shapes into the house picture.

**Listing 6-4** Defining new shapes for the house picture

```

gxShape  lawnPolygon;
gxShape  walkwayPolygon;
gxShape  chimneyRectangle;
.
.
.
const long lawnGeometry[] = {1, /* number of contours */
                             5, /* number of points */
                             0x70000000, /* 0111 0000 ... */
                             ff(20), ff(160), /* on */
                             ff(20), ff(130), /* off */
                             ff(140), ff(100), /* off */
                             ff(260), ff(130), /* off */
                             ff(260), ff(160)}; /* on */

const long walkwayGeometry[] = {1, /* number of contours */
                                3, /* number of points */
                                ff(102), ff(160),
                                ff(122), ff(100),
                                ff(142), ff(160)};

gxRectangle chimneyGeometry = {ff(110), ff(50),
                               ff(120), ff(80)};

lawnPolygon = GXNewPaths((gxPaths *) lawnGeometry);
SetShapeCommonColor(lawnPolygon, light + gxGray);

walkwayPolygon = GXNewPolygons((gxPolygons *) walkwayGeometry);
SetShapeCommonColor(walkwayPolygon, dark + gxGray);

chimneyRectangle = GXNewRectangle(&chimneyGeometry);
SetShapeCommonColor(chimneyRectangle, dark + gxGray);

```

## Picture Shapes

The sample code from Listing 6-4 defines a lawn shape, a walkway shape, and a chimney shape. The sample code in Listing 6-5 creates an array to store references to these three shapes, and then calls the `GXSetPictureParts` function to insert the shapes into the house picture.

---

**Listing 6-5** Adding new shapes to the house picture

```

gxShape  insertedShapes[3];
.
.
.
insertedShapes[0] = lawnPolygon;
insertedShapes[1] = walkwayPolygon;
insertedShapes[2] = chimneyRectangle;

GXSetPictureParts(housePicture,
                  1,    /* insert before first item */
                  0,    /* don't replace any existing items */
                  3,    /* insert three new items */
                  insertedShapes, /* shapes to insert */
                  nil, nil, nil); /* no overrides */

```

The first parameter to the `GXSetPictureParts` function specifies the picture whose item list you want to edit. The second parameter specifies where you want the editing to occur. In this example, the second parameter is set to 1, which indicates that the new items should be inserted before the first item of the picture. QuickDraw GX draws the items of a picture in order from back to front; therefore, inserting the new items before the existing items ensures that the new items are drawn behind the existing ones.

The third parameter to the `GXSetPictureParts` function specifies how many of the original picture items to remove. In this example, this parameter is set to 0. For examples of removing and replacing picture items, see the next section.

The fourth parameter to the `GXSetPictureParts` function specifies how many new items to insert into the picture, which in this case is 3.

The last four parameters to the `GXSetPictureParts` function specify the shapes, overriding styles, overriding inks, and overriding transforms that make up the new picture items.



## Picture Shapes

Once you have inserted the new shapes into the picture, you can dispose of the shapes (which simply lowers their owner count to 1), and then draw the picture:

```
GXDisposeShape(lawnPolygon);
GXDisposeShape(walkwayPolygon);
GXDisposeShape(chimneyRectangle);

GXDrawShape(housePicture);
```

The resulting picture is shown in Figure 6-21.

**Figure 6-21** A house with a lawn, walkway, and chimney



For more information about the `GXSetPictureParts` function, see page 6-65.

## Removing and Replacing Items in a Picture

You can use the `GXSetPicture` function or the `GXSetPictureParts` function to replace items in a picture.

The `GXSetPicture` function removes every item in a picture and inserts a new list of items. The `GXSetPictureParts` function allows you more control in replacing items. With this function, you can replace a subset of the items in a picture with another set of items. The inserted set does not have to have the same number of items as the replaced set.

## Picture Shapes

As a simple example, you can use the `GXSetPictureParts` function to remove a single item from a picture. Listing 6-6 shows how to use the `GXSetPictureParts` function to remove the chimney, which is item number 3, from the house picture shown in Figure 6-21.

---

**Listing 6-6** Removing an item from a picture

```
GXSetPictureParts(housePicture,  
                 3,    /* start editing at item 3 */  
                 1,    /* remove 1 item */  
                 0,    /* insert 0 items */  
                 nil, /* no shapes to insert */  
                 nil, nil, nil); /* no overrides */
```

The resulting picture is shown in Figure 6-22.

---

**Figure 6-22** A house with chimney removed



You can also use the `GXSetPictureParts` function to replace items in a picture; with a single call to `GXSetPictureParts`, you can remove items from a picture and insert new items into a picture.

## Picture Shapes

Like the sample code in Listing 6-6, the sample code in Listing 6-7 uses the `GXSetPictureParts` function to remove the chimney shape from the house picture. However, this call to the `GXSetPictureParts` function inserts a new chimney into the house picture at the same time.

**Listing 6-7** Replacing one shape with another

```
gxShape newChimneyRectangle;

gxRectangle newChimneyGeometry = {ff(170), ff(50),
                                   ff(180), ff(80)};

.
.
.
newChimneyRectangle = GXNewRectangle(&newChimneyGeometry);
SetShapeCommonColor(newChimneyRectangle, dark + gxGray);

GXSetPictureParts(housePicture,
                  3, /* start editing at item 3 */
                  1, /* remove 1 item */
                  1, /* insert 1 item */
                  &newChimneyRectangle, /* shape to insert */
                  nil, nil, nil); /* no overrides */

GXDisposeShape(newChimneyRectangle);
```

The resulting house picture is shown in Figure 6-23.

**Figure 6-23** A house with the chimney replaced



For more information about the `GXSetPictureParts` function, see page 6-65.

## Using Overriding Styles, Inks, and Transforms

---

As detailed in the previous three sections, QuickDraw GX provides a number of methods for adding items to a picture shape. In particular, you can add items when creating a picture using the `GXNewPicture` function, you can replace every item in an existing picture using the `GXSetPicture` function, and you can replace some of the items in a picture using the `GXSetPictureParts` function. All three of these functions allow you to specify overriding styles, inks, and transforms for the new picture items.

As an example, the code in Listing 6-8 and Listing 6-9 alters the house picture from Listing 6-1 on page 6-28. Listing 6-8 defines a style object, an ink object, and a transform object. Listing 6-9 uses these objects and the `GXSetPicture` function to create a house picture whose items contain overriding styles, inks, and transforms.

---

**Listing 6-8**      Creating style, ink, and transform objects

```

gxShape squarePattern;
gxStyle patternedStyle;
gxInk grayInk;
gxTransform skewedTransform;

const gxRectangle squareGeometry = {ff(0), ff(0),
                                     ff(2), ff(2)};

gxPatternRecord patternRecord;
.
.
.
squarePattern = GXNewRectangle(&squareGeometry);
patternRecord.attributes = gxNoAttributes;
patternRecord.pattern = squarePattern;
patternRecord.u.x = ff(1);
patternRecord.u.y = ff(4);
patternRecord.v.x = ff(3);
patternRecord.v.y = ff(1);

patternedStyle = GXNewStyle();
GXSetStylePattern(patternedStyle, &patternRecord);

grayInk = GXNewInk();
SetInkCommonColor(grayInk, gxGray);

skewedTransform = GXNewTransform();
GXSkewTransform(skewedTransform, -fl(.5), 0, ff(122), ff(110));

```

## Picture Shapes

Listing 6-9 uses the style, ink, and transform objects defined in Listing 6-8, and the `partsOfHouse` array (which is defined in Listing 6-1 on page 6-28) to create a house picture. In this house picture, the main part of the house has an overriding style, the roof has an overriding ink, and the door has an overriding transform.

---

**Listing 6-9** Creating a picture whose items have overriding styles, inks, and transforms

```

gxStyle overridingStyles[3];
gxInk overridingInks[3];
gxTransform overridingTransforms[3];
.
.
.
overridingStyles[0] = patternedStyle;
overridingStyles[1] = nil;
overridingStyles[2] = nil;

overridingInks[0] = nil;
overridingInks[1] = grayInk;
overridingInks[2] = nil;

overridingTransforms[0] = nil;
overridingTransforms[1] = nil;
overridingTransforms[2] = skewedTransform;

housePicture = GXNewShape(gxPictureType);
GXSetPicture(housePicture,
             3,
             partsOfHouse,
             overridingStyles,
             overridingInks,
             overridingTransforms);

```

## Picture Shapes

Once you have added the overriding style, ink, and transform objects to the picture, you can dispose of them, as shown in Listing 6-10. Since these objects are referenced twice (once by your application and once by the house picture), disposing of them lowers their owner counts to 1, but does not free the memory associated with them. When you eventually dispose of the house picture, QuickDraw GX disposes of these objects again and frees their memory.

---

**Listing 6-10** Disposing of overriding style, ink, and transform objects before drawing

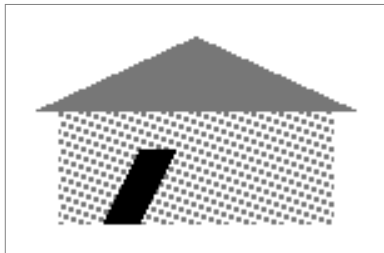
```
GXDisposeShape(squarePattern);
GXDisposeStyle(patternedStyle);
GXDisposeInk(grayInk);
GXDisposeTransform(skewedTransform);

GXDrawShape(housePicture);
```

The resulting picture is shown in Figure 6-24.

---

**Figure 6-24** A house picture with an overriding style, ink, and transform



For more information about overriding styles, inks, and transforms, see “Overriding Styles, Inks, and Transforms” beginning on page 6-8.

For more information about the `GXNewPicture` function, see page 6-57. For more information about the `GXSetPicture` function, see page 6-61.

## Adding Multiple References

---

Multiple items in a single picture can reference the same shape. You can use any of the functions that add items to a picture (`GXNewPicture`, `GXSetPicture`, `GXSetPictureParts`) to add multiple references to a single shape. The example in Listing 6-11 adds four new items to the house picture defined in Listing 6-1 on page 6-28. Each of these items references the same shape—a small, white rectangle. Because four items reference the same rectangle, four instances of this rectangle appear in the picture.

## Picture Shapes

Without overriding transforms, however, all four instances of this rectangle would appear in the same location. Therefore, the sample code in Listing 6-11 creates overriding transforms for three of the four new items.

---

**Listing 6-11** Adding four items that reference the same shape to a house picture

```

gxRectangle windowGeometry = {ff(155), ff(93),
                               ff(160), ff(112)};

gxShape windowRectangle;

gxShape insertedShapes[4];
gxTransform overridingTransforms[4];

windowRectangle = GXNewRectangle(&windowGeometry);
SetShapeCommonColor(windowRectangle, gxWhite);

insertedShapes[0] = windowRectangle;
insertedShapes[1] = windowRectangle;
insertedShapes[2] = windowRectangle;
insertedShapes[3] = windowRectangle;

overridingTransforms[0] = nil;
overridingTransforms[1] = GXNewTransform();
overridingTransforms[2] = GXNewTransform();
overridingTransforms[3] = GXNewTransform();

GXMoveTransform(overridingTransforms[1], ff(7), 0);
GXMoveTransform(overridingTransforms[2], ff(14), 0);
GXMoveTransform(overridingTransforms[3], ff(21), 0);

GXSetPictureParts(housePicture,
                  3, /* where to insert */
                  0, /* how many to replace */
                  4, /* how many to insert */
                  insertedShapes,
                  nil, nil,
                  overridingTransforms);

```

## Picture Shapes

This sample code creates one rectangle shape and three transform objects. Once you insert these objects in the picture, you can dispose of them to lower their owner count to 1, as shown in Listing 6-12. Since these objects are referenced twice (once by your application and once by the house picture), disposing of them lowers their owner counts to 1, but does not free the memory associated with them. When you dispose of the house picture, QuickDraw GX disposes of these objects again and frees their memory.

---

**Listing 6-12** Disposing of the white rectangle and the three transform objects before drawing

```
int count;
.
.
.
for (count = 1; count <= 3 ; count++)
    GXDisposeTransform(overridingTransforms[count]);

GXDisposeShape(windowRectangle);

GXDrawShape(housePicture);
```

The resulting picture is shown in Figure 6-25.

---

**Figure 6-25** A house with four windows



Notice that the sample code in Listing 6-11 creates three separate transform objects because three different transformations are happening to the instances of the window rectangle—the second instance is moved 7 grid points to the right, the third instance is moved 14 grid points to the right, and the fourth instance is moved 21 grid points to the right.

You can specify that QuickDraw GX copy the overriding transforms when adding them to the picture (rather than adding them by reference) by setting the unique items shape attribute, as discussed in the next section.

For more information about adding multiple items referencing the same shape, see “Multiple References” beginning on page 6-10.



## Adding Items With the Unique Items Attribute Set

The unique items shape attribute changes the way in which QuickDraw GX adds shapes to a picture. When you add a shape to a picture that does not have this attribute set, QuickDraw GX copies the reference to the existing shape, inserts this reference into the picture's item list, and increments the owner count of the shape. Similarly, if you specify an overriding style, ink, or transform object for the shape, QuickDraw GX copies the object's reference into the picture's item list and increments the owner count of the object.

However, when you add a shape to a picture that has the unique items attribute set, QuickDraw GX makes a copy of the shape and inserts a reference to the copy in the picture's item list. Similarly, overriding styles, inks, and transforms are also copied.

As an example, Listing 6-13 shows how to use the `GXGetShapeAttributes` and `GXSetShapeAttributes` functions to set the unique items shape attribute of a picture. You must set this attribute before you add any items to a picture; if the picture already contains items, setting this attribute results in an error.

This listing adds four instances of a window rectangle to the house picture from Listing 6-1 on page 6-28. This sample code specifies the same overriding transform for each instance of the window rectangle. However, the overriding transform is moved (with the `GXMoveTransform` function) after each call to the `AddToPicture` library function. Because the house picture has the unique items shape attribute set, QuickDraw GX makes a separate copy of the overriding transform each time a window rectangle is inserted into the picture.

**Listing 6-13** Adding unique items to a picture

```
GXSetShapeAttributes(housePicture,
    GXGetShapeAttributes(housePicture) | gxUniqueItemsShape);
.
.
.
moveToRight = GXNewTransform();

for (count = 0; count <= 3 ; count++) {
    AddToPicture(housePicture,
        windowRectangle,
        nil, nil,
        moveToRight);

    GXMoveTransform(moveToRight, ff(7), 0);
}
```

In this example, the first time that the `AddToPicture` function is called, QuickDraw GX creates a copy of the window rectangle shape and a copy of the overriding transform object, and inserts references to the copies in the item list of the house picture.

## Picture Shapes

The second time that the `AddToPicture` function is called, QuickDraw GX notices that the window rectangle shape has not changed, so it does not make another copy of the window rectangle. Instead, it creates a new item in the house picture that references the previously made copy. However, the overriding transform has changed, so QuickDraw GX makes a new copy of it for the new picture item.

The third and fourth calls to the `AddToPicture` function also create new copies of the overriding transform, but do not create new copies of the window rectangle.

After the code from Listing 6-13 finishes executing, there are a total of two window rectangles—the original one, which is referenced by the `windowRectangle` variable, and the copy, which is referenced four times by the items of the house picture. There are a total of five transform objects—the original one, which is referenced by the `moveToRight` variable, and four separate copies referenced by the four new items of the picture.

Figure 6-26 shows the resulting picture.

**Figure 6-26** A house with four windows and four unique overriding transforms



For more information about the unique items shape attribute, see “Unique Items Shape Attribute” beginning on page 6-15.

## Creating Picture Hierarchies

QuickDraw GX allows the items in a picture shape to reference other picture shapes. You can use any of the functions that allow you to add items to pictures (`GXNewPicture`, `GXSetPicture`, `GXSetPictureParts`) to create picture hierarchies.

When drawing a picture hierarchy, QuickDraw GX concatenates the mapping and clipping information contained in the transform objects (or overriding transform objects) at each level of the hierarchy. As an example, Listing 6-14 shows how QuickDraw GX concatenates mapping information from two levels of a picture hierarchy. In this example, the house picture from Figure 6-21 on page 6-35 is added to another picture as an item with an overriding transform that rotates the house clockwise 90 degrees. In turn, this picture is added as an item to yet another picture, with the same overriding transform.

**Listing 6-14** Creating a picture hierarchy

---

```

gxShape rootPicture, level1Picture;
gxTransform rotateHouse;
.
.
.
rotateHouse = GXNewTransform();
GXRotateTransform(rotateHouse, ff(90), ff(150), ff(100));

level1Picture = GXNewPicture(1,
                             &housePicture,
                             nil, nil,
                             &rotateHouse);
rootPicture = GXNewPicture(1,
                           &level1Picture,
                           nil, nil,
                           &rotateHouse);

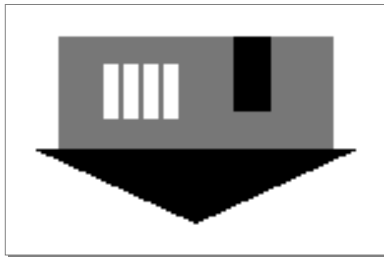
GXDrawShape(rootPicture);

```

When QuickDraw GX draws the root-level picture, it concatenates the information in the two overriding transforms, and draws the house picture rotated clockwise 180 degrees, as shown in Figure 6-27.

---

**Figure 6-27** A house rotated by 90 degrees two times



For more information about picture hierarchies and transform concatenation, see “Picture Hierarchies” beginning on page 6-18 and “Transform Concatenation” beginning on page 6-19.

## Hit-Testing Pictures

As described in “About Hit-Testing Picture Shapes” beginning on page 6-24, QuickDraw GX hit-tests a picture shape by

- hit-testing each item contained in the picture, using the hit-test information in that item’s transform object (or overriding transform object, if the item has one) to determine if the item was hit or not
- finding the hit item that corresponds to the depth you specify
- determining the item to return using the level you specify
- providing information about the item

The criteria you specify includes the depth at which you want to hit-test the picture, and the level of the picture hierarchy at which you want to hit-test.

To illustrate picture hit-testing, Listing 6-15 creates a picture hierarchy using the shapes defined in Listing 6-1 on page 6-28 and Listing 6-4 on page 6-33. This example creates a picture shape that contains two items. The first item is a picture of a lawn and a walkway, and the second item is a picture of a house, roof, and door.

**Listing 6-15**    Creating a picture hierarchy

```
gxShape  groundsPicture, housePicture, entirePicture;

gxShape  partsOfHouse[4];
gxShape  partsOfGrounds[2];
gxShape  partsOfEntirePicture[2];
.
.
.
partsOfGrounds[0] = lawnPolygon;
partsOfGrounds[1] = walkwayPolygon;
groundsPicture = GXNewPicture(2, partsOfGrounds, nil, nil, nil);

partsOfHouse[0] = houseRectangle;
partsOfHouse[1] = roofPolygon;
partsOfHouse[2] = doorRectangle;
housePicture = GXNewPicture(3, partsOfHouse, nil, nil, nil);

partsOfEntirePicture[0] = groundsPicture;
partsOfEntirePicture[1] = housePicture;
entirePicture = GXNewPicture(2, partsOfEntirePicture,
                             nil, nil, nil);
```

Picture Shapes

Figure 6-28 shows the items that make up the grounds picture.

**Figure 6-28**     Grounds picture

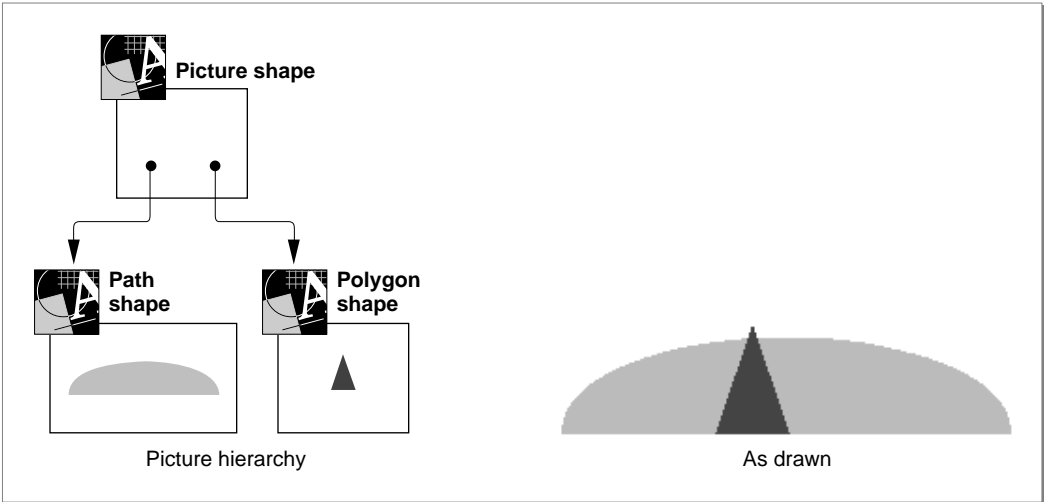


Figure 6-29 shows the items that make up the house picture.

**Figure 6-29**     House picture

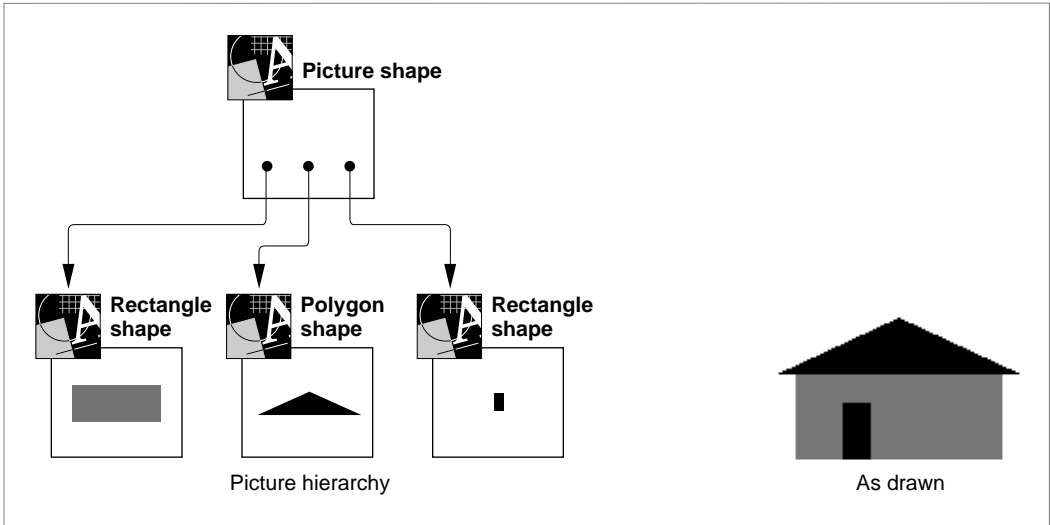
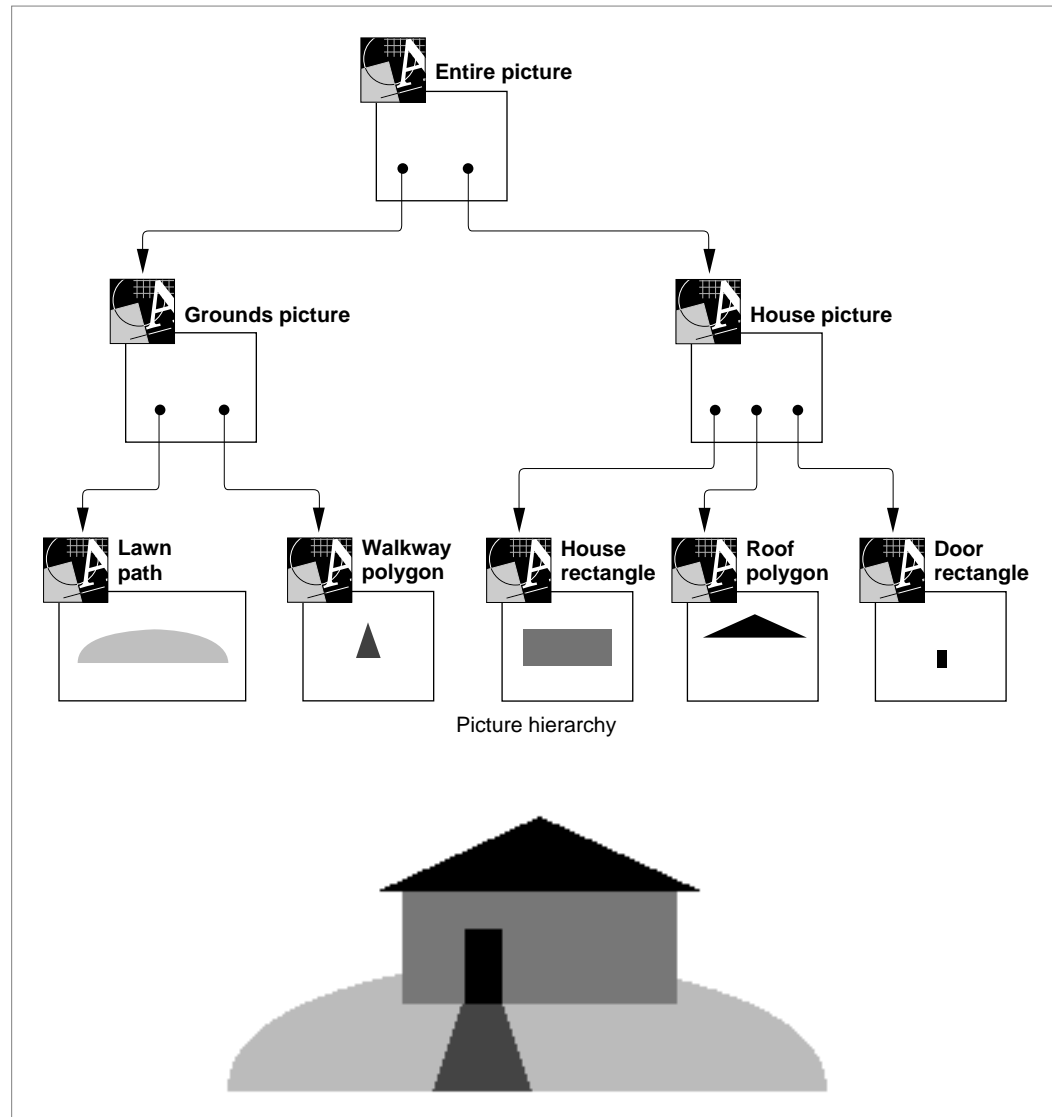


Figure 6-30 shows the entire picture created in Listing 6-15.

**Figure 6-30** Picture containing grounds picture and house picture



You hit-test a picture shape using the function `GXHitTestPicture`. This function takes as its parameters a reference to the picture to hit-test, the test point, an optional hit-test parameters structure, the level at which to hit-test, and the depth at which to hit-test. The sample code in Listing 6-16 shows how to hit-test the picture from Listing 6-15 using a test point of `ff(122), ff(110)`.

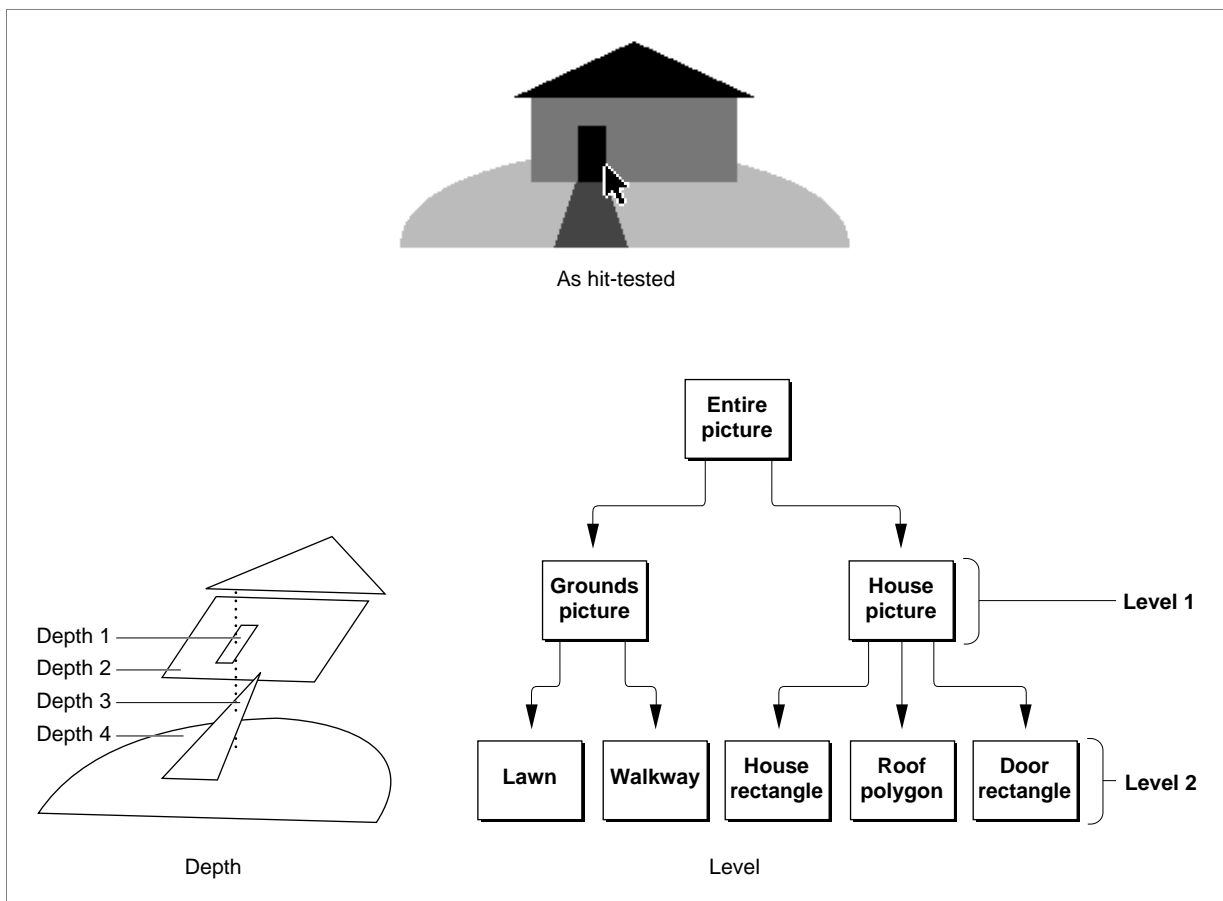
**Listing 6-16** Hit-testing a picture shape

```

gxPoint testPoint = {ff(122), ff(110)};
gxShape hitShape;
long level, depth;
.
.
.
hitShape = GXHitTestPicture(entirePicture, &testPoint, nil,
                             level, depth);

```

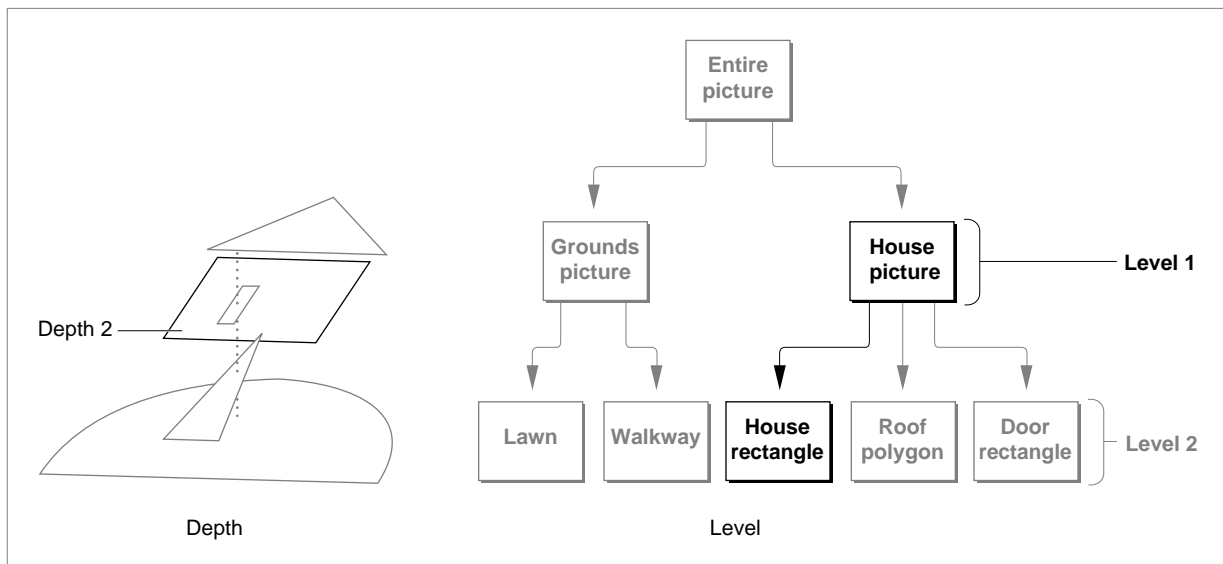
Figure 6-31 shows the location of the test point.

**Figure 6-31** Hit-testing the picture of house and grounds

## Picture Shapes

If you specify a depth of 2, the hit shape is the house rectangle. If you specify a level of 1, QuickDraw GX returns information about the house picture that contains the house rectangle. Figure 6-32 depicts this selection process.

**Figure 6-32** Hit-testing the picture at depth 2 and level 1



The `GXHitTestPicture` function returns a reference to the shape that was hit by the test point. In this example, the test point falls above four separate shapes: the door rectangle, the house rectangle, the walkway polygon, and the lawn path. By varying the values of the level and depth parameters, you can control which shape is returned by the `GXHitTestPicture` function.



Table 6-1 shows which shape is returned for various choices of level and depth.

**Table 6-1** Hit-testing a picture at different depths and levels

Depth	Level	Hit Shape
1	2	Door rectangle
1	1	House picture
2	2	House rectangle
2	1	House picture
3	2	Walkway polygon
3	1	Grounds picture
4	2	Lawn path
4	1	Grounds picture

At depth 1, the returned shape is the frontmost shape that was hit—in this case, the door rectangle, which is at level 2 in the picture hierarchy. If you specify a depth of 1 and a level of 1, the `GXHitTestPicture` function returns the picture that contains the door rectangle—in this case the house picture.

In a similar manner, depth 2 indicates the house rectangle, depth 3 indicates the walkway polygon, and depth 4 indicates the lawn path.

For information about the `GXHitTestPicture` function, see page 6-67.

## Applying Functions Described Elsewhere to Picture Shapes

---

QuickDraw GX provides only a small number of functions that apply exclusively to picture shapes. However, many of the QuickDraw GX functions that you can apply to other types of shapes you can also apply to picture shapes.

The next six sections discuss how functions described elsewhere operate when applied to picture shapes. These sections are

- “Functions That Post Errors or Warnings When Applied to Pictures” on page 6-52, which lists functions that you can apply to other types of shapes but not to picture shapes
- “Shape-Related Functions Applicable to Pictures” on page 6-54, which lists functions that operate on picture shape objects
- “Geometric Operations Applicable to Pictures” on page 6-55, which lists the few geometric operation functions that you can apply to pictures
- “Style-Related Functions Applicable to Pictures” on page 6-55, which discusses how style-related functions apply to pictures
- “Ink-Related Functions Applicable to Pictures” on page 6-56, which discusses how ink-related functions apply to pictures
- “Transform-Related Functions Applicable to Pictures” on page 6-56, which discusses how transform-related functions apply to pictures

### Functions That Post Errors or Warnings When Applied to Pictures

---

Some QuickDraw GX functions that operate on other types of shapes do nothing but post an error or a warning if you try to apply them to a picture shape.

For example, there are a number of shape-related functions and geometric operations that you cannot apply to picture shapes. Table 6-2 lists these functions, which are described in full in Chapter 2, “Geometric Shapes,” and Chapter 4, “Geometric Operations.”

**Table 6-2** Geometric operations that post errors or warnings when applied to pictures

Function name	Error or warning posted
GXBreakShape	graphic_type_does_not_contain_points
GXContainsShape	shape_operator_may_not_be_a_picture
GXCountShapePoints	graphic_type_does_not_contain_points
GXDifferenceShape	shape_operator_may_not_be_a_picture
GXExcludeShape	shape_operator_may_not_be_a_picture
GXGetShapeCenter	illegal_type_for_shape
GXGetShapeDirection	graphic_type_does_not_have_multiple_contours
GXGetShapeLength	shape_does_not_have_length
GXGetShapePoints	graphic_type_does_not_contain_points
GXInsetShape	graphic_type_cannot_be_inset
GXIntersectShape	shape_operator_may_not_be_a_picture
GXInvertShape	shape_cannot_be_inverted
GXReverseDifferenceShape	shape_operator_may_not_be_a_picture
GXReverseShape	contour_out_of_range
GXShapeLengthToPoint	shape_does_not_have_length
GXSetShapePoints	graphic_type_does_not_contain_points
GXTouchesShape	shape_operator_may_not_be_a_picture
GXUnionShape	shape_operator_may_not_be_a_picture

Most of these geometric operations do not apply to picture shapes because a picture's geometry is substantially different from the geometry of a geometric shape.

You can apply a few of the geometric operations to pictures, however. These functions are discussed in "Geometric Operations Applicable to Pictures" beginning on page 6-55.

## Shape-Related Functions Applicable to Pictures

You can apply all of the functions described in the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects* to picture shapes. These functions allow you to

- manipulate the shape object that represents the picture shape; for example, you can copy, clone, cache, compare, and dispose of the picture shape
- set the geometry, shape type, shape fill, and shape attributes of the picture shape
- change the style, ink, and transform objects that are associated with the picture shape
- manipulate the picture shape’s tags and owner count

Table 6-3 gives important picture-related information for a subset of the functions from the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*. Functions described in that chapter that do not appear in this list exhibit the same behavior when applied to picture shapes as they do when applied to other types of shapes.

**Table 6-3** Shape-related functions that exhibit special behavior when applied to pictures

Function name	Action taken
GXCopyToShape	Makes a copy of the picture shape; the items of the new picture shape reference the same shapes as the items of the original picture shape.
GXCopyDeepToShape	Makes a copy of the picture shape, including a complete copy of the entire picture hierarchy.
GXGetShapeSize	Determines the amount of memory currently used by all of the items in the picture.
GXGetShapeFill	Returns the shape’s shape fill, which for picture shapes is always even-odd fill or no fill.
GXSetShapeFill	Sets the shape’s shape fill; you must always set a picture shape’s shape fill to even-odd fill or no fill.
GXSetShapeType	Changes the shape type of the picture shape and converts the shape fill and geometry as appropriate. The resulting shape is a picture with one item—the original shape.

## Geometric Operations Applicable to Pictures

Many geometric operations post errors or warnings when applied to picture shapes, as described in “Functions That Post Errors or Warnings When Applied to Pictures” on page 6-52.

You can, however, apply the remainder of the functions described in Chapter 4, “Geometric Operations,” to picture shapes. Table 6-4 gives important picture-related information for a subset of these functions; the remainder of the geometric operations exhibit the same behavior when applied to picture shapes as they do when applied to other types of shapes.

**Table 6-4**      Geometric operations that exhibit special behavior when applied to pictures

Function name	Action taken
GXGetShapeArea	Returns summed areas of picture items.
GXGetShapeBounds	Returns bounding rectangle of specified item.
GXSetShapeBounds	If the picture’s <code>mapTransformShape</code> shape attribute is set, this function changes the picture’s transform so that the entire picture fits within the specified bounding rectangle. If this attribute is not set, this function posts an error.

## Style-Related Functions Applicable to Pictures

Picture shapes make limited use of their style objects. You may apply to a picture shape any of the functions described in Chapter 3, “Geometric Styles,” (such as `GXSetShapePen`, `GXSetShapeDash`, and so on) to set the properties of a picture’s style object, and you may use the corresponding functions (`GXGetShapePen`, `GXGetShapeDash`, and so on) to examine these properties. However, `QuickDraw GX` ignores these properties when drawing a picture.

## Picture Shapes

## Ink-Related Functions Applicable to Pictures

---

Picture shapes make limited use of their ink objects. You may apply to a picture shape any of the shape-related functions described in the chapter “Ink Objects” of *Inside Macintosh: QuickDraw GX Objects* (such as `GXSetShapeColor`, `GXSetShapeTransfer`) to set the properties of a picture’s ink object, and you may use the corresponding functions (`GXGetShapeColor`, `GXGetShapeTransfer`) to examine these properties. However, QuickDraw GX ignores these properties when drawing a picture.

## Transform-Related Functions Applicable to Pictures

---

Although picture shapes do not make full use of their style and ink objects, they do make full use of their transform objects. You can apply all of the shape-related functions that are described in the chapter “Transform Objects” of *Inside Macintosh: QuickDraw GX Objects* to picture shapes.

In general, you need to be sure that a picture shape’s `gxMapTransformShape` shape attribute is set before applying any of the mapping operations to a picture.

# Picture Shapes Reference

---

## Functions

---

This section describes the functions provided by QuickDraw GX specifically for creating and manipulating picture shapes. With the functions described in this section, you can

- create a new picture shape
- examine and edit the items of a picture shape
- draw pictures
- hit-test pictures

See the section “Applying Functions Described Elsewhere to Picture Shapes” beginning on page 6-52 for information about other QuickDraw GX functions that you can apply to picture shapes.

## Creating Picture Shapes

---

This section describes the `GXNewPicture` function, which you use to create new picture shapes.

## GXNewPicture

---

You can use the `GXNewPicture` function to create a new picture shape.

```
gxShape GXNewPicture(long count, const gxShape shapes[],
                     const gxStyle styles[], const gxInk inks[],
                     const gxTransform transforms[])
```

count	The number of picture items in the new picture shape.
shapes	An array of references to the shapes you want to include in the picture.
styles	An array of references to the style objects you want to use as overriding styles for the picture items. You may provide <code>nil</code> for this parameter if you do not want any overriding styles.
inks	An array of references to the ink objects you want to use as overriding inks for the picture items. You may provide <code>nil</code> for this parameter if you do not want any overriding inks.

## Picture Shapes

`transforms`

An array of references to the transform objects you want to use as overriding transforms for the picture items. You may provide `nil` for this parameter if you do not want any overriding transforms.

*function result* A reference to the newly created picture shape.

## DESCRIPTION

The `GXNewPicture` function creates a new picture shape.

In the `count` parameter, you specify the number of shapes you want to include as items of the picture, and in the `shapes` parameter, you provide references to the shapes.

In the `styles` parameter, you specify references to overriding styles. Each item of this array overrides the style of the corresponding shape in the `shapes` array. For example, the first style you provide in the `styles` array becomes the overriding style for the first shape in the `shapes` array, and so on. Similarly, in the `inks` and `transforms` parameters you specify references to overriding inks and transforms.

You may specify 0 for the `count` parameter and `nil` for the `shapes`, `styles`, `inks`, and `transforms` parameters to create an empty picture—a picture containing no picture items. You may provide `nil` for the `styles`, `inks`, or `transforms` parameters even if you provide shape references in the `shapes` parameter. In this case, the newly created picture shape contains picture items, but those items contain no overriding styles, inks, or transforms.

You may also provide `nil` for an individual item of a `styles`, `inks`, or `transforms` array if you do not want the corresponding picture item to have an overriding style, ink, or transform.

## SPECIAL CONSIDERATIONS

If no error results, the `GXNewPicture` function creates a picture shape; you are responsible for disposing of this shape when you no longer need it. See *Inside Macintosh: QuickDraw GX Objects* for information about creating and disposing of shapes.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

`out_of_memory`  
`parameter_is_nil`  
`shape_is_nil`  
`parameter_out_of_range`



## Picture Shapes

## SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

For an example using this function, see “Creating and Drawing Picture Shapes” beginning on page 6-27.

To draw a picture shape once you’ve created one, use the `GXDrawShape` function, described in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

For information about disposing of picture shapes, see the description of the `GXDisposeShape` function, which is in the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

## Getting and Setting Picture Geometries

---

This section describes the functions you can use to examine or replace the entire geometry of a picture shape—that is, all of the picture items included in the picture.

The `GXGetPicture` function provides references to the shapes contained in a picture geometry and references to their overriding styles, inks, and transforms.

The `GXSetPicture` function replaces references to the shapes contained in a picture geometry and references to their overriding styles, inks, and transforms.

## GXGetPicture

---

You can use the `GXGetPicture` function to obtain references to the shapes contained in a picture and references to their overriding styles, inks, and transforms.

```
long GXGetPicture(gxShape source, gxShape shapes[],
                  gxStyle styles[], gxInk inks[],
                  gxTransform transforms[]);
```

<code>source</code>	A reference to the picture shape whose items you want to examine.
<code>shapes</code>	An array of shape references. On return, this array contains references to the shapes contained in the source picture.
<code>styles</code>	An array of references to style objects. On return, this array contains references to the overriding styles contained in the source picture.
<code>inks</code>	An array of references to ink objects. On return, this array contains references to the overriding inks contained in the source picture.

## Picture Shapes

`transforms`

An array of references to transform objects. On return, this array contains references to the overriding transforms contained in the source picture.

*function result* The total number of items in the source picture.

## DESCRIPTION

If you provide arrays for the `shapes`, `styles`, `inks`, and `transforms` parameters, this function copies the references to shapes, styles, inks, and transforms from the picture's geometry into these arrays. However, you may provide `nil` for any of these parameters to indicate that you do not want to obtain the corresponding references.

Typically, you call this function twice. The first time you specify `nil` for all of the array parameters and use the function result to determine the number of picture items, which you can use to allocate arrays large enough to contain the shape, style, ink, and transform references. Then you call the function a second time to determine the actual references.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>cannot_set_item_shapes_to_nil</code>	(debugging version)
<code>cannot_use_original_item_shapes_when_growing_picture</code>	(debugging version)

**Warnings**

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>picture_expected</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>picture_cannot_contain_itself</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)
<code>cannot_dispose_default_shape</code>	(debugging version)
<code>cannot_dispose_default_style</code>	(debugging version)
<code>cannot_dispose_default_ink</code>	(debugging version)
<code>cannot_dispose_default_transform</code>	(debugging version)
<code>cannot_dispose_default_colorProfile</code>	(debugging version)

SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

For an example using this function, see “Getting and Setting Picture Geometries” beginning on page 6-31.

To examine a subset of the items in a picture geometry, use the `GXGetPictureParts` function, which is described on page 6-63.

To replace the information in the geometry of a picture shape, use the `GXSetPicture` function, which is described in the next section.

GXSetPicture

You can use the `GXSetPicture` function to replace the information in the geometry of a picture shape.

```
void GXSetPicture(gxShape target, long count,
                  const gxShape shapes[], const gxStyle styles[],
                  const gxInk inks[],
                  const gxTransform transforms[]);
```

target	A reference to the picture shape whose geometry you want to replace.
count	The number of picture items in the new picture geometry.
shapes	An array of references to the shapes to include in the new picture geometry.
styles	An array of references to the styles you want to use as overriding styles in the new picture geometry. You may provide <code>nil</code> for this parameter if you do not want to change the existing overriding styles. You may provide the <code>gxSetToNil</code> constant to remove all of the existing overriding styles.
inks	An array of references to the inks you want to use as overriding inks in the new picture geometry. You may provide <code>nil</code> for this parameter if you do not want to change the existing overriding inks. You may provide the <code>gxSetToNil</code> constant to remove all of the existing overriding inks.
transforms	An array of references to the transforms you want to use as overriding transforms in the new picture geometry. You may provide <code>nil</code> for this parameter if you do not want to change the existing overriding transforms. You may provide the <code>gxSetToNil</code> constant to remove all of the existing overriding transforms.

## Picture Shapes

## DESCRIPTION

The `GXSetPicture` function replaces the geometry of the picture shape object referenced by the `target` parameter with a new geometry. To maintain correct owner counts, this function disposes of the shapes, styles, inks, and transforms referenced by the items of the original picture geometry.

In the `count` parameter, you specify the number of shapes in the new picture geometry, and in the `shapes` parameter you provide references to the shapes.

In the `styles` parameter, you specify references to the styles to use as overriding styles in the new picture geometry. Each item of this array overrides the style of the corresponding shape in the `shapes` array. For example, the first style you provide in the `styles` array becomes the overriding style for the first shape in the `shapes` array, and so on. Similarly, in the `inks` and `transforms` parameters you specify references to overriding inks and transforms.

You may specify 0 for the `count` parameter and `nil` for the `shapes`, `styles`, `inks`, and `transforms` parameters to create an empty picture—a picture containing no picture items. You may provide the `gxSetToNil` constant for the `styles`, `inks`, or `transforms` parameters even if you provide shape references in the `shapes` parameter. In this case, the newly created picture shape contains picture items, but those items contain no overriding styles, inks, or transforms, respectively.

You may also provide `nil` for an individual item of a `styles`, `inks`, or `transforms` array if you do not want the corresponding picture item to have an overriding style, ink, or transform.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>cannot_set_item_shapes_to_nil</code>	(debugging version)
<code>cannot_use_original_item_shapes_when_growing_picture</code>	(debugging version)

**Warnings**

<code>picture_expected</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>picture_cannot_contain_itself</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)
<code>cannot_dispose_default_shape</code>	(debugging version)
<code>cannot_dispose_default_style</code>	(debugging version)
<code>cannot_dispose_default_ink</code>	(debugging version)
<code>cannot_dispose_default_transform</code>	(debugging version)
<code>cannot_dispose_default_colorProfile</code>	(debugging version)

## SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

To examine the items of a picture geometry, use the `GXGetPicture` function, which is described on page 6-59.

To replace a subset of the items in a picture geometry, use the `GXSetPictureParts` function, which is described on page 6-65.

For information about disposing of shapes, see the chapter “Shape Objects” of *Inside Macintosh: QuickDraw GX Objects*.

## Editing Picture Parts

---

This section describes the functions you can use to examine and replace specific items within a picture geometry.

The `GXGetPictureParts` function allows you to obtain information about a specified subset of the picture items contained in a picture geometry.

The `GXSetPictureParts` function allows you to replace a subset of the picture items in a picture geometry with new picture items.

## GXGetPictureParts

---

You can use the `GXGetPictureParts` function to obtain information about a specified subset of a picture’s items.

```
long GXGetPictureParts(gxShape source, long index, long count,
                      gxShape shapes[], gxStyle styles[],
                      gxInk inks[], gxTransform transforms[])
```

<code>source</code>	A reference to the picture shape whose items you want to examine.
<code>index</code>	The number of the first picture item you want to examine.
<code>count</code>	The total number of items you want to examine. You may supply the <code>gxSelectToEnd</code> constant (–1) to indicate that you want to examine all picture items (starting with the picture item indicated by the <code>index</code> parameter.)
<code>shapes</code>	An array of shape references. On return, this array contains references to the specified shapes contained in the source picture.
<code>styles</code>	An array of style references. On return, this array contains references to the overriding styles corresponding to the returned shapes.
<code>inks</code>	An array of ink references. On return, this array contains references to the overriding inks corresponding to the returned shapes.

## Picture Shapes

`transforms`

An array of transform references. On return, this array contains references to the overriding transforms corresponding to the returned shapes.

*function result* The total number of items returned.

## DESCRIPTION

The `GXGetPictureParts` function extracts information from a subset of the picture items in the picture shape referenced by the `source` parameter. You specify which picture items using the `index` and `count` parameters. The `index` parameter, which must have a value of 1 or greater, indicates the first picture item you want to examine. The `count` parameter indicates how many items you want to examine.

You provide arrays to hold the returned information in the `shapes`, `styles`, `inks`, and `transforms` parameters. In the `shapes` array, the `GXGetPictureParts` function returns references to the shapes that correspond to the picture items you specified with the `index` and `count` parameters. In the `styles`, `inks`, and `transforms` arrays, this function returns references to the overriding styles, inks, and transforms for the specified picture items. You may provide `nil` for any of the array parameters to indicate that you do not want to obtain the corresponding references.

This function returns as its function result the number of picture items returned. Typically, this value is the same as the value you provide for the `count` parameter.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)

**Warnings**

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>picture_expected</code>	(debugging version)

## SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

## Picture Shapes

To examine all of the items in a picture geometry, use the `GXGetPicture` function, which is described on page 6-59.

To replace a subset of the items in a picture geometry, use the `GXSetPictureParts` function, which is described in the next section.

## GXSetPictureParts

---

You can use the `GXSetPictureParts` function to add, remove, or replace a range of picture items in a picture shape's geometry.

```
void GXSetPictureParts(gxShape target, long index, long oldCount,
                      long newCount, const gxShape shapes[],
                      const gxStyle styles[],
                      const gxInk inks[],
                      const gxTransform transforms[]);
```

<code>target</code>	A reference to the picture shape whose picture item list you want to alter.
<code>index</code>	The number of the first picture item you want to replace.
<code>oldCount</code>	The total number of picture items you want to replace. A value of 0 indicates that you want to insert new picture items before the existing picture item indicated by the <code>index</code> parameter, rather than replace items. You may supply the <code>gxSelectToEnd</code> constant (-1) to indicate that you want to replace all picture items (starting with the picture item indicated by the <code>index</code> parameter.)
<code>newCount</code>	The total number of new picture items to insert in the picture. A value of 0 specifies that you do not want to insert new items into the picture; instead, the existing items you specified with the <code>index</code> and <code>oldCount</code> parameters are removed.
<code>shapes</code>	An array of references to the shapes to include as the new picture items in the new picture geometry.
<code>styles</code>	An array of references to the style objects you want to use as overriding styles in the new picture geometry. You may provide <code>gxSetToNil</code> for this parameter if you do not want any overriding styles.
<code>inks</code>	An array of references to the ink objects you want to use as overriding inks in the new picture geometry. You may provide <code>gxSetToNil</code> for this parameter if you do not want any overriding inks.
<code>transforms</code>	An array of references to the transform objects you want to use as overriding transforms in the new picture geometry. You may provide <code>gxSetToNil</code> for this parameter if you do not want any overriding transforms.

## Picture Shapes

## DESCRIPTION

The `GXSetPictureParts` function allows you to insert new picture items in a picture, to remove picture items from a picture, or to replace picture items with new picture items. In any of these three cases, the `target` parameter specifies the picture to be modified, the `oldCount` parameter specifies the number of items to remove, the `newCount` parameter specifies the number of items to add, and the `shapes`, `styles`, `inks`, and `transforms` parameters specify the information for the new picture items.

- To insert picture items, set the `oldCount` parameter to 0. Use the `index` parameters to specify where to add the new picture items. (This function inserts the new picture items before the existing item you specify with the `index` parameter. For example, if you specify 1 for this parameter, the new picture items are inserted before the first item of the existing picture item list.)
- To remove picture items, set the `newCount` parameter to 0 and the `shapes`, `styles`, `inks`, and `transforms` parameters to `nil`. Use the `index` and `oldCount` parameters to specify which picture items to remove.
- To replace picture items, use the `index` and `oldCount` parameters to specify the existing picture items to remove and use the `newCount`, `shapes`, `styles`, `inks`, and `transforms` parameters to specify the new picture items to insert in their place.

To maintain correct owner counts, this function clones the inserted `shapes`, `styles`, `inks`, and `transforms`, and disposes of any replaced `shapes`, `styles`, `inks`, and `transforms`.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>cannot_set_item_shapes_to_nil</code>	(debugging version)
<code>cannot_use_original_item_shapes_when_growing_picture</code>	(debugging version)

**Warnings**

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>picture_expected</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>picture_cannot_contain_itself</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)
<code>cannot_dispose_default_shape</code>	(debugging version)
<code>cannot_dispose_default_style</code>	(debugging version)
<code>cannot_dispose_default_ink</code>	(debugging version)
<code>cannot_dispose_default_transform</code>	(debugging version)
<code>cannot_dispose_default_colorProfile</code>	(debugging version)



## SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

For examples using this function, see “Removing and Replacing Items in a Picture” beginning on page 6-35.

To extract information from a subset of the items contained in a picture shape’s geometry, use the `GXGetPictureParts` function, which is described on page 6-63.

To replace every item in a picture geometry, use the `GXSetPicture` function, which is described on page 6-61.

For information about disposing of shapes, see the chapter “Shape Objects” in *Inside Macintosh: QuickDraw GX Objects*.

## Drawing Pictures

---

QuickDraw GX provides two methods of drawing a picture:

- You can create a picture shape (by calling the `GXNewPicture` function, by copying an existing picture shape, and so on) and use the `GXDrawShape` function to draw the picture.
- You can create an array of shape references, and arrays of references to overriding styles, inks, and transforms, and use the `GXDrawPicture` function to draw the corresponding picture.

In general, you should use the `GXDrawShape` function to draw any QuickDraw GX graphic, including picture shapes. In fact, the `GXDrawPicture` function creates a temporary picture shape, uses the `GXDrawShape` function to draw it, and then disposes of it. The `GXDrawShape` function is described in the “Shape Objects” chapter of *Inside Macintosh: QuickDraw GX Objects*.

You would typically use the `GXDrawPicture` function only in simple situations—for example, if you knew you wanted to draw a particular picture only once.

## GXDrawPicture

---

You can use the `GXDrawPicture` function to draw a picture without encapsulating the items of the picture geometry in a picture shape.

```
void gxDrawPicture(long count, const gxShape shapes[],
                  const gxStyle styles[], const gxInk inks[],
                  const gxTransform transforms[]);
```

count	The number of picture items in the new picture shape.
shapes	An array of references to the shapes you want to draw.

## Picture Shapes

<code>styles</code>	An array of references to the style objects you want to use to override the styles of the shapes specified in the <code>shapes</code> parameter. You may provide <code>nil</code> for this parameter if you do not want any overriding styles.
<code>inks</code>	An array of references to the ink objects you want to use to override the inks of the shapes specified in the <code>shapes</code> parameter. You may provide <code>nil</code> for this parameter if you do not want any overriding inks.
<code>transforms</code>	An array of references to the transform objects you want to use to override the transforms of the shapes specified in the <code>shapes</code> parameter. You may provide <code>nil</code> for this parameter if you do not want any overriding transforms.

## DESCRIPTION

The `GXDrawPicture` function allows you to draw a picture without having to create a picture shape yourself. Instead, you specify the items of a picture geometry using the `shapes`, `styles`, `inks`, and `transforms` parameters.

The `GXDrawPicture` function creates a temporary picture shape using the values specified in these arrays, and draws the picture shape using the `GXDrawShape` function. The `GXDrawPicture` function calls the `GXNewPicture` function to create the temporary picture shape.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

<code>out_of_memory</code>	
<code>shape_is_nil</code>	
<code>inconsistent_parameters</code>	(debugging version)
<code>parameter_out_of_range</code>	(debugging version)
<code>index_is_less_than_one</code>	(debugging version)
<code>count_is_less_than_one</code>	(debugging version)
<code>cannot_set_item_shapes_to_nil</code>	(debugging version)
<code>cannot_use_original_item_shapes_when_growing_picture</code>	(debugging version)

**Warnings**

<code>index_out_of_range</code>	
<code>count_out_of_range</code>	
<code>picture_expected</code>	(debugging version)
<code>shape_access_not_allowed</code>	(debugging version)
<code>picture_cannot_contain_itself</code>	(debugging version)
<code>cannot_dispose_locked_tag</code>	(debugging version)
<code>cannot_dispose_default_shape</code>	(debugging version)
<code>cannot_dispose_default_style</code>	(debugging version)
<code>cannot_dispose_default_ink</code>	(debugging version)
<code>cannot_dispose_default_transform</code>	(debugging version)
<code>cannot_dispose_default_colorProfile</code>	(debugging version)

## SEE ALSO

For information about picture items and their overriding styles, inks, and transforms, see “About Picture Shapes” beginning on page 6-3.

For an example using this function, see “Creating and Drawing Picture Shapes” beginning on page 6-27.

To encapsulate a picture geometry in a picture shape, use the `GXNewPicture` function, described on page 6-57.

To draw a picture shape, use the `DrawShape` function, described in the “Shape Objects” chapter of *Inside Macintosh: QuickDraw GX Objects*.

## Hit-Testing Pictures

---

This section describes the `GXHitTestPicture` function. To hit-test a picture, this function

- hit-tests each shape contained in the picture
- compiles a list of shapes that were hit
- selects one of the shapes using criteria you provide
- provides information about the shape in the picture that was hit

For more information about how QuickDraw GX hit-tests shapes, see the chapter “Shape Objects” and the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

## GXHitTestPicture

---

You can use the `GXHitTestPicture` function to determine whether a test point hits a picture shape and to discover which shape in the picture hierarchy is hit.

```
gxShape GXHitTestPicture(gxShape target, const gxPoint *test,
                        gxHitTestInfo *result, long level,
                        long depth);
```

<code>target</code>	A reference to the picture shape to hit-test.
<code>test</code>	A pointer to a <code>gxPoint</code> structure. The <code>GXHitTestPicture</code> function determines whether the location specified by this point hits the target picture.
<code>result</code>	A pointer to a <code>gxHitTestInfo</code> structure. On return, this structure contains information identifying the part of the target picture that was hit by the test point.

## Picture Shapes

<code>level</code>	A level in the picture hierarchy. This parameter, along with the <code>depth</code> parameter, is used to determine which shape in the picture to return as the function result. You must provide a nonnegative value for this parameter. A value of 0 indicates you want the item at the lowest level of the hierarchy.
<code>depth</code>	A shape depth in the picture as drawn. This parameter, along with the <code>level</code> parameter, is used to determine which shape in the picture to return as the function result. You must provide a nonnegative value for this parameter. A value of 0 indicates you want the hit item at the lowest depth.
<i>function result</i>	A reference to the shape (at the specified shape depth and hierarchy level) hit by the test point. The function result is <code>nil</code> if no shape was hit that satisfies the criteria.

## DESCRIPTION

The `GXHitTestPicture` function compares the point indicated by the `test` parameter with each shape in the picture referenced by the `target` parameter. To determine whether the test point hits a shape, this function uses the hit-test parameters contained in that shape's transform object, or contained in the overriding transform if there is one.

If the target picture contains shapes that overlap when drawn, more than one shape might be hit by the test point. The function uses the `depth` parameter to select which of these shapes is the hit shape. If you set this parameter to 1, the function selects the frontmost shape as the hit shape. If you set this parameter to 2, the function selects the shape immediately behind the frontmost shape as the hit shape, and so on.

Before returning a reference to the hit shape, this function examines how deep into the target picture's hierarchy the hit shape is. If the hit shape is deeper into the hierarchy than the level indicated by the `level` parameter, this function does not return a reference to the hit shape. Instead, it returns a reference to the subpicture at the appropriate level of the target picture's hierarchy that contains the hit shape.

For example, if the hit shape is at level 2 of the picture hierarchy—that is, it is an item of a picture which is an item of the target picture—then specifying a value of 2 for the `level` parameter causes the function to return a reference to the shape as the function result. However, if you specify a value of 1 for the `level` parameter, the function returns a reference to the picture that contains the hit shape, rather than a reference to the hit shape itself. Specifying a level of 0 indicates you want the item at the lowest level of the picture hierarchy.

## Picture Shapes

This function also returns information in the `GxHitTestInfo` structure pointed to by the `result` parameter:

- The `what` field indicates which shape part of the hit shape was hit by the test point.
- The `index` field indicates the index of the geometric point hit by the test point.
- The `distance` field indicates the distance of the test point from the shape part hit.
- The `which` field contains a reference to the hit shape.
- The `containerPicture` field contains a reference to the picture that contains the hit shape.
- The `containerIndex` field indicates the index of the hit shape within the container picture.
- The `totalIndex` field indicates the overall index of the hit shape within the target picture.

For more information about the `GxHitTestInfo` structure, see the chapter “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

## ERRORS, WARNINGS, AND NOTICES

**Errors**

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

**Warnings**

`character_substitution_took_place`  
`font_substitution_took_place`  
`picture_expected` (debugging version)  
`unable_to_traverse_open_contour_that_starts_or_ends_off_the_curve` (debugging version)

## SEE ALSO

For more information about hit-testing shapes, see the chapters “Shape Objects” and “Transform Objects” in *Inside Macintosh: QuickDraw GX Objects*.

For examples using this function, see “About Hit-Testing Picture Shapes” beginning on page 6-24.

## Summary of Picture Shapes

---

### Functions

---

#### Creating Picture Shapes

```
gxShape GXNewPicture      (long count, const gxShape shapes[],
                           const gxStyle styles[], const gxInk inks[],
                           const gxTransform transforms[])
```

#### Getting and Setting Picture Geometries

```
long GXGetPicture      (gxShape source, gxShape shapes[],
                        gxStyle styles[], gxInk inks[],
                        gxTransform transforms[]);

void GXSetPicture      (gxShape target, long count,
                        const gxShape shapes[], const gxStyle styles[],
                        const gxInk inks[],
                        const gxTransform transforms[]);
```

#### Editing Picture Parts

```
long GXGetPictureParts  (gxShape source, long index, long count,
                        gxShape shapes[], gxStyle styles[],
                        gxInk inks[], gxTransform transforms[])

void GXSetPictureParts  (gxShape target, long index, long oldCount,
                        long newCount, const gxShape shapes[],
                        const gxStyle styles[],
                        const gxInk inks[],
                        const gxTransform transforms[]);
```

#### Drawing Pictures

```
void gxDrawPicture      (long count, const gxShape shapes[],
                        const gxStyle styles[], const gxInk inks[],
                        const gxTransform transforms[]);
```

#### Hit-Testing Pictures

```
gxShape GXHitTestPicture (gxShape target, const gxPoint *test,
                        gxHitTestInfo *result, long level,
                        long depth);
```