CHAPTER 6

Offscreen Graphics Worlds

Contents

This chapter describes QuickDraw routines and data structures that your application can use to create offscreen graphics worlds. Whether your application uses Color QuickDraw or basic QuickDraw, you should read this chapter to improve your application's appearance and performance when it draws onscreen images.

Read this chapter to learn how to set up and use an **offscreen graphics world**—a sophisticated environment for preparing complex color or black-and-white images before displaying them on the screen. Offscreen graphics worlds are available on all Macintosh computers that support System 7.

You can use all of the drawing operations described in the chapters "QuickDraw Drawing" and "Color QuickDraw" in this book to create images in an offscreen graphics world. After preparing an image in an offscreen graphics world, you can use the `CopyBits`, `CopyMask`, or `CopyDeepMask` procedure to move the image to an onscreen color graphics port or basic graphics port. Color graphics ports are described in the chapter "Color QuickDraw," and basic graphics ports are described in the chapter "Basic QuickDraw" in this book.

To support your application in preparing an offscreen image for display on a screen, Color QuickDraw by default uses the screen's `GDevice` record to define the pixel depth and color table for the offscreen graphics world. The `GDevice` record is described in the chapter "Graphics Devices" in this book.

Your application can treat an offscreen graphics world as a virtual screen where your application has complete control over its drawing environment, and on which your application can draw a complex image where the user can't see the various steps your application must take before completing it. For example, your application can use QuickDraw drawing routines to build a complex color image in an offscreen graphics world; then, after building the image, your application can use `CopyBits` to copy it quickly to the screen. This prevents the choppiness that could occur if your application were to construct the image directly in a color graphics port on the screen.

About Offscreen Graphics Worlds

An offscreen graphics world is defined by a private data structure that, in Color QuickDraw, contains a `CGrafPort` record and its handles to associated `PixMap` and `ColorTable` records. The offscreen graphics world also contains a reference to a `GDevice` record and other state information. On computers lacking Color QuickDraw, `GWorldPtr` points to an extension of the `GrafPort` record. An offscreen graphics world for a basic QuickDraw system does not contain a reference to a `GDevice` record, but it does support a special type of 1-bit pixel map. When your application uses the `NewGWorld` function to create an offscreen world, `NewGWorld` returns a pointer of type `GWorldPtr`, which your application uses to refer to the offscreen graphics world. This pointer is defined as follows:

```
TYPE GWorldPtr = CGrafPtr;
```

Offscreen graphics worlds have two primary purposes.

- They prevent any other application or desk accessory from changing your drawing environment while your application is creating an image. An offscreen graphics world that you create cannot be modified by any other application.
- They increase onscreen drawing speed and visual smoothness. For example, suppose your application draws multiple graphics objects in a window, and then needs to update part of that window. If your image is very complex, your application can copy it from an offscreen graphics world onto the screen faster than it can repeat all of the steps necessary to redraw the image onscreen. At the same time, your application avoids the choppy visual effect that arises from drawing a large number of separate objects.

The term *offscreen graphics world* implies that you prepare an image on the global coordinate plane somewhere outside the boundary rectangles for the user's screens. While this is possible, you more typically use the coordinates of the port rectangle for an onscreen window when preparing your "offscreen" image; until you copy the image to the onscreen window, the image in the offscreen graphics world is drawn into a part of memory not used by the video device and therefore remains hidden from the user.

Using Offscreen Graphics Worlds

To use an offscreen graphics world, you generally

- use the `NewGWorld` function to create an offscreen graphics world
- use the `GetGWorld` procedure to save the onscreen graphics port for the active window
- use the `SetGWorld` procedure to make the offscreen graphics world the current graphics port
- use the `LockPixels` function to prevent the base address for the offscreen pixel image from moving while you draw into it or copy from it
- use the `EraseRect` procedure to initialize the offscreen pixel image
- use the basic QuickDraw and Color QuickDraw routines described elsewhere in this book to draw into the offscreen graphics world
- use the `SetGWorld` procedure to restore the active window as the current graphics port
- use the `CopyBits` procedure to copy the image from the offscreen graphics world into the active window
- use the `UnlockPixels` procedure to allow the Memory Manager to move the base address for the offscreen pixel image
- use the `DisposeGWorld` procedure to dispose of all the memory allocated for an offscreen graphics world when you no longer need its offscreen pixel image

If you want to use the CopyMask or CopyDeepMask procedure, you can create another offscreen graphics world and draw your mask into that offscreen world.

These tasks are explained in greater detail in the rest of this chapter.

Before using the routines described in this chapter, you must use the InitGraf procedure, described in the chapter "Basic QuickDraw," to initialize QuickDraw. You should also ensure the availability of these routines by checking for the existence of System 7 or Color QuickDraw.

You can make sure that offscreen graphics world routines are available on any computer—including one supporting only basic QuickDraw—by using the Gestalt function with the gestaltSystemVersion selector. Test the low-order word in the response parameter; if the value is $0700 or greater, then offscreen graphics worlds are supported.

You can also test for offscreen graphics world support by using the Gestalt function with the gestaltQuickDrawVersion selector. If the value returned in the response parameter is equal to or greater than the value of the constant gestalt32BitQD, then the system supports both Color QuickDraw and offscreen graphics worlds.

You can use the Gestalt function with the gestaltQuickDrawVersion selector to determine whether the user's system supports offscreen color pixel maps. If the bit indicated by the gestaltHasDeepGWorlds constant is set in the response parameter, then offscreen color pixel maps are available.

For more information about the Gestalt function, see the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities*.

Creating an Offscreen Graphics World

You create an offscreen graphics world with the NewGWorld function. It creates a new offscreen graphics port, a new offscreen pixel map, and (on computers that support Color QuickDraw) either a new offscreen GDevice record or a link to an existing one. It returns a data structure of type GWorldPtr by which your application refers to your new offscreen graphics world. Listing 6-1 illustrates how to create an offscreen graphics world.

Listing 6-1 Using a single offscreen graphics world and the CopyBits procedure

```
PROCEDURE MyPaintRectsThruGWorld (wp: WindowPtr);
   VAR
      origPort:              GrafPtr;
      origDev:               GDHandle;
      myErr:                 QDErr;
      myOffGWorld:           GWorldPtr;
      offPixMapHandle:       PixMapHandle;
      good:                  Boolean;
      sourceRect, destRect:  Rect;
```

```
BEGIN
   GetGWorld(origPort, origDev);          {save window's graphics port}
   myErr := NewGWorld(myOffGWorld, 0,     {create offscreen graphics world, }
                     wp^.portRect,        { using window's port rectangle}
                     NIL, NIL, []);
   IF (myOffGWorld = NIL) OR (myErr <> noErr) THEN
      ; {handle error here}
   SetGWorld(myOffGWorld, NIL);  {make offscreen world the current port}
   offPixMapHandle := GetGWorldPixMap(myOffGWorld);   {get handle to }
   good := LockPixels(offPixMapHandle); { offscreen pixel image and lock it}
   IF NOT good THEN
      ; {handle error here}
   EraseRect(myOffGWorld^.portRect);         {initialize its pixel image}
   MyPaintAndFillColorRects; {paint a blue rectangle, fill a green rectangle}
   SetGWorld(origPort, origDev);          {make window the current port}
      {next, for CopyBits, create source and destination rectangles that }
      { exclude scroll bar areas}
   sourceRect := myOffGWorld^.portRect;   {use offscreen portRect for source}
   sourceRect.bottom := myOffGWorld^.portRect.bottom - 15;
   sourceRect.right := myOffGWorld^.portRect.right - 15;
   destRect := wp^.portRect;        {use window portRect for destination}
   destRect.bottom := wp^.portRect.bottom - 15;
   destRect.right := wp^.portRect.right - 15;
      {next, use CopyBits to transfer the offscreen image to the window}
   CopyBits(GrafPtr(myOffGWorld)^.portBits,  {coerce graphics world's }
                                             { PixMap to a BitMap}
            GrafPtr(wp)^.portBits,     {coerce window's PixMap to a BitMap}
            sourceRect, destRect, srcCopy, NIL);
   IF QDError <> noErr THEN
      ; {likely error is that there is insufficient memory}
   UnlockPixels(offPixMapHandle);         {unlock the pixel image}
   DisposeGWorld(myOffGWorld);            {dispose of offscreen world}
END;
```

When you use `NewGWorld`, you can specify a pixel depth, a boundary rectangle (which also becomes the port rectangle), a color table, a `GDevice` record, and option flags for memory allocation for the offscreen graphics world. Typically, however, you pass 0 as the pixel depth, a window's port rectangle as the offscreen world's boundary rectangle, `NIL` for both the color table and `GDevice` record, and an empty set ([ ]) in your Pascal code or 0 in your C code for the option flags. This provides your application with the default behavior of `NewGWorld`, and it supports computers running only basic QuickDraw. This also allows QuickDraw to optimize the `CopyBits`, `CopyMask`, and `CopyDeepMask` procedures when your application copies the image you create in an offscreen graphics world into the window's port rectangle.

When creating an offscreen graphics world, if you specify 0 as the pixel depth, the port rectangle for a window as the boundary rectangle, and no option flags, the `NewGWorld` function

- uses the pixel depth of the screen with the greatest pixel depth from among all screens intersected by the window
- aligns the pixel image to the screen for optimum performance for the `CopyBits` procedure
- uses the color table and `GDevice` record for the screen with the greatest pixel depth from among all screens intersected by the window
- allocates an unpurgeable base address for the offscreen pixel image in your application heap
- allows graphics accelerators to cache the offscreen pixel image

The application-defined routine `MyPaintRectsThruGWorld` in Listing 6-1, for example, specifies the default behavior for `NewGWorld`. The `MyPaintRectsThruGWorld` routine dereferences the window pointer passed in the `wp` parameter to obtain a window's port rectangle, which `MyPaintRectsThruGWorld` passes to `NewGWorld` as the boundary and port rectangle for the offscreen graphics world.

Setting the Graphics Port for an Offscreen Graphics World

Before drawing into the offscreen graphics port created in Listing 6-1 on page 6-5, `MyPaintRectsThruGWorld` saves the graphics port for the front window by calling the `GetGWorld` procedure, which saves the current graphics port and its `GDevice` record. Then `MyPaintRectsThruGWorld` makes the offscreen graphics world the current port by calling the `SetGWorld` procedure. After drawing into the offscreen graphics world, `MyPaintRectsThruGWorld` also uses `SetGWorld` to restore the active window as the current graphics port.

Instead of using the `GetPort` and `SetPort` procedures for saving and restoring offscreen graphics worlds, you must use `GetGWorld` and `SetGWorld`; you can also use `GetGWorld` and `SetGWorld` for saving and restoring color and basic graphics ports.

Drawing Into an Offscreen Graphics World

You must call the `LockPixels` function before drawing to or copying from an offscreen graphics world. The `LockPixels` function prevents the base address for an offscreen pixel image from being moved while you draw into it or copy from it.

If the base address for an offscreen pixel image hasn't been purged by the Memory Manager or if its base address is not purgeable, `LockPixels` returns `TRUE` as its function result, and your application can draw into or copy from the offscreen pixel image. However, if the base address for an offscreen pixel image has been purged, `LockPixels` returns `FALSE` to indicate that you cannot draw into it or copy from it. (At that point, your application should either call the `UpdateGWorld` function to reallocate the offscreen pixel image and then reconstruct it, or draw directly into an onscreen graphics port.)

After setting the offscreen graphics world to the current graphics port, `MyPaintRectsThruGWorld` in Listing 6-1 on page 6-5 uses the `GetGWorldPixMap` function to get a handle to an offscreen pixel map. Passing this handle to the `LockPixels` function, `MyPaintRectsThruGWorld` locks the memory for the offscreen pixel image in preparation for drawing into its pixel map.

IMPORTANT

On a system running only basic QuickDraw, the `GetGWorldPixMap` function returns the handle to a 1-bit pixel map that your application can supply as a parameter to `LockPixels` and the other routines related to offscreen graphics worlds that are described in this chapter. On a basic QuickDraw system, however, your application should not supply this handle to Color QuickDraw routines. ▲

The `MyPaintRectsThruGWorld` routine initializes the offscreen pixel image to all white by calling the `EraseRect` procedure, which is described in the chapter "Basic QuickDraw." The `MyPaintRectsThruGWorld` routine then calls another application-defined routine, `MyPaintAndFillColorRects`, to draw color rectangles into the pixel map for the offscreen graphics world.

IMPORTANT

You cannot dereference the `GWorldPtr` data structure to get to the pixel map. The `baseAddr` field of the `PixMap` record for an offscreen graphics world contains a handle instead of a pointer, which is what the `baseAddr` field for an onscreen pixel map contains. You must use the `GetPixBaseAddr` function (described on page 6-38) to obtain a pointer to the `PixMap` record for an offscreen graphics world. ▲

Copying an Offscreen Image Into a Window

After preparing an image in the offscreen graphics world, your application must use `SetGWorld` to restore the active window as the current graphics port, as illustrated in Listing 6-1 on page 6-5.

To copy the image from an offscreen graphics world into a window, use the `CopyBits` procedure. Specify the offscreen graphics world as the source image for `CopyBits`, and specify the window as its destination. When using `CopyBits`, you must coerce the offscreen graphics world's `GWorldPtr` data type to a data structure of type `GrafPtr`. Similarly, whenever a color graphics port is your destination, you must coerce the window's `CGrafPtr` data type to a data structure of type `GrafPtr`. (The `CopyBits` procedure is described in the chapter "QuickDraw Drawing.")

As long as you're drawing into an offscreen graphics world or copying the image out of it, you must leave its pixel image locked. When you are finished drawing into and copying from an offscreen graphics world, use the `UnlockPixels` procedure. To help prevent heap fragmentation, the `UnlockPixels` procedure allows the Memory Manager to move the base address for the offscreen pixel image. (For more information about Macintosh memory management, see *Inside Macintosh: Memory*.)

Finally, call the `DisposeGWorld` procedure when your application no longer needs the pixel image associated with this offscreen graphics world, as illustrated in Listing 6-1.

Updating an Offscreen Graphics World

When the user resizes or moves a window, changes the pixel depth of a screen that a window intersects, or modifies a color table, you can use the `UpdateGWorld` function to reflect the user's choices in the offscreen graphics world. The `UpdateGWorld` function, described on page 6-23, allows you to change the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world without recreating it and redrawing its contents. You should also call `UpdateGWorld` after every update event.

Calling `UpdateGWorld` and then `CopyBits` when the user makes these changes helps your application get the maximum refresh speed when updating the window. See the chapters "Event Manager" and "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information about handling update events in windows and about resizing windows.

Creating a Mask and a Source Image in Offscreen Graphics Worlds

When you use the `CopyMask` or `CopyDeepMask` procedure (described in the chapter "QuickDraw Drawing"), you can create the source image and its mask in separate offscreen graphics worlds. Plates 3 and 4 at the front of this book illustrate how to use `CopyMask` in this way. The source image in Plate 3 consists of graduated gray stripes; this image is created in an offscreen graphics world. The mask in Plate 3 consists of a black rectangle alongside a red rectangle; this mask is created in a separate graphics world that shares the same coordinates as the source image.

When the `CopyMask` procedure copies the grayscale image through this mask, the result is the untitled window illustrated in the bottom half of Plate 3. The black pixels in the mask cause `CopyMask` to copy directly into the window those pixels from the source image that are masked by the black rectangle. The red pixels in the mask cause `CopyMask` to alter most of the source pixels masked by the red rectangle when copying them. That is, the source pixels that are completely black are changed to the mask's red when copied into the window. The source pixels that are completely white are left unaltered when copied into the window. The source pixels that are between black and white are given a graduated amount of the mask's red.

Listing 6-2 shows the code that produces the window shown in Plate 3.

Listing 6-2 Using two offscreen graphics worlds and the `CopyMask` procedure

```
PROCEDURE MyCopyBlackAndRedMasks (wp: WindowPtr);
   VAR
      origPort:                          GrafPtr;
      origDevice:                        GDHandle;
      myErr:                             QDErr;
      myOffScreen1, myOffScreen2:        GWorldPtr;
      theColor:                          RGBColor;
      i:                                 Integer;
      offPixMapHandle1, offPixMapHandle2: PixMapHandle;
      good:                              Boolean;
      myRect:                            Rect;
BEGIN
   GetGWorld(origPort, origDevice);       {save window's graphics port}
                        {create an offscreen world for building an image}
   myErr := NewGWorld(myOffScreen1, 0, wp^.portRect, NIL, NIL, []);
   IF (myOffScreen1 = NIL) OR (myErr <> noErr) THEN
      ; {handle error here}
                        {create another offscreen world for building a mask}
   myErr := NewGWorld(myOffScreen2, 0, wp^.portRect, NIL, NIL, []);
   IF (myOffScreen2 = NIL) OR (myErr <> noErr) THEN
      ; {handle error here}
   SetGWorld(myOffScreen1, NIL); {make first offscreen world the }
                                 { current port}
   offPixMapHandle1 := GetGWorldPixMap(myOffScreen1);
   good := LockPixels(offPixMapHandle1);  {lock its pixel image}
   IF NOT good THEN
      ; {handle error here}
   EraseRect(myOffScreen1^.portRect);     {initialize its pixel image}
   FOR i := 0 TO 9 DO      {draw graduated grayscale stripes for the image}
      BEGIN
         theColor.red := i * 7168;
         theColor.green := i * 7168;
         theColor.blue := i * 7168;
         RGBForeColor(theColor);
         SetRect(myRect, myOffScreen1^.portRect.left, i * 10,
                 myOffScreen1^.portRect.right, i * 10 + 10);
         PaintRect(myRect);
      END;
   SetGWorld(myOffScreen2, NIL); {make second offscreen world the }
                                 { current port}
   offPixMapHandle2 := GetGWorldPixMap(myOffScreen2);
   good := LockPixels(offPixMapHandle2);  {lock its pixel image}
   IF NOT good THEN
      ; {handle error here}
   EraseRect(myOffScreen2^.portRect);     {initialize its pixel image}
   SetRect(myRect, 20, 20, 80, 80);
   PaintRect(myRect);   {paint a black rectangle in the mask}
   SetRect(myRect, 100, 20, 160, 80);
   theColor.red := $FFFF;  theColor.green := $0000; theColor.blue := $0000;
   RGBForeColor(theColor);
   PaintRect(myRect);   {paint a red rectangle in the mask}
   SetGWorld(wp, GetMainDevice); {make window the current port}
   EraseRect(wp^.portRect);      {erase the window before using CopyMask}
   CopyMask(GrafPtr(myOffScreen1)^.portBits, {use gray image as source}
            GrafPtr(myOffScreen2)^.portBits, {use 2-rectangle image as mask}
            GrafPtr(wp)^.portBits,           {use window as destination}
            myOffScreen1^.portRect,
            myOffScreen2^.portRect,
            wp^.portRect);
   UnlockPixels(offPixMapHandle1);  UnlockPixels(offPixMapHandle2);
   DisposeGWorld(myOffScreen1);  DisposeGWorld(myOffScreen2);
   SetGWorld(origPort, origDevice);    {restore original graphics port}
END;
```

Offscreen Graphics Worlds Reference

This section describes the data structures and routines that your application can use to create and manage offscreen graphics worlds. “Data Structures” shows the Pascal data structures for the `GWorldPtr` and `GWorldFlags` data types. “Routines” describes routines for creating, altering, and disposing of offscreen graphics worlds; for saving and restoring offscreen graphics worlds; and for managing an offscreen graphics world’s pixel map.

Data Structures

This section shows the Pascal data structures for the `GWorldPtr` and `GWorldFlags` data types. Your application uses pointers of type `GWorldPtr` to refer to the offscreen graphics worlds it creates. Several routines in this chapter expect or return values defined by the `GWorldFlags` data type.

GWorldPtr

An offscreen graphics world in Color QuickDraw contains a `CGrafPort` record—and its handles to associated `PixMap` and `ColorTable` records—that describes an offscreen graphics port and contains references to a `GDevice` record and other state information. The actual data structure for an offscreen graphics world is kept private to allow for future extensions. However, when your application uses the `NewGWorld` function to create an offscreen world, `NewGWorld` returns a pointer of type `GWorldPtr` by which your application refers to the offscreen graphics world. This pointer is defined as follows:

```
TYPE GWorldPtr = CGrafPtr;
```

On computers lacking Color QuickDraw, `GWorldPtr` points to an extension of the `GrafPort` record.

GWorldFlags

Several routines in this chapter expect or return values defined by the `GWorldFlags` data type, which is defined as follows:

```
TYPE GWorldFlags =
SET OF (
   pixPurge,          {specify to NewGWorld to make base address }
                      { for offscreen pixel image purgeable}
   noNewDevice,       {specify to NewGWorld to not create a new }
                      { GDevice record for offscreen world}
   useTempMem,        {specify to NewGWorld to create base }
                      { address for offscreen pixel image in }
                      { temporary memory}
   keepLocal,         {specify to NewGWorld to keep offscreen }
                      { pixel image in main memory}
   gWorldFlag4,       {reserved}
   gWorldFlag5,       {reserved}
   pixelsPurgeable,   {returned by GetPixelsState to indicate }
                      { that base address for offscreen pixel }
                      { image is purgeable; specify to }
                      { SetPixelsState to make base address for }
                      { pixel image purgeable}
   pixelsLocked,      {returned by GetPixelsState to indicate }
                      { that base address for offscreen pixel }
                      { image is locked; specify to }
                      { SetPixelsState to lock base address for }
                      { offscreen pixel image}
   gWorldFlag8,       {reserved}
   gWorldFlag9,       {reserved}
   gWorldFlag10,      {reserved}
   gWorldFlag11,      {reserved}
   gWorldFlag12,      {reserved}
   gWorldFlag13,      {reserved}
   gWorldFlag14,      {reserved}
   gWorldFlag15,      {reserved}
   mapPix,            {returned by UpdateGWorld if it remapped }
                      { colors to a new color table}
   newDepth,          {returned by UpdateGWorld if it translated }
                      { pixel map to a different pixel depth}
   alignPix,          {returned by UpdateGWorld if it realigned }
                      { pixel image to onscreen window}
   newRowBytes,       {returned by UpdateGWorld if it changed }
                      { rowBytes field of PixMap record}
   reallocPix,        {returned by UpdateGWorld if it reallocated }
                      { base address for offscreen pixel image}
   gWorldFlag21,      {reserved}
   gWorldFlag22,      {reserved}
   gWorldFlag23,      {reserved}
   gWorldFlag24,      {reserved}
   gWorldFlag25,      {reserved}
   gWorldFlag26,      {reserved}
   gWorldFlag27,      {reserved}
   clipPix,           {specify to UpdateGWorld to update and clip }
                      { pixel image}
   stretchPix,        {specify to UpdateGWorld to update and }
                      { stretch or shrink pixel image}
   ditherPix,         {specify to UpdateGWorld to dither pixel }
                      { image}
   gwFlagErr,         {returned by UpdateGWorld if it failed}
);
```

Field descriptions

`pixPurge`	If you specify this flag for the `flags` parameter of the `NewGWorld` function, `NewGWorld` (described on page 6-16) makes the base address for the offscreen pixel image purgeable.
`noNewDevice`	If you specify this flag for the `flags` parameter of the `NewGWorld` function, `NewGWorld` does not create a new offscreen `GDevice` record; instead, `NewGWorld` uses either the `GDevice` record you specify or the `GDevice` record for a video card on the user's system.
`useTempMem`	If you specify this in the `flags` parameter of the `NewGWorld` function, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally should not use this flag. You should use temporary memory only for fleeting purposes and only with the `AllowPurgePixels` procedure (described on page 6-34) so that other applications can launch.
`keepLocal`	If you specify this in the `flags` parameter of the `NewGWorld` function, `NewGWorld` creates a pixel image in Macintosh main memory where it cannot be cached to a graphics accelerator card.
`gWorldFlag4`	Reserved.
`gWorldFlag5`	Reserved.
`pixelsPurgeable`	If you specify this in the `state` parameter of the `SetPixelsState` procedure (described on page 6-37), `SetPixelsState` makes the base address for an offscreen pixel map purgeable. If you use the `SetPixelsState` procedure without passing it this flag, then `SetPixelsState` makes the base address for an offscreen pixel map unpurgeable. If the `GetPixelsState` function (described on page 6-36) returns this flag, then the base address for an offscreen pixel is purgeable.

`pixelsLocked`	If you specify this flag for the `state` parameter of the `SetPixelsState` procedure, `SetPixelsState` locks the base address for an offscreen pixel image. If you use the `SetPixelsState` procedure without passing it this flag, then `SetPixelsState` unlocks the offscreen pixel image. If the `GetPixelsState` function returns this flag, then the base address for an offscreen pixel is locked.
`gWorldFlag8`	Reserved.
`gWorldFlag9`	Reserved.
`gWorldFlag10`	Reserved.
`gWorldFlag11`	Reserved.
`gWorldFlag12`	Reserved.
`gWorldFlag13`	Reserved.
`gWorldFlag14`	Reserved.
`gWorldFlag15`	Reserved.
`mapPix`	If the `UpdateGWorld` function (described on page 6-23) returns this flag, then it remapped the colors in the offscreen pixel map to a new color table.
`newDepth`	If the `UpdateGWorld` function returns this flag, then it translated the offscreen pixel map to a different pixel depth.
`alignPix`	If the `UpdateGWorld` function returns this flag, then it realigned the offscreen pixel image to an onscreen window.
`newRowBytes`	If the `UpdateGWorld` function returns this flag, then it changed the `rowBytes` field of the `PixMap` record for the offscreen graphics world.
`reallocPix`	If the `UpdateGWorld` function returns this flag, then it reallocated the base address for the offscreen pixel image. Your application should then reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.
`gWorldFlag21`	Reserved.
`gWorldFlag22`	Reserved.
`gWorldFlag23`	Reserved.
`gWorldFlag24`	Reserved.
`gWorldFlag25`	Reserved.
`gWorldFlag26`	Reserved.
`gWorldFlag27`	Reserved.
`clipPix`	If the `UpdateGWorld` function returns this flag, then it clipped the pixel image.
`stretchPix`	If the `UpdateGWorld` function returns this flag, then it stretched or shrank the offscreen image.
`ditherPix`	If the `UpdateGWorld` function returns this flag, then it dithered the offscreen image.
`gwFlagErr`	If the `UpdateGWorld` function returns this flag, then it was unsuccessful and the offscreen graphics world was left unchanged.

Routines

This section describes routines for creating, altering, and disposing of offscreen graphics worlds; for saving and restoring offscreen graphics worlds; and for managing an offscreen graphics world's pixel map.

Creating, Altering, and Disposing of Offscreen Graphics Worlds

To create an offscreen graphics world, use the `NewGWorld` function. The `NewGWorld` function uses the `NewScreenBuffer` function to create and allocate memory for an offscreen pixel image; your application generally won't need to use `NewScreenBuffer`, but it is described here for completeness. The `NewGWorld` function similarly uses the `NewTempScreenBuffer` function to create and allocate temporary memory for an offscreen pixel image.

To change the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world, use the `UpdateGWorld` function.

When you no longer need the pixel image associated with this offscreen graphics world, use the `DisposeGWorld` procedure to dispose of all the memory allocated for the offscreen graphics world. The `DisposeGWorld` procedure uses the `DisposeScreenBuffer` procedure when disposing of an offscreen graphics world; generally, your application won't need to use `DisposeScreenBuffer`.

Note
Before drawing into an offscreen graphics world, be sure to use the `SetGWorld` procedure (described on page 6-29) to make that offscreen world the current graphics port. In addition, before drawing into—or copying from—an offscreen pixel map, be sure to use the `LockPixels` function, which is described on page 6-32. ◆

NewGWorld

Use the `NewGWorld` function to create an offscreen graphics world.

```
FUNCTION NewGWorld (VAR offscreenGWorld: GWorldPtr;
                    pixelDepth: Integer; boundsRect: Rect;
                    cTable: CTabHandle; aGDevice: GDHandle;
                    flags: GWorldFlags): QDErr;
```

`offscreenGWorld`
: A pointer to the offscreen graphics world created by this routine.

`pixelDepth`
: The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. If you specify 0 in this parameter, you get the default behavior for the `NewGWorld` function—that is, it uses the pixel depth of the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `NewGWorld` also uses the `GDevice` record from this device instead of creating a new `GDevice` record for the offscreen world. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only 0 or 1 in this parameter.

`boundsRect`
: The boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` record, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. (`NewGWorld` then uses the pixel depth, color table, and `GDevice` record from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle.) Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

`cTable`
: A handle to a `ColorTable` record. If you pass `NIL` in this parameter, `NewGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter. If you set the `pixelDepth` parameter to 0, `NewGWorld` ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use `NewGWorld` on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

`aGDevice`
: A handle to a `GDevice` record that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case `NewGWorld` attaches this `GDevice` record to the new offscreen graphics world. If you set the `pixelDepth` parameter to 0, or if you do not set the `noNewDevice` flag, `NewGWorld` ignores the `aGDevice` parameter, so you should set it to `NIL`. If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` record for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` records for offscreen graphics worlds.

`flags` Options available to your application. You can set a combination of the flags `pixPurge`, `noNewDevice`, `useTempMem`, and `keepLocal`. If you don't wish to use any of these flags, pass the empty set ([ ]) in your Pascal code or 0 in your C code in this parameter, in which case you get the default behavior for `NewGWorld`—that is, it creates an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, it uses an existing `GDevice` record (if you pass 0 in the `depth` parameter) or creates a new `GDevice` record, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. The available flags are described here:

```
TYPE GWorldFlags =
SET OF (       {flags for only NewGWorld are listed here}
 pixPurge,     {make base address for offscreen pixel }
               { image purgeable}
 noNewDevice,  {do not create an offscreen GDevice }
               { record}
 useTempMem,   {create base address for offscreen pixel }
               { image in temporary memory}
 keepLocal,    {keep offscreen pixel image in main }
               { memory where it cannot be cached to }
               { a graphics accelerator card}
);
```

DESCRIPTION

The `NewGWorld` function creates an offscreen graphics world with the pixel depth you specify in the `pixelDepth` parameter, the boundary rectangle you specify in the `boundsRect` parameter, the color table you specify in the `cTable` parameter, and the options you specify in the `flags` parameter. The `NewGWorld` function returns a pointer to the new offscreen graphics world in the `offscreenGWorld` parameter. You use this pointer when referring to this new offscreen world in other routines described in this chapter.

Typically, you pass 0 in the `pixelDepth` parameter, a window's port rectangle in the `boundsRect` parameter, `NIL` in the `cTable` and `aGDevice` parameters, and—in the `flags` parameter—an empty set ([ ]) for Pascal code or 0 for C code. This provides your application with the default behavior of `NewGWorld`, and it supports computers running basic QuickDraw. This also allows QuickDraw to optimize the `CopyBits`, `CopyMask`, and `CopyDeepMask` procedures when your application copies the image in an offscreen graphics world into an onscreen graphics port.

The `NewGWorld` function allocates memory for an offscreen graphics port and its pixel map. On computers that support only basic QuickDraw, `NewGWorld` creates a 1-bit pixel map that your application can manipulate using other relevant routines described in this chapter. Your application can copy this 1-bit pixel map into basic graphics ports.

Unless you specify 0 in the `pixelDepth` parameter—or pass the `noNewDevice` flag in the `flags` parameter and supply a `GDevice` record in the `aGDevice` parameter—`NewGWorld` also allocates a new offscreen `GDevice` record.

When creating an image, your application can use the `NewGWorld` function to create an offscreen graphics world that is optimized for an image's characteristics—for example, its best pixel depth. After creating the image, your application can then use the `CopyBits`, `CopyMask`, or `CopyDeepMask` procedure to copy that image to an onscreen graphics port. Color QuickDraw automatically renders the image at the best available pixel depth for the screen. Creating an image in an offscreen graphics port and then copying it to the screen in this way prevents the visual choppiness that would otherwise occur if your application were to build a complex image directly onscreen.

The `NewGWorld` function initializes the offscreen graphics port by calling the `OpenCPort` function. The `NewGWorld` function sets the offscreen graphics port's visible region to a rectangular region coincident with its boundary rectangle. The `NewGWorld` function generates an inverse table with the Color Manager procedure `MakeITable`, unless one of the `GDevice` records for the screens has the same color table as the `GDevice` record for the offscreen world, in which case `NewGWorld` uses the inverse table from that `GDevice` record.

The address of the offscreen pixel image is not directly accessible from the `PixMap` record for the offscreen graphics world. However, you can use the `GetPixBaseAddr` function (described on page 6-38) to get a pointer to the beginning of the offscreen pixel image.

For purposes of estimating memory use, you can compute the size of the offscreen pixel image by using this formula:

```
rowBytes * (boundsRect.bottom – boundsRect.top)
```

In the `flags` parameter, you can specify several options that are defined by the `GWorldFlags` data type. If you don't wish to use any of these options, pass an empty set ([ ]) in the `flags` parameter for Pascal code or pass 0 here for C code.

- If you specify the `pixPurge` flag, `NewGWorld` stores the offscreen pixel image in a purgeable block of memory. In this case, before drawing to or from the offscreen pixel image, your application should call the `LockPixels` function (described on page 6-32) and ensure that it returns `TRUE`. If `LockPixels` returns `FALSE`, the memory for the pixel image has been purged, and your application should either call `UpdateGWorld` to reallocate it and then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world. Never draw to or copy from an offscreen pixel image that has been purged without reallocating its memory and then reconstructing it.
- If you specify the `noNewDevice` flag, `NewGWorld` does not create a new offscreen `GDevice` record. Instead, it uses the `GDevice` record that you specify in the `aGDevice` parameter—and its associated pixel depth and color table—to create the offscreen graphics world. (If you set the `pixelDepth` parameter to 0, `NewGWorld` uses the `GDevice` record for the screen with the greatest pixel depth among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter—even if you specify the `noNewDevice` flag.) The

`NewGWorld` function keeps a reference to the `GDevice` record for the offscreen graphics world, and the `SetGWorld` procedure (described on page 6-29) uses that record to set the current graphics device.

- If you set the `useTempMem` flag, `NewGWorld` creates the base address for an offscreen pixel image in temporary memory. You generally would not use this flag, because you should use temporary memory only for fleeting purposes and only with the `AllowPurgePixels` procedure (described on page 6-34).
- If you specify the `keepLocal` flag, your offscreen pixel image is kept in Macintosh main memory and is not cached to a graphics accelerator card. Use this flag carefully, as it negates the advantages provided by any graphics acceleration card that might be present.

As its function result, `NewGWorld` returns one of three result codes.

SPECIAL CONSIDERATIONS

If you supply a handle to a `ColorTable` record in the `cTable` parameter, `NewGWorld` makes a copy of the record and stores its handle in the offscreen `PixMap` record. It is your application's responsibility to make sure that the `ColorTable` record you specify in the `cTable` parameter is valid for the offscreen graphics port's pixel depth.

If when using `NewGWorld` you specify a pixel depth, color table, or `GDevice` record that differs from those used by the window into which you copy your offscreen image, the `CopyBits`, `CopyMask`, and `CopyDeepMask` procedures require extra time to complete.

To use a custom color table in an offscreen graphics world, you need to create the associated offscreen `GDevice` record, because Color QuickDraw needs its inverse table.

The `NewGWorld` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `NewGWorld` function are

Trap macro	Selector
`_QDExtensions`	$00160000

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter
`cDepthErr`	–157	Invalid pixel depth

SEE ALSO

Listing 6-1 on page 6-5 and Listing 6-2 on page 6-10 illustrate how to use `NewGWorld` to create offscreen graphics worlds.

If your application needs to change the pixel depth, boundary rectangle, or color table for an offscreen graphics world, use the `UpdateGWorld` function, described on page 6-23.

NewScreenBuffer

The `NewGWorld` function uses the `NewScreenBuffer` function to create an offscreen `PixMap` record and allocate memory for the base address of its pixel image; applications generally don't need to use `NewScreenBuffer`.

```
FUNCTION NewScreenBuffer (globalRect: Rect;
                          purgeable: Boolean; VAR gdh: GDHandle;
                          VAR offscreenPixMap: PixMapHandle):
                          QDErr;
```

`globalRect`	The boundary rectangle, in global coordinates, for the offscreen pixel map.
`purgeable`	A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.
`gdh`	The handle to the `GDevice` record for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.
`offscreenPixMap`	A handle to the new offscreen `PixMap` record.

DESCRIPTION

The `NewScreenBuffer` function creates a new offscreen `PixMap` record, using the pixel depth and color table of the device whose `GDevice` record is returned in the `gdh` parameter. The `NewScreenBuffer` function returns a handle to the new offscreen pixel map in the `offscreenPixMap` parameter.

As its function result, `NewScreenBuffer` returns one of three result codes.

SPECIAL CONSIDERATIONS

The `NewScreenBuffer` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `NewScreenBuffer` function are

Trap macro	**Selector**
`_QDExtensions`	$000E0010

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter
`cNoMemErr`	–152	Failed to allocate memory for structures

NewTempScreenBuffer

The `NewGWorld` function uses the `NewTempScreenBuffer` function to create an offscreen `PixMap` record and allocate temporary memory for the base address of its pixel image; applications generally don't need to use `NewTempScreenBuffer`.

```
FUNCTION NewTempScreenBuffer (globalRect: Rect;
                              purgeable: Boolean;
                              VAR gdh: GDHandle;
                              VAR offscreenPixMap: PixMapHandle):
                              QDErr;
```

`globalRect`
: The boundary rectangle, in global coordinates, for the offscreen pixel map.

`purgeable`
: A value of `TRUE` to make the memory block for the offscreen pixel map purgeable, or a value of `FALSE` to make it unpurgeable.

`gdh`
: The handle to the `GDevice` record for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle specified in the `globalRect` parameter.

`offscreenPixMap`
: A handle to the new offscreen `PixMap` record.

DESCRIPTION

The `NewTempScreenBuffer` function performs the same functions as `NewScreenBuffer` except that it creates the base address for the offscreen pixel image in temporary memory. When an application passes it the `useTempMem` flag, the `NewGWorld` function uses `NewTempScreenBuffer` instead of `NewScreenBuffer`.

SPECIAL CONSIDERATIONS

The `NewTempScreenBuffer` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `NewTempScreenBuffer` function are

Trap macro	Selector
`_QDExtensions`	$000E0015

UpdateGWorld

To change the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world, use the `UpdateGWorld` function. You should call `UpdateGWorld` after every update event and whenever your windows move or change size.

```
FUNCTION UpdateGWorld (VAR offscreenGWorld: GWorldPtr;
                       pixelDepth: Integer; boundsRect: Rect;
                       cTable: CTabHandle; aGDevice: GDHandle;
                       flags: GWorldFlags): GWorldFlags;
```

`offscreenGWorld`
: On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

`pixelDepth`
: The pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. If you specify 0 in this parameter, `UpdateGWorld` rescans the device list and uses the depth of the screen with the greatest pixel depth among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify 0 in this parameter, `UpdateGWorld` also copies the `GDevice` record from this device to create an offscreen `GDevice` record. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` record in the `aGDevice` parameter.

`boundsRect`
: The boundary rectangle and port rectangle for the offscreen pixel map. This also becomes the boundary rectangle for the `GDevice` record, if `NewGWorld` creates one. If you specify 0 in the `pixelDepth` parameter, `NewGWorld` interprets the boundaries in global coordinates, with which it determines which screens intersect the rectangle. (`NewGWorld` then uses the pixel depth, color table, and `GDevice` record from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle.) Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

`cTable`	A handle to a `ColorTable` record. If you pass `NIL` in this parameter, `UpdateGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter; if you set the `pixelDepth` parameter to 0, `UpdateGWorld` copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. The `UpdateGWorld` function ignores the value you supply for this parameter if you specify a `GDevice` record in the `aGDevice` parameter.
`aGDevice`	As an option, a handle to a `GDevice` record whose pixel depth and color table you want to use for the offscreen graphics world. To use the pixel depth and color table that you specify in the `pixelDepth` and `cTable` parameters, set this parameter to `NIL`.
`flags`	Options available to your application. You can set a combination of the flags `keepLocal`, `clipPix`, `stretchPix`, and `ditherPix`. If you don't wish to use any of these flags, pass the empty set ([ ]) in this parameter for Pascal code or pass 0 here for C code. However, you should pass either `clipPix` or `stretchPix` to ensure that the pixel map is updated to reflect the new color table. The available flags are described here:

```
TYPE GWorldFlags =
SET OF (       {flags for UpdateGWorld are listed here}
 keepLocal,    {keep data structures in main memory}
 clipPix,      {update and clip pixel image to new }
               { boundary rectangle}
 stretchPix,   {update and stretch or shrink pixel }
               { image to the new boundary rectangle}
 ditherPix,    {include with clipPix or stretchPix }
               { flag to dither the pixel image}
);
```

DESCRIPTION

The `UpdateGWorld` function changes an offscreen graphics world to the specified pixel depth, rectangle, color table, and options that you supply in the `pixelDepth`, `boundsRect`, `cTable`, and `flags` parameters, respectively. In the `offscreenGWorld` parameter, pass the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world.

If the `LockPixels` function (described on page 6-32) reports that the Memory Manager has purged the base address for the offscreen pixel image, you can use `UpdateGWorld` to reallocate its memory. Your application should then reconstruct the pixel image or draw directly in a window instead of preparing the image in an offscreen graphics world.

In the `flags` parameter, you can specify the `keepLocal` flag, which keeps the offscreen pixel image in Macintosh main memory or returns the image to main memory if it had been previously cached. If you use `UpdateGWorld` without passing it the `keepLocal` flag, you allow the offscreen pixel image to be cached on a graphics accelerator card if one is present.

As its function result, `UpdateGWorld` returns the `gwFlagErr` flag if `UpdateGWorld` was unsuccessful; in this case, the offscreen graphics world is left unchanged. You can use the `QDError` function, described in the chapter "Color QuickDraw," to help you determine why `UpdateGWorld` failed.

If `UpdateGWorld` is successful, it returns a combination of the following flags, which are defined by the `GWorldFlags` data type:

Flag	Meaning
`mapPix`	Color QuickDraw remapped the colors to a new color table.
`newDepth`	Color QuickDraw translated the pixel values in the offscreen pixel image to those for a different pixel depth.
`alignPix`	QuickDraw realigned the offscreen image to the window.
`newRowBytes`	QuickDraw changed the value of the `rowBytes` field in the `PixMap` record for the offscreen graphics world.
`reallocPix`	QuickDraw had to reallocate memory for the offscreen pixel image; your application should then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world.
`clipPix`	QuickDraw clipped the pixel image.
`stretchPix`	QuickDraw stretched or shrank the offscreen image.
`ditherPix`	Color QuickDraw dithered the offscreen pixel image.

The `UpdateGWorld` function uses the following algorithm when updating the offscreen pixel image:

1. If the color table that you specify in the `cTable` parameter is different from the previous color table, or if the color table associated with the `GDevice` record that you specify in the `aGDevice` parameter is different, Color QuickDraw maps the pixel values in the offscreen pixel map to the new color table.
2. If the value you specify in the `pixelDepth` parameter differs from the previous pixel depth, Color QuickDraw translates the pixel values in the offscreen pixel image to those for the new pixel depth.
3. If the rectangle you specify in the `boundsRect` parameter differs from, but has the same size as, the previous boundary rectangle, QuickDraw realigns the pixel image to the screen for optimum performance for the `CopyBits` procedure.
4. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `clipPix` flag, the pixel image is clipped along the bottom and right edges.

5. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `clipPix` flag, the bottom and right edges of the pixel image are undefined.
6. If the rectangle you specify in the `boundsRect` parameter is smaller than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is reduced to the new size.
7. If the rectangle you specify in the `boundsRect` parameter is bigger than the previous boundary rectangle and you specify the `stretchPix` flag, the pixel image is stretched to the new size.
8. If the Memory Manager purged the base address for the offscreen pixel image, `UpdateGWorld` reallocates the memory, but the pixel image is lost. You must reconstruct it.

SPECIAL CONSIDERATIONS

The `UpdateGWorld` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `UpdateGWorld` function are

Trap macro	Selector
`_QDExtensions`	$00160003

DisposeGWorld

Use the `DisposeGWorld` procedure to dispose of all the memory allocated for an offscreen graphics world.

```
PROCEDURE DisposeGWorld (offscreenGWorld: GWorldPtr);
```

`offscreenGWorld`
A pointer to an offscreen graphics world.

DESCRIPTION

The `DisposeGWorld` procedure disposes of all the memory allocated for the offscreen graphics world pointed to in the `offscreenGWorld` parameter, including its pixel map, color table, pixel image, and `GDevice` record (if one was created). In the `offscreenGWorld` parameter, pass the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world.

Call `DisposeGWorld` only when your application no longer needs the pixel image associated with this offscreen graphics world. If this offscreen graphics world was the current device, the current device is reset to the device stored in the global variable `MainDevice`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `DisposeGWorld` procedure are

Trap macro	Selector
`_QDExtensions`	$00040004

DisposeScreenBuffer

The `DisposeGWorld` procedure uses the `DisposeScreenBuffer` procedure when disposing of an offscreen graphics world; generally, applications do not need to use `DisposeScreenBuffer`.

```
PROCEDURE DisposeScreenBuffer (offscreenPixMap: PixMapHandle);
```

`offscreenPixMap`
A handle to an existing offscreen `PixMap` record.

DESCRIPTION

The `DisposeScreenBuffer` procedure disposes of the memory allocated for the base address of an offscreen pixel image.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `DisposeScreenBuffer` procedure are

Trap macro	Selector
`_QDExtensions`	$00040011

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

To save the current graphics port (basic, color, or offscreen) and the current `GDevice` record, use the `GetGWorld` procedure. To change the current graphics port (basic, color, or offscreen), use the `SetGWorld` procedure; any drawing your application performs then occurs in this graphics port.

You can use the `GetGWorldDevice` function to obtain a handle to the `GDevice` record associated with an offscreen graphics world.

GetGWorld

To save the current graphics port (basic, color, or offscreen) and the current `GDevice` record, use the `GetGWorld` procedure.

```
PROCEDURE GetGWorld (VAR port: CGrafPtr; VAR gdh: GDHandle);
```

`port`	A pointer to an offscreen graphics world, color graphics port, or basic graphics port, depending on which is the current port.
`gdh`	A handle to the `GDevice` record for the current device.

DESCRIPTION

The `GetGWorld` procedure returns a pointer to the current graphics port in the `port` parameter. This parameter can return values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether the current graphics port is a basic graphics port, color graphics port, or offscreen graphics world. The `GetGWorld` procedure returns a handle to the `GDevice` record for the current device in the `gdh` parameter.

After using `GetGWorld` to save a graphics port and a `GDevice` record, your application can later use the `SetGWorld` procedure, described next, to restore them.

SPECIAL CONSIDERATIONS

The `GetGWorld` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetGWorld` procedure are

Trap macro	Selector
`_QDExtensions`	$00080005

SEE ALSO

Listing 6-1 on page 6-5 and Listing 6-2 on page 6-10 illustrate how to use the `GetGWorld` procedure to save the current graphics port for an active window, the `SetGWorld` procedure to change the current graphics port to an offscreen graphics world, and then `SetGWorld` again to restore the active window as the current graphics port.

SetGWorld

To change the current graphics port (basic, color, or offscreen), use the `SetGWorld` procedure.

```
PROCEDURE SetGWorld (port: CGrafPtr; gdh: GDHandle);
```

`port`	A pointer to an offscreen graphics world, color graphics port, or basic graphics port.
`gdh`	A handle to a `GDevice` record. If you pass a pointer to an offscreen graphics world in the `port` parameter, set this parameter to `NIL`, because `SetGWorld` ignores this parameter and sets the current device to the device attached to the offscreen graphics world.

DESCRIPTION

The `SetGWorld` procedure sets the current graphics port to the one specified by the `port` parameter and—unless you set the current graphics port to be an offscreen graphics world—sets the current device to that specified by the `gdh` parameter.

In the `port` parameter, you can specify values of type `GrafPtr`, `CGrafPtr`, or `GWorldPtr`, depending on whether you want to set the current graphics port to be a basic graphics port, color graphics port, or offscreen graphics world. Any drawing your application performs then occurs in this graphics port.

SPECIAL CONSIDERATIONS

The `SetGWorld` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SetGWorld` procedure are

Trap macro	Selector
`_QDExtensions`	$00080006

SEE ALSO

Listing 6-1 on page 6-5 and Listing 6-2 on page 6-10 illustrate how to use the `GetGWorld` procedure to save the current graphics port for an active window, the `SetGWorld` procedure to change the current graphics port to an offscreen graphics world, and then `SetGWorld` again to restore the active window as the current graphics port.

GetGWorldDevice

Use the `GetGWorldDevice` function to obtain a handle to the `GDevice` record associated with an offscreen graphics world.

```
FUNCTION GetGWorldDevice (offscreenGWorld: GWorldPtr): GDHandle;
```

`offscreenGWorld`
A pointer to an offscreen graphics world. The pointer returned to your application by the `NewGWorld` function.

DESCRIPTION

The `GetGWorldDevice` function returns a handle to the `GDevice` record associated with the offscreen graphics world specified by the `offscreenGWorld` parameter. In this parameter, supply the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world. If you created the offscreen world by specifying the `noNewDevice` flag, the `GDevice` record is for one of the screen devices or is the `GDevice` record that you specified to `NewGWorld` or `UpdateGWorld`.

If you point to a `GrafPort` or `CGrafPort` record in the `offscreenGWorld` parameter, `GetGWorldDevice` returns the current device.

SPECIAL CONSIDERATIONS

The `GetGWorldDevice` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetGWorldDevice` function are

Trap macro	Selector
`_QDExtensions`	$00040012

Managing an Offscreen Graphics World's Pixel Image

Use the `GetGWorldPixMap` function to obtain a handle to the `PixMap` record for an offscreen graphics world. You can then pass this handle, which is of data type `PixMapHandle`, in parameters to several routines that allow you to manage an offscreen graphics world's pixel image.

To prevent the base address for an offscreen pixel image from being moved (while you draw into or copy from its pixel map, for example), pass its handle to the `LockPixels` function. When you are finished drawing into or copying from an offscreen pixel map, pass its handle to the `UnlockPixels` procedure.

You can use the `AllowPurgePixels` procedure to make the base address for an offscreen pixel image purgeable. To prevent the Memory Manager from purging the base address for an offscreen pixel map, use the `NoPurgePixels` procedure.

To save the current information about the memory allocated for an offscreen pixel image, you can use the `GetPixelsState` function. To restore this state, you can use the `SetPixelsState` procedure.

You can use the `GetPixBaseAddr` function to obtain a pointer to the beginning of a pixel image in memory. You can use the `PixMap32Bit` function to determine whether a pixel map requires 32-bit addressing mode for access to its pixel image.

GetGWorldPixMap

Use the `GetGWorldPixMap` function to obtain the pixel map created for an offscreen graphics world.

```
FUNCTION GetGWorldPixMap (offscreenGWorld: GWorldPtr):
                        PixMapHandle;
```

`offscreenGWorld`
A pointer to an offscreen graphics world.

DESCRIPTION

The `GetGWorldPixMap` function returns a handle to the pixel map created for an offscreen graphics world. In the `offscreenGWorld` parameter, pass the pointer returned to your application by the `NewGWorld` function when you created the offscreen graphics world. Your application can, in turn, pass the handle returned by `GetGWorldPixMap` as a parameter to other QuickDraw routines that accept a handle to a pixel map.

On a system running only basic QuickDraw, the `GetGWorldPixMap` function returns the handle to a 1-bit pixel map that your application can supply as a parameter to the other routines related to offscreen graphics worlds. However, your application should not supply this handle to Color QuickDraw routines.

SPECIAL CONSIDERATIONS

To ensure compatibility on systems running basic QuickDraw instead of Color QuickDraw, use `GetGWorldPixMap` whenever you need to gain access to the bitmap created for a graphics world—that is, do *not* dereference the `GWorldPtr` record for that graphics world.

The `GetGWorldPixMap` function is not available in systems preceding System 7. You can make sure that the `GetGWorldPixMap` function is available by using the `Gestalt` function with the `gestaltSystemVersion` selector. Test the low-order word in the `response` parameter; if the value is $0700 or greater, then `GetGWorldPixMap` is available.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetGWorldPixMap` function are

Trap macro	Selector
`_QDExtensions`	$00040017

SEE ALSO

The `Gestalt` function is described in the chapter "Gestalt Manager" of *Inside Macintosh: Operating System Utilities*.

LockPixels

To prevent the base address for an offscreen pixel image from being moved while you draw into or copy from its pixel map, use the `LockPixels` function.

```
FUNCTION LockPixels (pm: PixMapHandle): Boolean;
```

`pm`	A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` function, described on page 6-31.

DESCRIPTION

The `LockPixels` function prevents the base address for an offscreen pixel image from being moved. You must call `LockPixels` before drawing to or copying from an offscreen graphics world.

The `baseAddr` field of the `PixMap` record for an offscreen graphics world contains a handle instead of a pointer (which is what the `baseAddr` field for an onscreen pixel map contains). The `LockPixels` function dereferences the `PixMap` handle into a pointer. When you use the `UnlockPixels` procedure, which is described next, the handle is recovered.

If the base address for an offscreen pixel image hasn't been purged by the Memory Manager or is not purgeable, `LockPixels` returns `TRUE` as its function result, and your application can draw into or copy from the offscreen pixel map. However, if the base address for an offscreen pixel image has been purged, `LockPixels` returns `FALSE` to indicate that you can perform no drawing to or copying from the pixel map. At that point, your application should either call the `UpdateGWorld` function (described on page 6-23) to reallocate the offscreen pixel image and then reconstruct it, or draw directly in a window instead of preparing the image in an offscreen graphics world.

As soon as you are finished drawing into and copying from the offscreen pixel image, you should call the `UnlockPixels` procedure.

SPECIAL CONSIDERATIONS

The `LockPixels` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `LockPixels` function are

Trap macro	Selector
`_QDExtensions`	$00040001

SEE ALSO

Listing 6-1 on page 6-5 and Listing 6-2 on page 6-10 illustrate the use of this function. See *Inside Macintosh: Memory* for more information about memory management.

UnlockPixels

When you are finished drawing into and copying from an offscreen graphics world, use the `UnlockPixels` procedure.

```
PROCEDURE UnlockPixels (pm: PixMapHandle);
```

`pm`	A handle to an offscreen pixel map. Pass the same handle that you passed previously to the `LockPixels` function.

DESCRIPTION

The `UnlockPixels` procedure allows the Memory Manager to move the base address for the offscreen pixel map that you specify in the `pm` parameter. To ensure the integrity of the data in a pixel image, call `LockPixels` (explained in the preceding section) before drawing into or copying from a pixel map; then, to prevent heap fragmentation, call `UnlockPixels` as soon as your application finishes drawing to and copying from the offscreen pixel map.

The `baseAddr` field of the `PixMap` record for an offscreen graphics world contains a handle instead of a pointer (which is what the `baseAddr` field for an onscreen pixel map contains). The `LockPixels` function dereferences the `PixMap` handle into a pointer. When you use the `UnlockPixels` procedure, the handle is recovered.

You don't need to call `UnlockPixels` if `LockPixels` returns `FALSE`, because `LockPixels` doesn't lock the memory for a pixel image if that memory has been purged. However, calling `UnlockPixels` on purged memory does no harm.

SPECIAL CONSIDERATIONS

The `UnlockPixels` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `UnlockPixels` procedure are

Trap macro	Selector
`_QDExtensions`	$00040002

AllowPurgePixels

You can use the `AllowPurgePixels` procedure to make the base address for an offscreen pixel image purgeable.

```
PROCEDURE AllowPurgePixels (pm: PixMapHandle);
```

`pm`	A handle to an offscreen pixel map.

DESCRIPTION

The `AllowPurgePixels` procedure marks the base address for an offscreen pixel image as purgeable; this allows the Memory Manager to free the memory it occupies if available memory space becomes low. By default, `NewGWorld` creates an unpurgeable base address for an offscreen pixel image.

To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` function, described on page 6-31. Then supply this handle for the `pm` parameter of `AllowPurgePixels`.

Your application should call the `LockPixels` function (described on page 6-32) before drawing into or copying from an offscreen pixel map. If the Memory Manager has purged the base address for its pixel image, `LockPixels` returns `FALSE`. In that case either your application should use the `UpdateGWorld` function (described on page 6-23) to begin reconstructing the offscreen pixel image, or it should draw directly to an onscreen graphics port.

Only unlocked memory blocks can be made purgeable. If you use `LockPixels`, you must use the `UnlockPixels` procedure (explained in the preceding section) before calling `AllowPurgePixels`.

SPECIAL CONSIDERATIONS

The `AllowPurgePixels` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `AllowPurgePixels` procedure are

Trap macro	Selector
`_QDExtensions`	$0004000B

SEE ALSO

See *Inside Macintosh: Memory* for more information about memory management.

NoPurgePixels

To prevent the Memory Manager from purging the base address for an offscreen pixel image, use the `NoPurgePixels` procedure.

```
PROCEDURE NoPurgePixels (pm: PixMapHandle);
```

`pm`	A handle to an offscreen pixel map.

DESCRIPTION

The `NoPurgePixels` procedure marks the base address for an offscreen pixel image as unpurgeable. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` function, described on page 6-31. Then supply this handle for the `pm` parameter of `NoPurgePixels`.

SPECIAL CONSIDERATIONS

The `NoPurgePixels` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `NoPurgePixels` procedure are

Trap macro	Selector
`_QDExtensions`	$0004000C

GetPixelsState

To save the current information about the memory allocated for an offscreen pixel image, you can use the `GetPixelsState` function.

```
FUNCTION GetPixelsState (pm: PixMapHandle): GWorldFlags;
```

`pm`	A handle to an offscreen pixel map.

DESCRIPTION

The `GetPixelsState` function returns information about the memory allocated for the base address for an offscreen pixel image. This information can be either of the following flags defined by the `GWorldFlags` data type:

```
TYPE GWorldFlags =
SET OF (       {flags for GetPixelsState only are listed here}
   pixelsPurgeable,  {the base address for an offscreen pixel }
                     { image is purgeable}
   pixelsLocked,     {the offscreen pixel image is locked and }
                     { not purgeable}
);
```

If the `pixelsPurgeable` flag is not returned, then the base address for the offscreen pixel image is unpurgeable. If the `pixelsLocked` flag is not returned, then the base address for the offscreen pixel image is unlocked.

After using `GetPixelsState` to save this state information, your application can later use the `SetPixelsState` procedure, described next, to restore this state to the offscreen graphics world.

Specify a handle to a pixel map in the `pm` parameter. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` function, described on page 6-31.

SPECIAL CONSIDERATIONS

The `GetPixelsState` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetPixelsState` function are

Trap macro	Selector
`_QDExtensions`	$0004000D

SEE ALSO

After using `GetPixelsState` and before using `SetPixelsState`, your application can temporarily use the `AllowPurgePixels` procedure (described on page 6-34) to make the base address for an offscreen pixel image purgeable, the `NoPurgePixels` procedure (described on page 6-35) to make it unpurgeable, the `LockPixels` function (described on page 6-32) to prevent it from being moved, and the `UnlockPixels` procedure (described on page 6-33) to allow it to be moved.

SetPixelsState

To restore an offscreen pixel image to the state that you saved with the `GetPixelsState` function (explained in the preceding section), you can use the `SetPixelsState` procedure.

```
PROCEDURE SetPixelsState (pm: PixMapHandle; state: GWorldFlags);
```

`pm` A handle to an offscreen pixel map.

`state` Flags, which you usually save with the `GetPixelsState` function, defined by the `GWorldFlags` data type:

```
TYPE GWorldFlags =
SET OF ( {flags for SetPixelsState are listed here}
  pixelsPurgeable,  {make the base address for an }
                    { offscreen pixel image purgeable}
  pixelsLocked      {prevent the base address for an }
                    { offscreen pixel image from }
                    { being moved}
);
```

DESCRIPTION

The `SetPixelsState` procedure changes the state of the memory allocated for an offscreen pixel image to the state indicated by the flags specified in the `state` parameter, which you typically save using the `GetPixelsState` function.

Because only an unlocked memory block can be purged, `SetPixelsState` calls the `UnlockPixels` and `AllowPurgePixels` procedures (described on page 6-33 and page 6-34, respectively) if the `state` parameter specifies the `pixelsPurgeable` flag. If the `state` parameter does not specify the `pixelsPurgeable` flag, `SetPixelsState` makes the base address for the offscreen pixel image unpurgeable.

If the `state` parameter does not specify the `pixelsLocked` flag, `SetPixelsState` allows the base address for the offscreen pixel image to be moved.

SPECIAL CONSIDERATIONS

The `SetPixelsState` procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `SetPixelsState` procedure are

Trap macro	Selector
`_QDExtensions`	$0008000E

SEE ALSO

After using `GetPixelsState` and before using `SetPixelsState`, your application can temporarily alter the offscreen graphics world by using the `AllowPurgePixels` procedure (described on page 6-34) to temporarily mark the memory block for its offscreen pixel map as purgeable, the `NoPurgePixels` procedure (described on page 6-35) to make it unpurgeable, the `LockPixels` function (described on page 6-32) to prevent it from being moved, and the `UnlockPixels` procedure (described on page 6-33) to unlock it.

GetPixBaseAddr

You can use the `GetPixBaseAddr` function to obtain a pointer to an offscreen pixel map.

```
FUNCTION GetPixBaseAddr (pm: PixMapHandle): Ptr;
```

`pm`	A handle to an offscreen pixel map. To get a handle to an offscreen pixel map, use the `GetGWorldPixMap` function, described on page 6-31.

DESCRIPTION

The `GetPixBaseAddr` function returns a 32-bit pointer to the beginning of a pixel image. The `baseAddr` field of the `PixMap` record for an offscreen graphics world contains a handle instead of a pointer, which is what the `baseAddr` field for an onscreen pixel map contains. You must use the `GetPixBaseAddr` function to obtain a pointer to the `PixMap` record for an offscreen graphics world.

Your application should never directly access the `baseAddr` field of the `PixMap` record for an offscreen graphics world; instead, your application should always use `GetPixBaseAddr`. If your application is using 24-bit mode, your application should then use the `PixMap32Bit` function (described next) to determine whether a pixel map requires 32-bit addressing mode for access to its pixel image.

If the offscreen buffer has been purged, `GetPixBaseAddr` returns `NIL`.

SPECIAL CONSIDERATIONS

Any QuickDraw routines that your application uses after calling `GetPixBaseAddr` may change the base address for the offscreen pixel image.

The `GetPixBaseAddr` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `GetPixBaseAddr` function are

Trap macro	Selector
`_QDExtensions`	$0004000F

SEE ALSO

See *Inside Macintosh: Memory* for information about determining addressing modes.

PixMap32Bit

You can use the `PixMap32Bit` function to determine whether a pixel map requires 32-bit addressing mode for access to its pixel image.

```
FUNCTION PixMap32Bit (pmHandle: PixMapHandle): Boolean;
```

`pmHandle`	A handle to an offscreen pixel map.

DESCRIPTION

The `PixMap32Bit` function returns `TRUE` if a pixel map requires 32-bit addressing mode for access to its pixel image. If your application is in 24-bit mode, you must change to 32-bit mode.

To get a handle to an offscreen pixel map, first use the `GetGWorldPixMap` function, described on page 6-31. Then supply this handle for the `pm` parameter of `PixMap32Bit`.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for the `PixMap32Bit` function are

Trap macro	Selector
`_QDExtensions`	$00040016

SEE ALSO

See *Inside Macintosh: Memory* for information about determining and setting addressing modes.

Summary of Offscreen Graphics Worlds

Pascal Summary

Constants

```
CONST
   cDepthErr            = -157;  {invalid pixel depth}
   pixPurgeBit          = 0;     {set to to make base address for  }
                                 { offscreen pixel image purgeable}
   noNewDeviceBit       = 1;     {set to not create a new GDevice }
                                 { record for offscreen world}
   useTempMemBit        = 2;     {set to create base address for offscreen }
                                 { pixel image in temporary memory}
   keepLocalBit         = 3;     {set to keep offscreen pixel image in }
                                 { main memory}
   pixelsPurgeableBit   = 6;     {set to make base address for pixel image }
                                 { purgeable}
   pixelsLockedBit      = 7;     {set to lock base address for offscreen }
                                 { pixel image}
   mapPixBit            = 16;    {set by UpdateGWorld if it remapped }
                                 { colors to a new color table}
   newDepthBit          = 17;    {set by UpdateGWorld if it translated }
                                 { pixel map to a different pixel depth}
   alignPixBit          = 18;    {set by UpdateGWorld if it realigned }
                                 { pixel image to onscreen window}
   newRowBytesBit       = 19;    {set by UpdateGWorld if it changed }
                                 { rowBytes field of PixMap record}
   reallocPixBit        = 20;    {set by UpdateGWorld if it reallocated }
                                 { base address for offscreen pixel image}
   clipPixBit           = 28;    {set to clip pixel image}
   stretchPixBit        = 29;    {set to stretch or shrink pixel image}
   ditherPixBit         = 30;    {set to dither pixel image}
   gwFlagErrBit         = 31;    {set by UpdateGWorld if it failed}
```

Data Types

```
TYPE GWorldPtr = CGrafPtr;

TYPE GWorldFlags =
   SET OF (
      pixPurge,         {specify to NewGWorld to make base address for }
                        { offscreen pixel image purgeable}
      noNewDevice,      {specify to NewGWorld to not create a new GDevice }
                        { record for offscreen world}
      useTempMem,       {specify to NewGWorld to create base address for }
                        { offscreen pixel image in temporary memory}
      keepLocal,        {specify to NewGWorld to keep offscreen pixel image }
                        { in main memory}
      gWorldFlag4,      {reserved}
      gWorldFlag5,      {reserved}
      pixelsPurgeable,  {returned by GetPixelsState to indicate that base }
                        { address for offscreen pixel image is purgeable; }
                        { specify to SetPixelsState to make base address }
                        { for pixel image purgeable}
      pixelsLocked,     {returned by GetPixelsState to indicate that base }
                        { address for offscreen pixel image is locked; }
                        { specify to SetPixelsState to lock base address }
                        { for offscreen pixel image}
      gWorldFlag8,      {reserved}
      gWorldFlag9,      {reserved}
      gWorldFlag10,     {reserved}
      gWorldFlag11,     {reserved}
      gWorldFlag12,     {reserved}
      gWorldFlag13,     {reserved}
      gWorldFlag14,     {reserved}
      gWorldFlag15,     {reserved}
      mapPix,           {returned by UpdateGWorld if it remapped colors to }
                        { a new color table}
      newDepth,         {returned by UpdateGWorld if it translated pixel }
                        { map to a different pixel depth}
      alignPix,         {returned by UpdateGWorld if it realigned pixel }
                        { image to onscreen window}
      newRowBytes,      {returned by UpdateGWorld if it changed rowBytes }
                        { field of PixMap record}
      reallocPix,       {returned by UpdateGWorld if it reallocated }
                        { base address for offscreen pixel image}
      gWorldFlag21,     {reserved}
      gWorldFlag22,     {reserved}
      gWorldFlag23,     {reserved}
      gWorldFlag24,     {reserved}
      gWorldFlag25,     {reserved}
      gWorldFlag26,     {reserved}
      gWorldFlag27,     {reserved}
      clipPix,          {specify to UpdateGWorld to update and clip pixel }
                        { image}
      stretchPix,       {specify to UpdateGWorld to update and stretch or }
                        { shrink pixel image}
      ditherPix,        {specify to UpdateGWorld to dither pixel image}
      gwFlagErr,        {returned by UpdateGWorld if it failed}
   );
```

Routines

Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
FUNCTION NewGWorld           (VAR offscreenGWorld: GWorldPtr;
                              pixelDepth: Integer; boundsRect: Rect;
                              cTable: CTabHandle; aGDevice: GDHandle;
                              flags: GWorldFlags): QDErr;
FUNCTION NewScreenBuffer     (globalRect: Rect;
                              purgeable: Boolean; VAR gdh: GDHandle;
                              VAR offscreenPixMap: PixMapHandle): QDErr;
FUNCTION NewTempScreenBuffer
                             (globalRect: Rect;
                              purgeable: Boolean;
                              VAR gdh: GDHandle;
                              VAR offscreenPixMap: PixMapHandle): QDErr;
FUNCTION UpdateGWorld        (VAR offscreenGWorld: GWorldPtr;
                              pixelDepth: Integer; boundsRect: Rect;
                              cTable: CTabHandle; aGDevice: GDHandle;
                              flags: GWorldFlags): GWorldFlags;
PROCEDURE DisposeGWorld      (offscreenGWorld: GWorldPtr);
PROCEDURE DisposeScreenBuffer
                             (offscreenPixMap: PixMapHandle);
```

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

```
PROCEDURE GetGWorld          (VAR port: CGrafPtr; VAR gdh: GDHandle);
PROCEDURE SetGWorld          (port: CGrafPtr; gdh: GDHandle);
FUNCTION GetGWorldDevice     (offscreenGWorld: GWorldPtr): GDHandle;
```

Managing an Offscreen Graphics World's Pixel Image

```
FUNCTION GetGWorldPixMap     (offscreenGWorld: GWorldPtr): PixMapHandle;
FUNCTION LockPixels          (pm: PixMapHandle): Boolean;
PROCEDURE UnlockPixels       (pm: PixMapHandle);
PROCEDURE AllowPurgePixels   (pm: PixMapHandle);
PROCEDURE NoPurgePixels      (pm: PixMapHandle);
FUNCTION GetPixelsState      (pm: PixMapHandle): GWorldFlags;
PROCEDURE SetPixelsState     (pm: PixMapHandle; state: GWorldFlags);
FUNCTION GetPixBaseAddr      (pm: PixMapHandle): Ptr;
FUNCTION PixMap32Bit         (pmHandle: PixMapHandle): Boolean;
```

C Summary

Constants

```
enum {                          /* bit assignments for GWorldFlags data type */
   pixPurgeBit          = 0,  /* set to to make base address for
                                 offscreen pixel image purgeable */
   noNewDeviceBit       = 1,  /* set to not create a new GDevice
                                 record for offscreen world */
   useTempMemBit        = 2,  /* set to create base address for offscreen
                                 pixel image in temporary memory */
   keepLocalBit         = 3,  /* set to keep offscreen pixel image in
                                 main memory */
   pixelsPurgeableBit   = 6,  /* set to make base address for pixel image
                                 purgeable */
   pixelsLockedBit      = 7,  /* set to lock base address for offscreen
                                 pixel image */
   mapPixBit            = 16, /* set by UpdateGWorld if it remapped
                                 colors to a new color table */
   newDepthBit          = 17, /* set by UpdateGWorld if it translated
                                 pixel map to a different pixel depth */
   alignPixBit          = 18, /* set by UpdateGWorld if it realigned
                                 pixel image to onscreen window */
   newRowBytesBit       = 19, /* set by UpdateGWorld if it changed
                                 rowBytes field of PixMap record */
   reallocPixBit        = 20, /* set by UpdateGWorld if it reallocated
                                 base address for offscreen pixel image */
   clipPixBit           = 28, /* set to update and clip pixel image */
   stretchPixBit        = 29, /* set to update and stretch or shrink pixel
                                 image */
   ditherPixBit         = 30, /* set to dither pixel image */
   gwFlagErrBit         = 31  /* set by UpdateGWorld if it failed */
};

enum {                  /* constants for GWorldFlags data type */
   pixPurge          = 1 << pixPurgeBit,
   noNewDevice       = 1 << noNewDeviceBit,
   useTempMem        = 1 << useTempMemBit,
   keepLocal         = 1 << keepLocalBit,
   pixelsPurgeable   = 1 << pixelsPurgeableBit,
   pixelsLocked      = 1 << pixelsLockedBit,
   mapPix            = 1 << mapPixBit,
   newDepth          = 1 << newDepthBit,
   alignPix          = 1 << alignPixBit,
   newRowBytes       = 1 << newRowBytesBit,
   reallocPix        = 1 << reallocPixBit,
   clipPix           = 1 << clipPixBit,
   stretchPix        = 1 << stretchPixBit,
   ditherPix         = 1 << ditherPixBit,
   gwFlagErr         = 1 << gwFlagErrBit
};

enum {
   cDepthErr         = -157   /* invalid pixel depth */
};
```

Data Types

```
typedef CGrafPtr GWorldPtr;

typedef unsigned long GWorldFlags;
```

Functions

Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
pascal QDErr NewGWorld      (GWorldPtr *offscreenGWorld, short PixelDepth,
                             const Rect *boundsRect, CTabHandle cTable,
                             GDHandle aGDevice, GWorldFlags flags);
pascal QDErr NewScreenBuffer
                            (const Rect *globalRect, Boolean purgeable,
                             GDHandle *gdh, PixMapHandle *offscreenPixMap);
pascal QDErr NewTempScreenBuffer
                            (const Rect *globalRect, Boolean purgeable,
                             GDHandle *gdh, PixMapHandle *offscreenPixMap);
pascal GWorldFlags UpdateGWorld
                            (GWorldPtr *offscreenGWorld, short pixelDepth,
                             const Rect *boundsRect, CTabHandle cTable,
                             GDHandle aGDevice, GWorldFlags flags);
pascal void DisposeGWorld   (GWorldPtr offscreenGWorld);
pascal void DisposeScreenBuffer
                            (PixMapHandle offscreenPixMap);
```

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

```
pascal void GetGWorld       (CGrafPtr *port, GDHandle *gdh);
pascal void SetGWorld       (CGrafPtr port, GDHandle gdh);
pascal GDHandle GetGWorldDevice
                            (GWorldPtr offscreenGWorld);
```

Managing an Offscreen Graphics World’s Pixel Image

```
pascal PixMapHandle GetGWorldPixMap
                            (GWorldPtr offscreenGWorld);
pascal Boolean LockPixels   (PixMapHandle pm);
pascal void UnlockPixels    (PixMapHandle pm);
pascal void AllowPurgePixels
                            (PixMapHandle pm);
pascal void NoPurgePixels   (PixMapHandle pm);
pascal GWorldFlags GetPixelsState
                            (PixMapHandle pm);
pascal void SetPixelsState  (PixMapHandle pm, GWorldFlags state);
pascal Ptr GetPixBaseAddr   (PixMapHandle pm);
pascal Boolean PixMap32Bit  (PixMapHandle pmHandle);
```

Assembly-Language Summary

Trap Macros Requiring Routine Selectors

`_QDExtensions`

Selector	Routine
$00160000	`NewGWorld`
$00040001	`LockPixels`
$00040002	`UnlockPixels`
$00160003	`UpdateGWorld`
$00040004	`DisposeGWorld`
$00080005	`GetGWorld`
$00080006	`SetGWorld`
$0004000B	`AllowPurgePixels`
$0004000C	`NoPurgePixels`
$0004000D	`GetPixelsState`
$0008000E	`SetPixelsState`
$0004000F	`GetPixBaseAddr`
$000E0010	`NewScreenBuffer`
$00040011	`DisposeScreenBuffer`
$00040012	`GetGWorldDevice`
$000E0015	`NewTempScreenBuffer`
$00040016	`PixMap32Bit`
$00040017	`GetGWorldPixMap`

Result Codes

`noErr`	0	No error
`paramErr`	–50	Illegal parameter
`cNoMemErr`	–152	Failed to allocate memory for structures
`cDepthErr`	–157	Invalid pixel depth