Optimizing Display Modes and Window Arrangement With the Display Manager

(Legacy)

Carbon > User Experience

2007-05-03

Apple Inc.
© 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

.Mac is a registered service mark of Apple Inc.

Apple, the Apple logo, Carbon, Mac, Mac OS, Macintosh, Power Mac, PowerBook, Quartz, and QuickDraw are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Figures, Tables, and Listings

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Introduction to Optimizing Display Modes and Window Arrangement With the Display Manager

Important: The Display Manager is deprecated as of Mac OS X v10.4. You should use Quartz Display Services instead (as described in *Quartz Display Services Programming Topics*).

The Display Manager is a legacy technology that supported dynamic changes to the arrangement and display modes of the displays attached to a user's computer. The Display Manager was included in Carbon to facilitate porting older applications to Mac OS X. While the Display Manager is still supported, it has been deprecated and should not be used for new software development. If you have an existing application that still uses the Display Manager, you should update your application to use Quartz Display Services, the replacement technology in Mac OS X.

The target audience for this legacy document was developers writing applications for Mac OS 9 and earlier. This document is not relevant in Mac OS X.

Organization of This Document

This document contains the following chapters:

"About the Display Manager" (page 9) describes the features of the Display Manager and helps you determine whether your application needs to use its API.

"Using the Display Manager" (page 19) describes some of the tasks you can perform with the Display Manager.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

About the Display Manager

This chapter explains how the Display Manager allows users to dynamically change the arrangement and display modes of the monitors attached to their computers. For example, users can move their displays, add or remove displays, switch displays to higher or lower screen resolutions, and move the menu bar from one display to another—all without restarting their computers. When the user changes the display environment (as when disconnecting a display, for example), the Display Manager further assists the user by repositioning standard windows so that the user can find them in the new display environment.

This chapter helps you determine whether your application must move its own windows instead of relying on the Display Manager to move them. For example, if your application implements a tool palette that lacks a title bar, and the user disconnects the monitor that displays the tool palette, your application must move your tool palette to the main screen where the user can find it. Because the Display Manager never resizes windows, this chapter helps you determine whether to resize your application's windows after a display configuration change.

The Display Manager is available on all Power Macintosh computers and on color-capable Macintosh computers running system software version 7.5 and later. Applications that use only the standard window definition functions provided by the Window Manager generally do not need to use the Display Manager.

Users indirectly inform the Display Manager of changes they wish to make to their display environment by using the Monitors control panel or by adding and removing additional displays. The Monitors control panel in turn calls the Display Manager to change the display environment. The Display Manager sends an Apple event—the Display Notice event—to notify applications that it changed the display environment. In addition, the Display Manager generates an update event to notify all current applications to update their windows.

The Display Manager provides your application with functions that obtain `GDevice` structures for the video devices controlling the displays connected to the user's computer system. When repositioning a window, for example, your application can use the `GDevice` structures stored in the device list to determine which video device supports the largest display area or the greatest pixel depth.

This chapter explains the capabilities of the Display Manager and describes its default behavior when repositioning windows. This chapter helps determine whether your application needs to perform its own window positioning or sizing. If your application needs to perform its own window management in a changing environment, the next chapter, "Using the Display Manager," discusses how your application can determine if the user changed the display environment and how to manage its windows accordingly.

Overview

The Display Manager is a set of system software functions that support dynamic changes to the arrangement and display modes of the displays attached to a user's computer. (This book uses the term displays to represent output devices—such as video monitors and flat-panel displays—on which applications can show interactive visual information to the user. A video device is the hardware, such as the plug-in video card or the built-in video interface, that controls a display.)

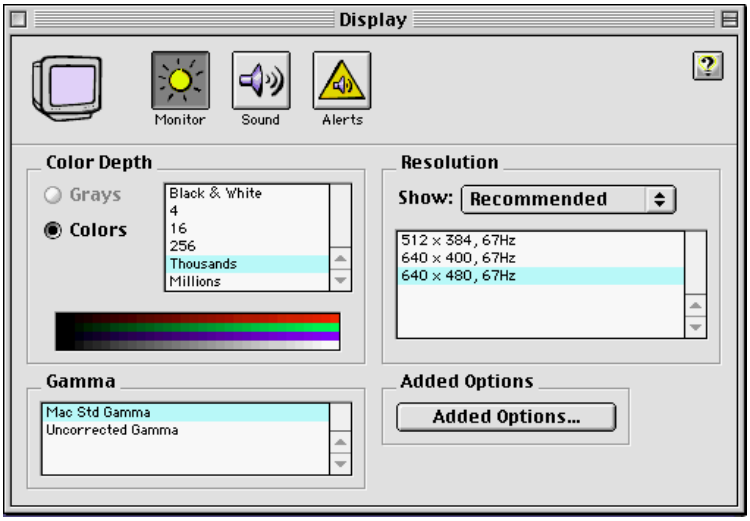
Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

The Monitors control panel mostly uses the Display Manager functions. After opening the Monitors control panel, the user can choose to

- move displays
- switch multiple-resolution displays to use higher or lower screen resolutions
- move the menu bar from one display to another
- select different pixel depths for video devices that support multiple depths

For example, a user can use a PowerBook computer that comes with an external video port to attach a second display. After the user opens the Monitors control panel, the user can move the menu bar from one display to another and the menu bar immediately moves to the user's desired location without the user restarting the computer.

Figure 1-1 The Monitors control panel



The user can also add or remove displays without restarting the computer. For example, a user can attach an external monitor to a sleeping PowerBook computer, wake the computer, and use both the external and built-in displays. If the user puts the PowerBook computer to sleep, detaches the external monitor, then wakes the computer, the Display Manager automatically moves windows that previously appeared on the external monitor onto the PowerBook built-in display.

The next several sections illustrate the default window positioning behaviors of the Display Manager.

When the User Removes a Display

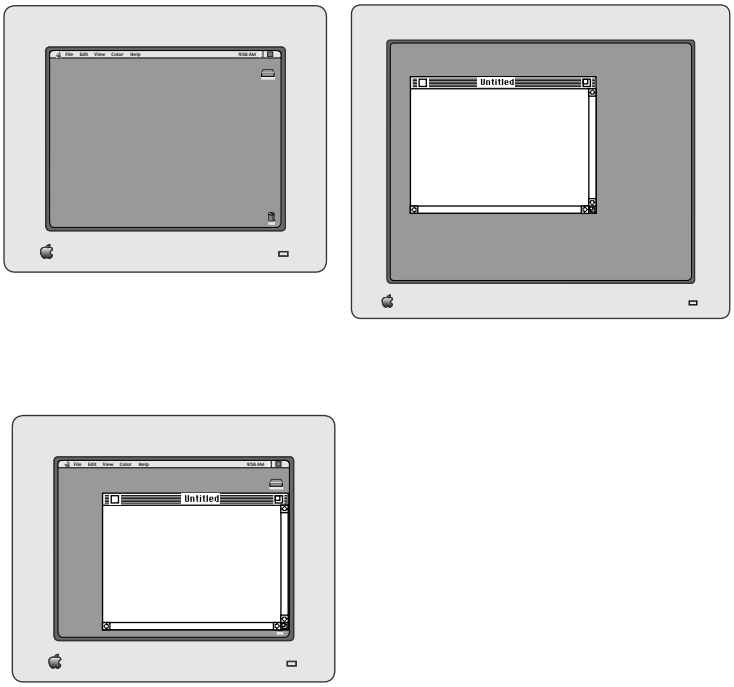
When a user removes a display, the Display Manager moves the windows that previously appeared on the disconnected display to the next closest display.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

The Display Manager attempts to center the window of an alert or modal dialog box on the next closest display. If the alert or modal dialog box is larger than the screen, the Display Manager aligns its lower-left corner with the lower-left corner of the next closest display, thereby providing access to the area of the alert or modal dialog box with the OK and Cancel buttons.

The Display Manager assumes that any other type of window has a standard title bar. As illustrated in Figure 1-2 and Figure 1-3, the Display Manager then moves the window to the closest display by the shortest distance necessary to show the entire title bar.

Figure 1-2 Default window repositioning when the user removes the right display

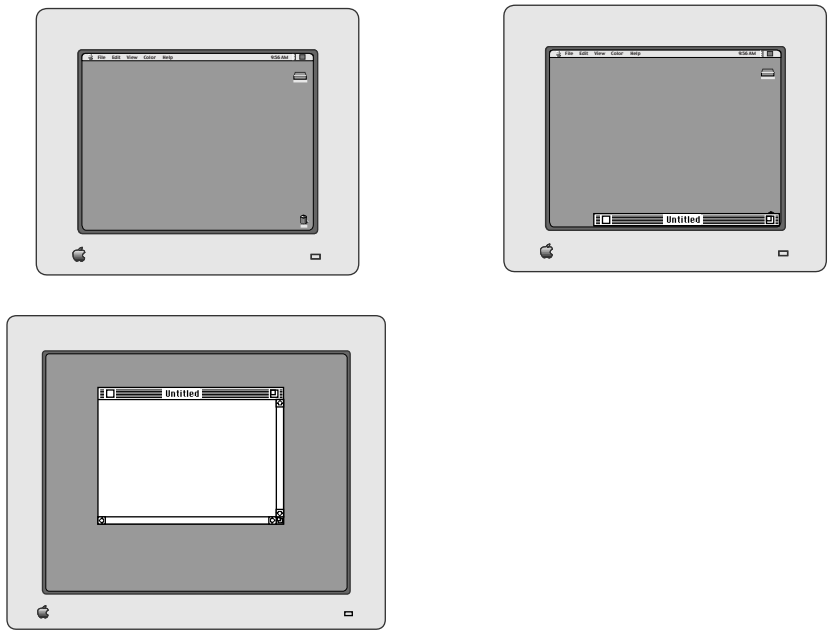


As shown in Figure 1-3, the content region of the window may still lie offscreen; but in a standard window, the user has access to the drag region of the title bar and to the zoom box. The user can therefore easily move the entire window onto the screen.

If the window is wider than the screen, the Display Manager fits the area in the title bar where the close box should appear onscreen.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Figure 1-3 Default window repositioning when the user removes the bottom display



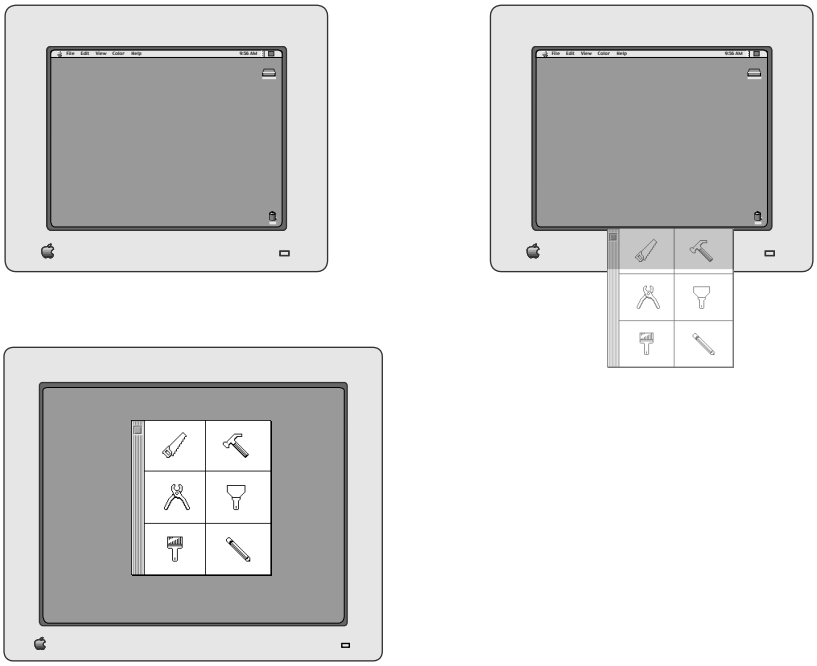
Display Manager Problems Moving Windows

When repositioning any window other than a window of type `dBoxProc`, the Display Manager assumes that the window has a standard title bar and moves the window to the closest display so that the title bar appears to the user. However, if the window does not have a title bar, the Display Manager may move the window to a position where the user cannot see it.

For example, on the left side of Figure 1-4 a window containing a tool palette and a nonstandard drag region appears in the lower display. When the user removes the lower display, as shown in the right side of the figure, the Display Manager moves the tool palette onto the main screen by the shortest distance necessary to display a standard title bar for the window. However, the window does not have a standard title bar, and so no part of the window appears onscreen. Applications that use windows without standard title bars must reposition their own windows as described in the chapter “Using the Display Manager.”

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Figure 1-4 A problem with repositioning a nonstandard window

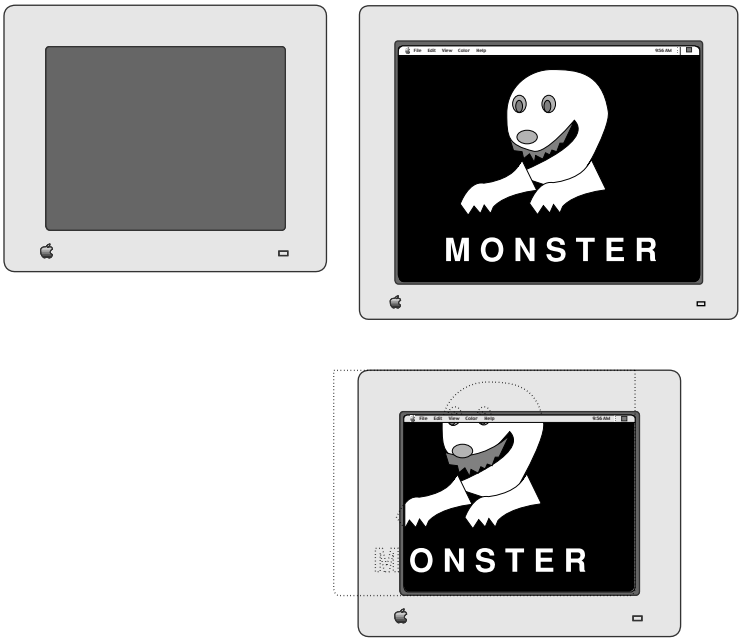


The Display Manager makes no attempt to stack or tile windows so that the user can see all of their titles bars simultaneously. Multiple windows repositioned by the Display Manager may obscure each other's title bars.

The Display Manager never resizes windows. Because of this, fixed size windows can present a problem. If a fixed size window appears on a large display, and the user removes that display, only part of the window appears when the Display Manager repositions it on a smaller display. Figure 1-5 illustrates how the Display Manager might reposition the window of a game that draws into a fixed size window.

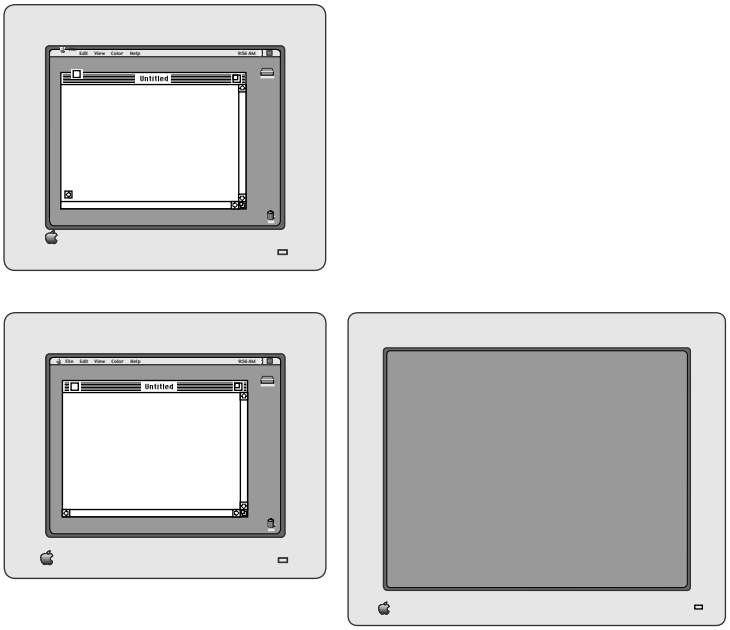
Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Figure 1-5 Default repositioning of a fixed-size window



When the user adds a display, the Display Manager does not move any windows to that display. For example, in Figure 1-6 either the user or the application must move the window on the main screen to the display added on the right. If your application works best on the largest available screen or on the one displaying the greatest number of colors, you may want your application to move its windows to the added display.

Figure 1-6 Default window positioning when the user adds a display



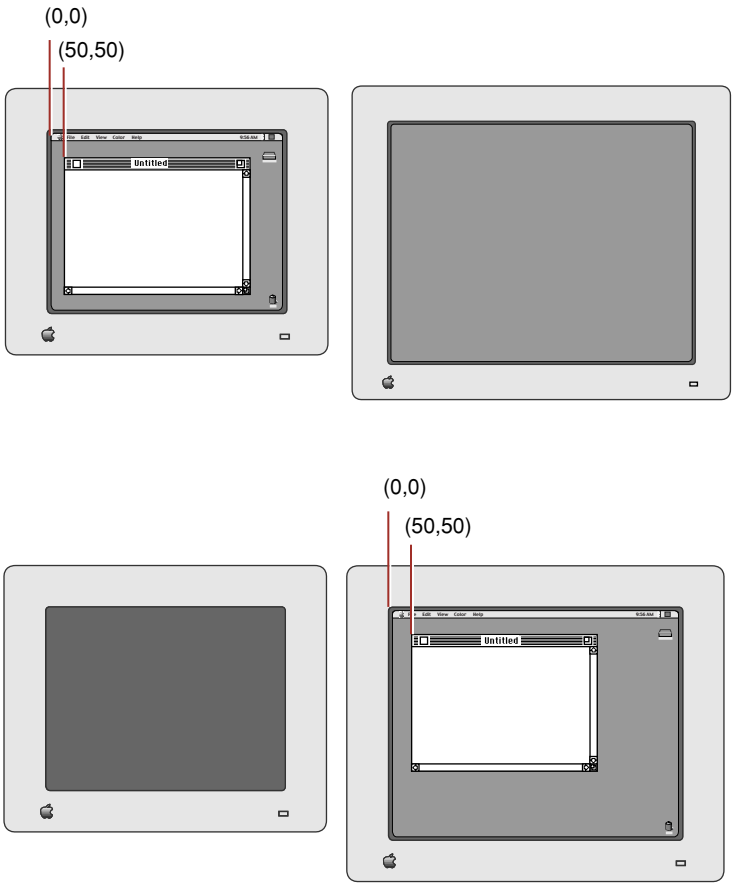
Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

When the User Moves the Menu Bar

On a computer with multiple screens, the user can use the Monitors control panel to change the main screen—that is, the one that contains the menu bar. Color QuickDraw maps the (0,0) origin point of the global coordinate system to the main screen's upper-left corner, and other screens are positioned adjacent to it. The Window Manager automatically maintains window positions according to this global coordinate system.

When the user changes the main screen, the upper-left corner of the new main screen becomes the (0,0) origin point of QuickDraw's global coordinate system, and all windows initially maintain their position relative to this new origin point. When a user moves the menu bar, the user sees the windows that previously appeared beneath the menu bar on one display moved to the display that now contains the menu bar.

Figure 1-7 Default window positioning when the user moves the menu bar



For example, the top of Figure 1-7 shows a window on the left display. The left display is the main screen, and the upper-left corner of the window is at coordinates (50,50) on the global coordinate system. At the bottom of the figure, the user moves the menu bar to the right display. The window retains its upper-left coordinates of (50,50), but because the (0,0) origin of the global coordinate system moved to the right screen, the window now appears in the right display.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

If the Display Manager finds that any windows move offscreen after the user moves the menu bar, the Display Manager repositions the windows as previously described—that is, it tries to move the title bar onto the closest screen or it tries to center the alert or modal dialog box on the closest screen.

Display Modes

The Display Manager allows users to choose from the various display modes available on their displays. A display mode is a combination of several interrelated capabilities that you can alter using the Display Manager to affect the display. You can characterize a display mode by

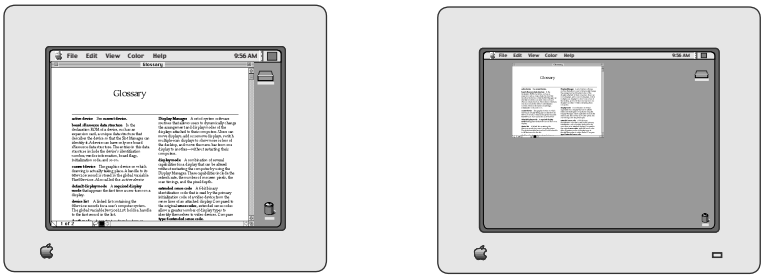
- the screen resolution, which determines the number of pixels that appear on the display screen
- the horizontal and vertical scan timings in use by the display
- the display's refresh rate

In addition to these capabilities, a display mode may also support multiple pixel depths, which determine the number of colors available on the display. You refer to the pixel depths available for a display mode as depth modes, and in various Display Manager data structures, depth modes are represented by constants or by their values from an enumerated list. A depth mode is also called a video mode.

Single-resolution grayscale or color monitors support multiple pixel depths only. Some multiple-resolution displays support display modes that change only the screen resolution and the pixel depth. For example, by choosing a lower screen resolution, a user with limited RAM can set the display to show a greater number of colors. Multiple-scan displays, however, are also capable of operating at multiple horizontal and vertical scan timings and at different refresh rates.

For example, a multiple-scan display might support display modes with screen resolutions of 640 by 480 pixels and 1152 by 870 pixels. The left side of Figure 1-8 illustrates a multiple-scan display operating at a screen resolution of 640 by 480 pixels. The right side of the figure illustrates the same display after it has been switched to a screen resolution of 1152 by 870 pixels.

Figure 1-8 Lower and higher screen resolutions on a multiple-scan monitor



When editing a bitmap image with a paint application, a user might wish to use the lower screen resolution, which, compared to the higher resolution, displays fewer pixels on the screen but displays them at a larger size. When using a spreadsheet application, however, the user might then want to switch to the higher resolution to increase the number of onscreen pixels and thereby view a greater number of cells in a spreadsheet.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

To change the screen resolution, the user opens the Monitors control panel and selects the display mode for that resolution. The Display Manager then sends the video device driver a control request to switch the display to the newly selected display mode.

All required display modes appear when the user opens the Monitors control panel. For a particular type of display (for example, a 21-inch video monitor), a required display mode is one that Apple requires the display to support. A multiple-scan display must support several required display modes, one of which is designated to be the default display mode. The default display mode appears the first time a user turns on a display. For example, the first time a user connects and starts a 21-inch video monitor, it should use a mode displaying 1152 by 870 pixels. However, a 21-inch multiple-scan display is also required to support display modes with resolutions of 640 by 480 pixels, 832 by 624 pixels, and 1024 by 768 pixels, which the user can select with the Monitors control panel.

Using Display Manager functions, your application can change the display mode and the pixel depth of any display for the user, but your application should do so only with the consent of the user. The Monitors control panel is the user interface for changing the pixel depth, color capabilities, and positions of video devices. Because the user can control the capabilities of the video devices, your application should be flexible. Although it may have a preferred pixel depth, your application should do its best to accommodate less than ideal conditions.

However, if your application must have a specific pixel depth, or a particular screen resolution, it can display a dialog box that offers the user a choice between changing to that depth or canceling display of the image. This dialog box saves the user the trouble of going to the Monitors control panel before returning to your application. Your application can then use Display Manager functions to change the display mode or pixel depth of a display.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Using the Display Manager

The previous chapter explains how the Display Manager automatically repositions windows if necessary to ensure that windows are accessible when the user changes the display environment. If the Display Manager moves windows in a manner inappropriate for your application, your application should reposition them instead. Applications that use only the standard window definition functions provided by the Window Manager generally do not need to use the Display Manager.

However, you may need or want your application to perform its own window positioning under various circumstances, such as when

- your application benefits by displaying windows and their contents on the display controlled by the video device with the greatest pixel depth
- your application benefits by displaying windows on the largest available display
- your application uses nonstandard window definition functions that draw windows lacking title bars; examples include fixed-sized windows without title bars (games often use such windows), tool palettes with drag regions on the left sides of their windows, and floating windows

When necessary, the Display Manager automatically repositions windows of type `dBoxProc` (that is, alert boxes and modal dialog boxes) so that the lower-left corners of the windows appear onscreen. This gives users access to the area with the OK and Cancel buttons.

In addition, your application should respond to Display Manager changes if your application relies on display information that it stores internally. For example, if your application caches display positions, `GDevice` structures for displays other than the main screen, or the value in the `screenBits.bounds` field of the `screenBits` global variable, this information may become invalid after the user changes the display configuration. Therefore, your application should update its internal values accordingly after a display configuration change.

To determine whether the Display Manager is available, use the `Gestalt` function with the `gestaltDisplayMgrAttr` selector. Test the bit field indicated by the `gestaltDisplayMgrPresent` constant in the `response` parameter. If the bit is set, then the Display Manager is present.

Presence of the Display Manager does not guarantee that a computer also supports video mirroring. To determine whether QuickDraw supports video mirroring on the user's computer system, use the `DMQDIsMirroringCapable` function.

Handling Events in Response to Display Manager Changes

Users indirectly inform the Display Manager of changes they wish to make to their display environment by using the Monitors control panel, or by attaching or removing additional displays. The Display Manager in turn sends an Apple event—the Display Notice event—to notify applications that the display environment has changed.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

After changing the display environment, the Display Manager also generates an update event to notify all current applications to update their windows.

Your application should always handle update events for its windows. However, your application needs to respond to the Display Notice event only if your application repositions its own windows, uses nonstandard windows, or must update any display information that it stores internally.

To receive the Display Notice event informing you of changes to the user's display configuration, you must either

- handle the Display Notice event as a high-level event in your application's normal event loop; or
- use the `DMRegisterExtendedNotifyProc` function to register a function that handles the Display Notice event as soon as the Display Manager issues it

If you write a utility—such as a control panel—that does not handle events through a normal event loop, or if you want your application to handle the Display Notice event as soon as it is issued instead of waiting for it to appear in the event queue, you should use the `DMRegisterExtendedNotifyProc` function.

Here is a summary of the Display Notice event (remember that you must use Apple Event Manager functions to obtain the information contained in Apple events such as this):

Display Notice—respond to display configuration changes

```
Event class
      kCoreEventClass
Event ID
      kAESystemConfigNotice
Required parameter
Keyword:
      kAEDisplayNotice
Descriptor type:
      AEDesc
Data:
```

A list of descriptor structures, each specified by the keyword `kDisplayID`. Each `kDisplayID` descriptor structure contains information about a video device attached to the user's system. Within each `kDisplayID` descriptor structure are a pair of additional keyword-specified descriptor structures: `keyDisplayOldConfig` and `keyDisplayNewConfig`. A description of the video device's previous state is saved in the `keyDisplayOldConfig` descriptor structure, and a description of the video device's current state is saved in the `keyDisplayNewConfig` descriptor structure.

Descriptions of these keyword-specified descriptor structures are in Table 2-1.

`Requested action`

Ensure that all windows appear to the user, and update any necessary display information that your application or utility stores internally.

Table 2-1 Keyword-specified descriptor structures.

Keyword	Value	Type	Description
`keyDeviceDepthMode`	`'dddm'`	`typeLongInteger`	The depth mode for the video device; that is, the value of the `gdMode` field in the `GDevice` structure for the device

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Keyword	Value	Type	Description
keyDeviceFlags	'dddf'	typeShortInteger	The attributes for the video device as maintained in the gdFlags field of the GDevice structure for the device
keyDeviceRect	'dddr'	typeQDRectangle	The boundary rectangle of the video device; that is, the value of the gdRect field in the GDevice structure for the device
keyDisplayDevice	'dmdd'	typeLongInteger	A handle to the GDevice structure for the video device
keyDisplayID	'dmid'	typeLongInteger	The display ID for the video device
keyDisplayMode	'dmdm'	typeLongInteger	The sResource number from the video device for this display mode
keyDMConfigVersion	'dmcv'	typeLongInteger	The version number for this Display Notice event
keyPixMapAlignment	'dppa'	typeLongInteger	Reserved for future use
keyPixMapCmpCount	'dpcc'	typeShortInteger	The number of components used to represent a color for a pixel; that is, the value of the cmpCount field in the PixMap structure for the GDevice structure for the device
keyPixMapCmpSize	'dpcs'	typeShortInteger	The size in bits of each component for a pixel; that is, the value of the cmpSize field in the PixMap structure for the GDevice structure for the device
keyPixMapColor-TableSeed	'dpct'	typeLongInteger	The value of the ctSeed field of the ColorTable structure for the PixMap structure for the GDevice structure for the video device
keyPixMapHResolution	'dphr'	typeFixed	he horizontal resolution of the pixel image in the PixMap structure for the GDevice structure for the video device
keyPixMapPixelSize	'dpps'	typeShortInteger	Pixel depth for the device; that is, the value of the pixelSize field in the PixMap structure for the GDevice structure for the video device
keyPixMapPixelType	'dppt'	typeShortInteger	The storage format for the pixel image on the device; that is, the value of the pixelType field in the PixMap structure for the GDevice structure for the video device

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Keyword	Value	Type	Description
keyPixMapRect	'dpdr'	typeQDRectangle	The boundary rectangle into which QuickDraw can draw; that is, the bounds field in the PixMap structure for the GDevice structure for the video device
keyPixMapReserved	'dppr'	typeLongInteger	Reserved for future use
keyPixMapResReserved	'dprr'	typeLongInteger	Reserved for future use
keyPixMapVResolution	'dpvr'	typeFixed	The vertical resolution of the pixel image in the PixMap structure for the GDevice structure for the video device

Handling the Display Notice Event as a High-Level Event

To handle the Display Notice event as a high-level event like any other Apple event, you need to

- set the isHighLevelEventAware bit in your application's 'SIZE' resource to indicate that your application supports high-level events (in which case your application must also support the four required Apple events)
- include code to handle high-level events in your main event loop (as illustrated in Listing 2-1 (page 22))
- write a function that handles the Display Notice event (as illustrated in Listing 2-2 (page 23))
- use the AEInstallEventHandler function to install the entry for handling the Display Notice event in your application's Apple event dispatch table

If you want your application to handle all window positioning itself (that is, if you do not want the Display Manager to automatically move any of your windows), you should also set the isDisplayManagerAware bit in the 'SIZE' resource.

Listing 2-1 Handling Apple events in the event loop

```
void MyDoEvent(EventRecord *event)
{
    short           part, err;
    WindowPtr       window;
    char            key;
    switch ( event->what ) {
        /* here, handle null, mouse down, key down, update, and
            other necessary events */
        case kHighLevelEvent:
DoHighLevelEvent(event);
break;
    }
}
void DoHighLevelEvent(EventRecord *event)
{
    OSErr   myErr;
        /* handling only Apple-event types of high-level events */
```


Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

```
        myErr = AEProcessAppleEvent(event);
}
```

Your application must use the `AEInstallEventHandler` function to add an entry to your application's Apple event dispatch table. This entry is the function that responds to the Display Notice event. For example, the following code fragment illustrates how to use `AEInstallEventHandler` to install an application-defined function called `DoAEDisplayUpdate`.

```
err = AEInstallEventHandler (kCoreEventClass,
                             kAESystemConfigNotice,
                             (ProcPtr)DoAEDisplayUpdate, 0, false);
```

Listing 2-2 shows an application-defined function called `DoAEDisplayUpdate` that uses Apple Event Manager functions to obtain information about the various video devices reported by the Display Notice event. The function `DoAEDisplayUpdate` uses this information to update its internal data structures for its windows and then calls another application-defined function that ensures that its windows are displayed optimally in the new configuration environment.

Listing 2-2 Responding to the Display Notice event

```
pascal OSErr DoAEDisplayUpdate
(AppleEvent theAE,AppleEvent reply,long ref) {
    #pragma unused(theAE,reply,ref)
    AEDescList      DisplayList;
    AEDescList      DisplayID;
    AERecord        OldConfig,NewConfig;
    AEKeyword       tempWord;
    AEDesc          returnType;
    OSErr           myErr;
    long            result;
    long            count;
    Rect            oldRect, newRect;
    Size            actualSizeUnused;
/* get a list of the displays from the Display Notice event */

    myErr =
    AEGetParamDesc(&theAE,kAEDisplayNotice,typeWildCard,&DisplayList);

/* count the elements in the list */
    myErr = AECountItems(&DisplayList,&count);
    while (count >0)        /* decode the Display Notice event */
{
        myErr = AEGetNthDesc(&DisplayList, count, typeWildCard,
            &tempWord, &DisplayID);
        myErr = AEGetNthDesc(&DisplayID, 1, typeWildCard, &tempWord,
            &OldConfig);
        myErr = AEGetKeyPtr(&OldConfig, keyDeviceRect, typeWildCard,
            &returnType, &oldRect, 8, actualSizeUnused);
        myErr = AEGetNthDesc(&DisplayID, 2, typeWildCard, &tempWord,
            &NewConfig);
        myErr = AEGetKeyPtr(&NewConfig, keyDeviceRect, typeWildCard,
            &returnType, &newRect, 8, actualSizeUnused);
/* update internal info about the gdRects for the devices */
        MyUpdateWindowStructures(oldRect, newRect);
        count--;
}
/* move and resize windows as necessary*/
```


Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

```
    MyDisplayWindows();
    return (noErr);
}
```

Handling the Display Notice Event Outside of an Event Loop

You may want your application to handle the Display Notice event as soon as it is issued instead of waiting for it to appear in the event queue. You can use the `DMRegisterExtendedNotifyProc` function to register a function to which the Display Manager directly sends the Display Notice event. By using `DMRegisterExtendedNotifyProc`, and by not setting the `isHighLevelEventAware` bit in the `'SIZE'` resource, you cause the Display Manager to send a Display Notice event directly to your handling function; your application or utility then receives no high-level Display Notice event.

To remove your Display Notice event-handling function, use the `DMRemoveExtendedNotifyProc` function.

Managing Windows In Response to the Display Notice Event

Using the Monitors control panel, the user can switch displays to use a different display mode and to change the display configurations. When your application receives the Display Notice event as described in the previous section, your application must determine whether it needs to reposition and perhaps resize its windows.

Listing 2-3 illustrates how an application can check whether its nonstandard window appears onscreen after Display Manager configuration changes have occurred. In this example, the application has a window with a title bar on its left side, as shown in the tool palette illustrated in Figure 1-4 (page 13). After receiving the Display Notice event as shown in Listing 2-2 (page 23), the application calls its `MyDisplayWindows` function, which in turn calls its `MyMakeToolWindowVisible` function. If `MyMakeToolWindowVisible` determines that the nonstandard title bar does not appear on any displays (in which case the user cannot move the window), `MyMakeToolWindowVisible` moves the entire window to the main screen where the user has access to the window.

Listing 2-3 Ensuring that a nonstandard window appears onscreen

```
static pascal OSErr MyMakeToolWindowVisible (WindowPeek window) {
    if (window->windowKind == applicationFloatKind) {
        Rect        checkRect;
        Rect        mainRect;
        GDHandle    maxAreaDevice;
        short       theWVariant;
        Rect        windowRect;
        theWVariant = GetWVariant(&window->port);
        MyGetWindowGlobalRect(window, &windowRect);
        /*get rectangle of window, in global coordinates, here */
        if (0 != (kVerBarFW & theWVariant))
        /* check if this is the window with a vertical title bar */
        {
        /* following line gets the rectangle of the title bar */
        SetRect(&checkRect, windowRect.left-kMyVertTitleWidth+kMyMinVisX,
                                    windowRect.top+kkMyMinVisV,
                                    windowRect.left-1-kMyMinVisX,
```


Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

```
                                windowRect.bottom-kMyMinVisV);
        /* following line calls an application-defined function that
        determines which screen contains the largest amount of the title
        bar */
        maxAreaDevice = MyFindMaxCoverageDevice(&checkRect);
        if (nil == maxAreaDevice)
        /* if the title bar doesn't appear on any screen, move window to
        the main screen */
        {   mainRect = (*GetMainDevice()) -> gdRect;
            MoveWindow(&Window->port, mainRect.left+10+kMyVertTitleWidth,
            mainRect.bottom-10-(windowRect.bottom-windowRec.top, FALSE);
        } }
        MyKeepWindowOnscreen(window, nil);
        /* handle other nonstandard window variants here */
        }
        return noErr;
}
```

Your application may find it useful to resize a window after moving it, or to optimize the color for its newly configured video device. You can use Display Manager functions to determine the characteristics of video devices, as explained in the next section.

Determining the Characteristics of the Video Devices

To determine the characteristics of available video devices, your application can use the `DMGetFirstScreenDevice` function to obtain a handle to the `GDevice` structure for the first video device in the device list. The `DMGetFirstScreenDevice` function is similar to the QuickDraw function `GetDeviceList`, except that when returning `GDevice` structures, `GetDeviceList` does not distinguish between the `GDevice` structures for video devices and the `GDevice` structures associated with no video devices. (For example, if system software uses the function `DMDisableDisplay` to disable the last remaining device in the device list, then `DMDisableDisplay` inserts into the device list a `GDevice` structure that is not associated with any video device. The `DMGetFirstScreenDevice` function will not return this `GDevice` structure, but `GetDeviceList` might.)

After using the `DMGetFirstScreenDevice` function to obtain a handle to the first `GDevice` structure for a display in the device list, your application can use the `DMGetNextScreenDevice` function to loop through all of the video devices in the device list. The `DMGetNextScreenDevice` function is similar to the QuickDraw function `GetNextDevice`, except that when returning `GDevice` structures, `GetNextDevice` does not distinguish between the `GDevice` structures for video devices and the `GDevice` structures associated with no video devices.

Another important difference between these two Display Manager functions (`DMGetFirstScreenDevice` and `DMGetNextScreenDevice`) and their related QuickDraw functions (`GetDeviceList` and `GetNextDevice`) is that with both Display Manager functions, your application can specify that the Display Manager return handles only to active video devices. (An active device is a video device whose display area is included in the user's desktop; the display area of an inactive device does not appear on the user's desktop.)

To get a handle to the `GDevice` structure for a video device that mirrors another, your application can use the `DMGetNextMirroredDevice` function.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Your application can pass the `GDevice` handle returned for any of these video devices to a QuickDraw function like `TestDeviceAttribute` or `HasDepth` to determine various characteristics of the video device, or your application can examine the `gdRect` field of the `GDevice` structure to determine the dimensions of the screen it represents.

Macintosh system software uses the `DMCheckDisplayMode` function to determine whether a video device supports a particular display mode and pixel depth. Typically, your application does not need to know whether a display mode is supported, but only whether a specific pixel depth is supported, in which case your application can use the Color QuickDraw function `HasDepth`.

To determine whether QuickDraw supports video mirroring on the user's computer system, your application can use the `DMQDIsMirroringCapable` function. Your application can use the `DMCanMirrorNow` function to determine whether video mirroring can activate. And to determine whether the user's computer system currently uses video mirroring, your application can use the `DMIsMirroringOn` function.

Finally, your application can use the `DMGetDisplayIDByGDevice` function to determine the display ID for a video device. A display ID is a long integer used by the Display Manager to uniquely identify a video device. Associating a display by its display ID is helpful when using functions such as `DMRemoveDisplay` that could change the `GDevice` structure associated with a video device. You can first determine the display ID for a device by using the `DMGetDisplayIDByGDevice` function. To later retrieve that device's `GDevice` structure after calling various Display Manager functions, your application can use the `DMGetGDeviceByDisplayID` function. Display IDs are not guaranteed to be persistent across reboots or sleep.

Setting Configurations and Display Modes for Video Devices

The Monitors control panel is the user interface for changing the pixel depth, color capabilities, and positions of video devices. Because the user can control the capabilities of the video devices, your application should be flexible. For instance, although your application may have a preferred pixel depth, it should do its best to accommodate less than ideal conditions.

Your application can use Display Manager functions to change the display mode and display configuration of the user's video devices, but your application should do so only with the consent of the user.

If your application must have a specific pixel depth, for example, it can display a dialog box that offers the user a choice between changing to that depth or canceling display of the image. This dialog box saves the user the trouble of going to the control panel before returning to your application. If it is absolutely necessary for your application to draw on a video device of a specific pixel depth, your application can then use either the `SetDepth` function or the `DMSetDisplayMode` function.

With the possible exception of the `DMSetDisplayMode` function and the `DMMirrorDevices` and `DMUnmirrorDevice` functions, applications should not need to use any of the Display Manager functions that change the user's display configuration. However, they are described for completeness, in case you find a compelling need for your application to change the user's display configuration. If your application must use multiple Display Manager calls that configure the user's displays, your application should first use the `DMBeginConfigureDisplays` function to postpone Display Manager configuration checking, the rebuilding of desktop regions, and Apple event notification of Display Manager changes. When finished configuring the user's displays, use the `DMEndConfigureDisplays` function. Using `DMBeginConfigureDisplays` and `DMEndConfigureDisplays` allows your application to wait until it has made all display changes before managing its windows in response to a single Display Notice event. It is important to pass the `displayState` variable obtained in `DMBeginConfigureDisplays` to the `DMEndConfigureDisplays` function.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Optimizing Display Modes and Window Arrangement With the Display Manager*.

Date	Notes
2007-05-03	Moved to Legacy Documents area; no content update.
2003-05-01	Fixed a broken link to Figure 2-4.
2003-02-01	Structured document.
2000-04-01	Last update of this document.

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.

Document Revision History

Legacy Document | 2007-05-03 | © 2007 Apple Inc. All Rights Reserved.