New Technical Notes

Macintosh

Developer Support

Graphics Devices Manager Q&As

Devices M.DV.GrDevMgr.Q&As

Revised by: Developer Support Center October 1992
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As and Q&As revised this month are marked with a bar in the side margin.

Macintosh slot number and gRefNum
Written: 11/1/90
Last reviewed: 2/20/91

How can I determine what hardware device is driven by a particular Macintosh gDevice? I can call _GetDeviceList and _GetNextDevice to get the driver reference number of each gDevice but not the hardware ID. The system 'scrn' resource for the hardware ID of each device doesn't give me the driver reference number. Parsing the system 'scrn' resource, then counting that many with _GetDeviceList and _GetNextDevice produces a device list that doesn't seem to be consistently in the same order as the 'scrn' resource.

___

My impression is that you don't make the connection between the gRefNum and the slot number. The following code shows how to do it:

```
DCEHand = (AuxDCEHandle) &(*UTableBase[-DevInfoList[index].gdRefNum - 1]);

/* Note that the previous could have been accomplished by
DCEHand = (AuxDCEHandle) GetDCtlEntry(DevInfoList[index].gdRefNum);
But what good is C if you cannot do twisted things with it!
```

```
   */

   DevInfoList[devCount].gdSlot = (*DCEHand) -> dCtlSlot; /* get slot no. */
```

Once you have the slot number you can call the Slot Manager to get the board name and other information you may need to identify the device, as in the following code:

```
 /* to find the board name we have to go into the slot manager guts, and
 get the name from the board sResource */
   spB.spSlot = DevInfoList[devCount].gdSlot;
   spB.spID = 0;
   spB.spExtDev = 0;
   spB.spCategory = 1;
   spB.spCType = 0;
   spB.spDrvrSW = 0;
   spB.spDrvrHW = 0;

   if (! SNextTypesRsrc(&spB) )
     {
       spB.spID = 2;                /* Get Board ID Cards and Drivers */
       if ( ! SGetCString(&spB) )
         {                          /* let C unravel its own strings */
           for (count = 0; DevInfoList[devCount].bdName[count+1] =
                                    *((char *)(spB.spResult)+count);count++);
             DevInfoList[devCount].bdName[0] = count;

         }
     }
```

X-Refs:
Inside Macintosh Volume II, pages 190-192
Inside Macintosh Volume V, page 424
Designing Cards and Drivers for the Macintosh Family

Code snippet for getting Macintosh display information

Written: 11/28/90
Last reviewed: 1/16/91

In my Macintosh system, I have two screens. The main screen in a 19" monochrome. The second screen, to the left of the main one, is color. How can my program know whether or not a color display is available, its depth, and its coordinates?

—

The following bit of code will show you basically how to walk the GDevice list and find the information you want from it. Most of this information can be found on pages 119 and 124 of Inside Macintosh Volume V.

```
count = 0;
GDevHand = GetDeviceList();
ScreenRect[count] = (*(*gDevHand)->gdPMap)->bounds;
ScreenDepth[count] = (*(*gDevHand)->gdPMap)->pixelSize;
ScreenColor[count] = TestDeviceAttribute (GDevHand, gdDevType);

While ((GDevHand = GetNextDevice(GDevHand)) != nil) {
    ++count;
    ScreenRect[count] = (*(*gDevHand)->gdPMap)->bounds;
    ScreenDepth[count] = (*(*gDevHand)->gdPMap)->pixelSize;
    ScreenColor[count] = TestDeviceAttribute (GDevHand, gdDevType);
    }
```

Technique for Macintosh GC card GWorld access

Written: 3/15/91
Last reviewed: 8/1/91

How can a Macintosh in 24-bit addressing mode read from disk into a GWorld? If the GC card is installed, sometimes the GWorld is a 32-bit one cached on the card. How do we ensure the GWorld will be in main memory?

—

When you create your GWorld, set the keepLocal flag in the flags field of the NewGWorld call. This ensures that the newly created GWorld will be in main memory where you can access it. Read your data into the GWorld and then clear the keepLocal flag with a call to SetPixelState. This will issue a "GWorld has been updated" type of message, causing the GWorld to be cached off to the GC card, giving you the best performance. When you want to load another image, call SetPixelState again, setting the keepLocal flag to bring the GWorld back to main memory. Using this technique you'll be able to load your GWorld and cache it too, and you won't need to recreate the GWorld each time you want to load more data into it.

Macintosh GWorld baseAddresses
Written: 3/15/91
Last reviewed: 7/29/91

How can I transform the content of a GWorld on a NuBus™ card? The idea is to clone the GWorld, copy the data to the card, transform them, patch the address into the GWorld, and display them directly on the screen. I´d like to do it in a way that works well with the 8•24 GC and similar asynchronous QuickDraw stuff.

—

NewGWorld allocates off-screen buffers simply by using the same Memory Manager calls that you and I make. To actually allocate the memory it simply calls NewHandle to allocate the buffer in your application's heap if you have the useTempMem (née useMFTemp) bit clear. It then tries to move it as high in your heap as possible by calling MoveHHi. If you have the useTempMem bit set, then NewGWorld uses the temporary memory calls to allocate the off-screen buffer in temporary memory, and then it tries to move it as high as possible in the temporary memory space. That's really all there is to it. The GWorld's PixMap, GDevice and CGrafPort are allocated similarly—they're all allocated in your heap using regular Memory Manager calls with no special options, patches, or other nefarious tricks.

None of this changes when you have the 8•24 GC software active—all memory is allocated out of your application's heap. Once you start drawing into the GWorld, though, the GC software can copy the parts of a GWorld to the 8•24 GC memory. The GWorld and its parts go still occupy your heap's memory though, regardless of whether it's cached on the 8•24 GC card or not.

If you have a NuBus card with gobs of memory, NewGWorld can't take advantage of it because the Memory Manager calls that it uses can't allocate memory on NuBus memory

cards. There are no options to NewGWorld or any other GWorld calls that let you say, “there’s lots of memory over on this NuBus card, all for you.” That means that GWorlds aren’t appropriate if you want to have control over where the off-screen buffer is allocated. Conceivably, you could allocate a GWorld, stuff the address of your NuBus memory card into the baseAddr of your GWorld’s PixMap, and then put the constant baseAddr32 into its pmVersion field, but engineering here didn’t feel very comfortable with that idea because of compatibility concerns.

QuickDraw is the only thing that’s supposed to know how GWorlds are constructed. We know that they’re CGrafPorts and we can get their PixMap, GDevice, and off-screen buffer, but we can’t make any assumptions about how they were allocated and where they are. For example,

we know that the off-screen buffer is allocated as a handle now, but that won't necessarily be the case in the future. There's no guaranteed way to tell which way it was allocated, or even if NewGWorld uses the Memory Manager to allocate it at all (which it always does currently of course). Even the GWorld's CGrafPort is allocated as a handle which just happens to be always locked. If you try to dispose of a GWorld in which you've modified the baseAddr, you'll need DisposeGWorld to make sure everything is deallocated properly, but it'll act unpredictably when it tries to deallocate the off-screen buffer.

So if you want to use the memory on your NuBus memory card, you're going to have to create your own PixMap, color table (if it needs one), GDevice, and CGrafPort. The Technical Note "Principia Off-Screen Graphics Environments" covers creating your own PixMap, CGrafPort, and color table, but it has the same depth and equivalent color table to the screen, so it just steals a screen's GDevice. I think it's always a good idea to create your own GDevice when you draw off screen. If you use a screen's GDevice, then you have to depend on that GDevice's depth and color table. By creating your own GDevice, your off-screen drawing can use any depth and color table you want at any time, and Color QuickDraw won't choke.

To create your own GDevice, it's better not to use NewGDevice because it always creates the GDevice in the system heap, and it's just better to keep your own data structures in your own heap. Here's what you should set each of its fields to be:

gdRefNum - Your GDevice has no driver, so just set this to zero.

gdID - It doesn't matter what you set this to; might as well set it to zero.

gdType - Set this to 2 if your off screen uses direct colors (16 or 32 bits per pixel) or 0 if your off screen uses color table (1 through 8 bits per pixel).

gdITable - Allocate a small (maybe just 2-byte) handle for this field. After you're done setting up this GDevice and your off-screen PixMap, color table (if any) and CGrafPort, then set this GDevice as the current GDevice by calling SetGDevice, and then call MakeITable, passing it NIL for both the color table and inverse table parameters, and 0 for the preferred inverse table resolution.

gdResPref - Probably more than 99.9% of all inverse tables out there have a resolution of 4. Unless you have some reason not to, I'd recommend the same here.

gdSearchProc - Set to NIL. Use AddSearch if you want to use a SearchProc.

gdCompProc - Set to NIL. Use AddComp if you want to use a CompProc.

gdFlags - Set to 0 initially, and then use SetDeviceAttribute after you've set up the rest of this GDevice.

gdPMap - Set this to be a handle to your off-screen PixMap.

gdRefCon - Set this to whatever you want.

gdNextGD - Set this to nil.

gdRect - Set this to be equal to your off-screen PixMap's bounds.

gdMode - Set this to -1. Why? I'm not sure. This is intended for GDevices with drivers anyway.

gdCCBytes - Set to 0.

gdCCDepth - Set to 0.

gdCCXData - Set to 0.

gdCCXMask - Set to 0.

gdReserved - Set to 0.

For gdFlags, you should use SetDeviceAttribute to set the noDriver bit and the gdDevType bit. You should set the gDevType bit to 1 even if you have a monochrome color table. The 0 setting was only used for the days when monochrome mode was handled by the video driver, which 32-Bit QuickDraw eliminated. Your GDevice doesn't have a driver anyway.

One last warning is that you should set the pmVersion field of your PixMap to be the constant baseAddr32 (equals 4). That tells Color QuickDraw to use 32-bit addressing mode to access your off-screen buffer, and that's a requirement if your off-screen buffer is allocated on a NuBus card.

UpdateGWorld dithering bug workaround
Written: 11/14/91
Last reviewed: 12/12/91

UpdateGWorld doesn't seem to respond to the ditherPix flag unless color depth changes. The return flag after changing my color table is 0x10000, indicating that color mapping happened but not dithering. Is this a bug?

—

Yes, this is a bug.

UpdateGWorld ignores dithering if no depth change is made. UpdateGWorld probably won't be changed in the near future. The workaround, therefore, is as follows:

1. Ceate a new pixmap with the the new color table
2. Call CopyBits to transfer your image to the newly created pixmap with dithering from the original GWorld's pixmap
3. Update the GWorld with the new color table without using ditherPix
4. Use CopyBits from the newly created pixmap without dithering back to the GWorld.

This will give you the same effect as UpdateGWorld with ditherPix.

Drawing into GWorld after using UpdateGWorld

Written: 11/21/91
Last reviewed: 12/11/91

When I resize my realtime animation window, I call UpdateGWorld with the new size, and after that any drawing into the GWorld has no effect—almost like the call to SetGWorld does nothing. This same code works perfectly in System 7. What could cause this?

—

You probably can't draw anything into your GWorld after using UpdateGWorld to resize it is because of the clip region of your GWorld. Under system software versions before 7.0, UpdateGWorld always resizes the GWorld's clip region proportional to the amount that the GWorld itself is resized. Unfortunately, NewGWorld initializes the clip region of the GWorld to the entire QuickDraw coordinate plane, [T:-32767 L:-32767 B:32767 R:32767]. If UpdateGWorld resizes any of these coordinates so that they fall outside of this range, then the coordinates wrap around to the other end of the signed integer space, and that makes the clip region empty. Empty clip regions stop any drawing from happening.

The change to system software version 7.0 is that now UpdateGWorld explicitly checks for a clip region that looks like [T:-32767 L:-32767 B:32767 R:32767]. If it finds this, then it doesn't resize the clip region. Otherwise, UpdateGWorld acts the same way that it did before 7.0.

One of my mottoes in life is, "never give QuickDraw a chance to do the wrong thing." In keeping with that, I always explicitly set the clip regions of GWorlds whenever it could change. So after calling NewGWorld, set its clip region to be coincident with its portRect. After calling UpdateGWorld to resize it, set its clip region to be coincident with its new portRect. That way, you'll always have a known environment and you won't have to worry about the change that was made in System 7.0, and you'll be less susceptible to bugs in this area in the future. It's an easy enough workaround so that it has no real penalty.

Drawing dimmed outline across screens with different depths
Written: 12/9/91
Last reviewed: 1/27/92

When the OK button is disabled in the System 7 Standard File dialog box, it's drawn in gray. I was looking for sample code on how to do this in a way that is appropriate for multiple screens at various color depths. For example, how should you draw the outline if you have a OK button in a movable modal dialog box (with half the OK button on an 8-bit color screen and the other half on a 1-bit monochrome screen)?

—

There are two ways to answer your question on how to draw the gray (dimmed) outline across several screens in different depths: one uses MakeRGBPat (Inside Macintosh Volume V, page V-73), the other DeviceLoop (Inside Macintosh Volume VI, page 21-23). Look for GrayishOutline.p in the Snippets folder on the developer CD for a code sample that demonstrates both ways.

Testing for GWorld availability
Written: 1/16/92
Last reviewed: 2/17/92

How do I tell if GWorld calls such as NewGWorld are available? Is the existence of 32-Bit Color QuickDraw the only condition?

—

Presence of 32-Bit QuickDraw is a sufficient but not a necessary condition for GWorld support. Under System 7, GWorld routines and certain other color calls are allowed on 68000-based Macintosh models, even though Color QuickDraw is not available. However, special precautions must be taken to use GWorlds in a compatible way.

On a Macintosh without Color QuickDraw running System 7.0, the GWorld created by NewGWorld uses an extended GrafPort, not a CGrafPort. NewGWorld will return an error if asked to create a GWorld with depth greater than 1 unless Color QuickDraw is available.

The PixMap field of a GWorld should not be accessed directly; instead, use the function GetGWorldPixMap. Similarly, to obtain the address of the PixMap data, use GetPixBaseAddr rather than access the baseAddr field of the PixMap. These routines are documented in the Graphics Devices chapter of Inside Macintosh Volume VI. One-bit GWorlds are also discussed in the article, "QuickDraw's CopyBits Procedure: Better Than Ever in System 7.0," from issue 6 of develop.

Unfortunately, GetGWorldPixMap does not work properly under 32-Bit QuickDraw 1.2, so use theGWorldPtr^.portPixMap instead if and only if Gestalt reports the QuickDraw version as greater than or equal to gestalt32BitQD and less than gestalt32BitQD13.

The proper test for the availability GWorlds is

```
retCode1 = Gestalt(gestaltQuickdrawVersion, &qdVersResponse);
retCode2 = Gestalt(gestaltSystemVersion, &sysVersResponse);
if ((retCode1 == noErr) &&
   (retCode2 == noErr) &&
   ((qdVersResponse >= gestalt32BitQD) ||
    (sysVersResponse >= 0x0700))) {
   /* do GWorld stuff */
}
```

This is also the appropriate test for other "color" routines available on all Macintosh models under System 7: RGBForeColor, RGBBackColor, GetForeColor, GetBackColor, and QDError. (QDError returns zero if Color QuickDraw is not present.)

Alternatively, if color or greyscale GWorlds are required, check bit gestaltHasDeepGWorlds (bit 1) of the response to selector gestaltQuickdrawFeatures. It is set if GWorlds with true CGrafPorts are available. Incidentally, the gestaltHasColor bit for gestaltQuickdrawFeatures erroneously returns true for black-and-white machines under System 7.0.

Changing the color table of an existing Macintosh GWorld

Written: 5/21/92
Last reviewed: 9/15/92

How do I update the color table of my off-screen graphics world without destroying the picture?

—

The recommended approach for changing the color table of an existing GWorld involves calling UpdateGWorld, passing either clipPix or stretchPix for gWorldFlags. When passed either of these constants, QuickDraw knows to update the pixels of the pixMap image. Even though the actual image isn't changed, the flags are still needed to remap the pixels to their new colors.