**GetMaxDevice** Gives a handle to device with the deepest pixMap

#include <Quickdraw.h> **Graphics Devices**

| | | |
|---|---|---|
| GDHandle | **GetMaxDevice(** *globalRect* **);** | |
| Rect | **globalRect* ; | Entire drawing area |
| | ***returns*** | a handle leading to the device having the maximum pixel depth |

**GetMaxDevice** provides a handle to the greatest pixel-depth gDevice in the entire global rectangle.

*globalRect* is the rectangle in global coordinates.

**Returns**: a GDHandle; a handle leading to the device in the list with the most available colors.

Notes: You can use this routine when creating an offscreen PixMap for subsequent transfer to visible display areas. Alternatively, you might want control of a drawing's colors, especially if it's going to be printed on a device with a different color table than the screen. In either case, you need control of the color and, hence, the gDevices.

In either case, if your application spans devices with varying color capabilities, **GetMaxDevice** lets you use a PixMap to set up the colors to take advantage of the best color table from among all the devices.

If you decide to use a PixMap as for a device with the greatest pixel depth, you have to obtain an offscreen CGrafPort for that particular monitor. What's involved in that is: 1)save the current gDevice with the **GetGDevice** procedure; 2) call **GetMaxDevice** ; 3) use **SetGDevice** to make the new choice the currently active screen; 4) call **OpenCPort** to create the new CGrafPort; and 5) call **SetGDevice** yet again to bring back the formerly active gDevice. Because **OpenCPort** uses the global theGDevice to initialize its PixMap the current CGrafPort and the deepest screen become one and the same.

All of the above, however, is simply preparation. Now you get storage for the pixels by defining the PixMap's boundaries to the image's height and width, and setting the rowBytes to take into account the particular pixel size being used on that device--( ( width*portPixMap^^.pixSize )+ 15 )DIV 16 * 2 --always bearing in mind that rowBytes has to be even to work at all and a multiple of 4 to work best. Now define the interior of the PixMap by setting portRect. The amount of storage is now height times the value of rowBytes. If you allocate the storage as a handle, then your application can lock the handle and place a pointer to it in the PixMap's base address.

Next, in order to draw to the CGrafPort, you save the current gDevice, set theGDevice to be the maximum device, and restore theGDevice when the drawing operation is complete.

After you've done all of this preparation, be sure to protect it since all of the above can be brought to nothing if the user changes the depth of the screen or moves the window across device boundaries. Include an environment test in your application's update routine by having it compare the ctSeed field of the new maximum device with that of the old one. If they're different, the program should see if the screen depth is different. If that's different too, the PixMap needs to be reallocated. If the depth is the same but the color table is different, the program will have to redraw the objects in the PixMap again.

What if your application is optimized for speed instead of richness of color? If so, it needs to examine each element in the device list to find the one with the biggest area by calling **GetDeviceList**, intersecting the first device's drawing area with the total window, and calling **GetNextDevice** to examining the next entry on the list. Use the memory allocation procedure above to create the storage area for the CGrafPort.

The last case where you might want to draw to an offscreen device would be when you're creating an image for output to a color printer. The process for creating the PixMap is similar to what's already been described. However, you have to describe a new drawing environment instead of letting an existing gDevice do it. You do this by calling **NewGDevice** to obtain a new gDevice data structure. Your program initializes all of the PixMap and color table fields (see **Color Quickdraw** for instructions on how to do this), calls **MakeITable** to build the device's inverse color table (see the information in **Color Manager**) and sets the gDevice as the current device before drawing to the offscreen PixMap. The colors will then be drawn as defined by the gDevice.