Developer Note

Display Device Driver Guide

Device support for the Display Manager

Developer Note

Developer Press
© Apple Computer, Inc. 1994

Contents

Figures

About This Developer Note

This document describes how the Display Manager communicates with display devices. It is written primarily for experienced Macintosh hardware and software developers who want to create video device drivers that are compatible with the Macintosh computer. If you are unfamiliar with Macintosh computers or would simply like more technical information, you may want to read the related technical manuals listed in "Supplementary Documents."

Supplementary Documents

To supplement the information in this document, you might wish to obtain related documentation such as *Guide to the Macintosh Family Hardware,* second edition; *Designing Cards and Drivers for the Macintosh Family,* third edition; and *Inside Macintosh*. These documents are available through APDA.

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA
America Online	APDAorder
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

Conventions and Abbreviations

This developer note uses typographical conventions and abbreviations that are standard in Apple publications.

Typographical Conventions

Computer-language text—any text that is literally the same as it appears in computer input or output—appears in `Courier` font.

Hexadecimal numbers are preceded by a dollar sign ($). For example, the hexadecimal equivalent of decimal 16 is written as $10.

C H A P T E R 1

Device Support for the Display Manager

This chapter explains how the Display Manager communicates with a video device driver to change display modes for displays that support multiple screen resolutions. (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 **display mode** is a combination of several interrelated capabilities that can be altered using the Display Manager to affect the display. A display mode is characterized by

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

Some multiple-resolution displays, such as some flat-panel displays on PowerBook computers, support display modes that change only the screen resolution and the pixel depth. For example, by choosing a lower screen resolution, a color PowerBook 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.

If you create video devices—such as plug-in video cards or built-in video interfaces—or write video device drivers, read this chapter to learn how the Display Manager communicates with your devices.

To use this chapter, you must be familiar with building devices for Macintosh computers as explained in *Designing Cards and Drivers for the Macintosh Family,* third edition. You must also be familiar with the Device Manager and the Slot Manager, both of which are described in *Inside Macintosh: Devices*. You should also be familiar with the way QuickDraw and QuickDraw GX prepare images for video devices, as explained in *Inside Macintosh: Imaging With QuickDraw* and in the *Inside Macintosh: QuickDraw GX* suite of books.

About the Display Manager

The **Display Manager** allows users to dynamically change the arrangement and display modes of the displays attached to their computers. For example, users can move their displays, add or remove displays, switch displays to show more or fewer pixels, and move the menu bar from one display to another—all without restarting their computers. When the user changes the display environment, the Display Manager notifies all current applications about these changes. Applications can then reposition their windows according to their own criteria; otherwise, the Display Manager repositions all windows so that the user can find them in the new display environment.

From the perspective of your video device, the Display Manager allows users to choose from the various display modes available on their displays. For example, a multiple-scan display might support display modes with screen resolutions of 640 by 480 pixels and

1024 by 768 pixels. When editing a bitmap image with a paint application, a user might wish to use the lower screen resolution, which—compared with 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.

To change to the higher resolution, the user opens the Monitors control panel and selects the display mode for that resolution. The Display Manager then sends your 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.

Note
If a device driver does not support the type 6 extended sense codes as described on page 1-5, the user of a multiple-scan monitor generally sees only a default display mode of 640 by 480 pixels. ◆

In addition to its required display modes, a display may also support additional display modes, which are called **optional display modes.** Optional display modes are shown when the user holds the Option key and clicks the Options button in the Monitors control panel. After the user selects an optional display mode, the Display Manager asks your device driver to switch to that display mode, and the Monitors control panel displays a confirmation dialog box asking whether the mode works. If the mode does not work, the user may not be able to see the dialog box, in which case pressing Command-period, Escape, Return, or Enter all cause the Display Manager to revert to the previous display mode. If within 6 seconds the user does not click a button in the dialog box or press any of these keys, the Display Manager reverts to the previous display mode.

Table 1-1 shows the screen resolutions and scan timings for the default, required, and optional display modes of Apple displays that can be altered using the Display Manager.

Table 1-1 Characteristics of the display modes for Apple displays

Display type	Screen resolution and scan timing of default mode	Screen resolutions and scan timings of other required modes	Screen resolutions and scan timings of optional modes
Multiple-scan type 1	640 × 480 pixels at 67 Hz (e.g., 13-inch display)	832 × 624 pixels at 75 Hz	512 × 384 pixels at 60 Hz
Multiple-scan type 2	832 × 624 pixels at 75 Hz (e.g., 16-inch display)	640 × 480 pixels at 67 Hz 1024x768 pixels at 75 Hz	
Multiple-scan type 3	1152 × 870 pixels at 75 Hz (e.g., 21-inch display)	640 × 480 pixels at 67 Hz 832 × 624 pixels at 75 Hz 1024 × 768 pixels at 75 Hz	
VGA	640 × 480 pixels at 60 Hz		800 × 600 pixels at 56 Hz (i.e., SVGA) 1024 × 768 pixels at 60 Hz (i.e., VESA 1K-60Hz) 1024 × 768 pixels at 70 Hz (i.e., VESA 1K-70Hz)
Full-page portrait	640 × 870 pixels at 75 Hz	640 × 818 pixels at 75 Hz	
Flat-panel display	640 × 480 pixels	640 × 400 pixels	

If the user changes the display mode for the display attached to your video device before shutting down the computer, the Monitors control panel makes the `SetDefaultMode` call to your driver, which should store that mode in parameter RAM and use it as the startup mode (that is, as a new default display mode) when the user next restarts the computer.

In determining what type of display is attached to your video device, device drivers usually examine the display's sense code. As described in "Supporting Type 6 Extended Sense Codes for Multiple-Scan Displays" beginning on page 1-5, a **sense code** is an identification code that is read by the primary initialization code for a video device from the sense lines of an attached display. Your device driver can associate each sense code with a default display mode and a set of required and optional display modes for the display.

Supporting the Display Manager With Your Device Driver

To support the Display Manager, your video device must

- recognize type 6 extended sense codes, either in the `PrimaryInit` code in your declaration ROM or in a patch to your device driver
- allow all functional sResources supported by the attached display to remain in the slot resource table; that is, your `PrimaryInit` code and your driver should enable one functional sResource and disable the other sResources supported by the display
- support two status requests and one control request in your device driver
- report timing information for the attached display either by including timing information for each functional sResource in your declaration ROM or by supporting the `cscGetModeTiming` status request in your device driver

The Display Manager requires System 7.1 and Color QuickDraw. If the Display Manager is not present but your device driver supports it, the user can still select different display modes on a Color QuickDraw system by using the Monitors control panel; however, the user must restart the computer for the new display mode to take effect.

Supporting Type 6 Extended Sense Codes for Multiple-Scan Displays

Listed here are the extended sense codes for multiple-scan displays that your device must recognize. (If you create your own displays, they must communicate these extended sense codes to their video devices.)

```
extendedMSB1   Equ   $03   ; type 6, 12 or 13-inch multiple-scan
                           ;  display
extendedMSB2   Equ   $0B   ; type 6, 16 or 17-inch multiple-scan
                           ;  display
extendedMSB3   Equ   $23;  ; type 6, 19 or 21-inch multiple-scan
                           ;  display
extendedHR     Equ   $2B   ; 12 or 13-inch single-scan display
                           ;  without an extended sense code
```

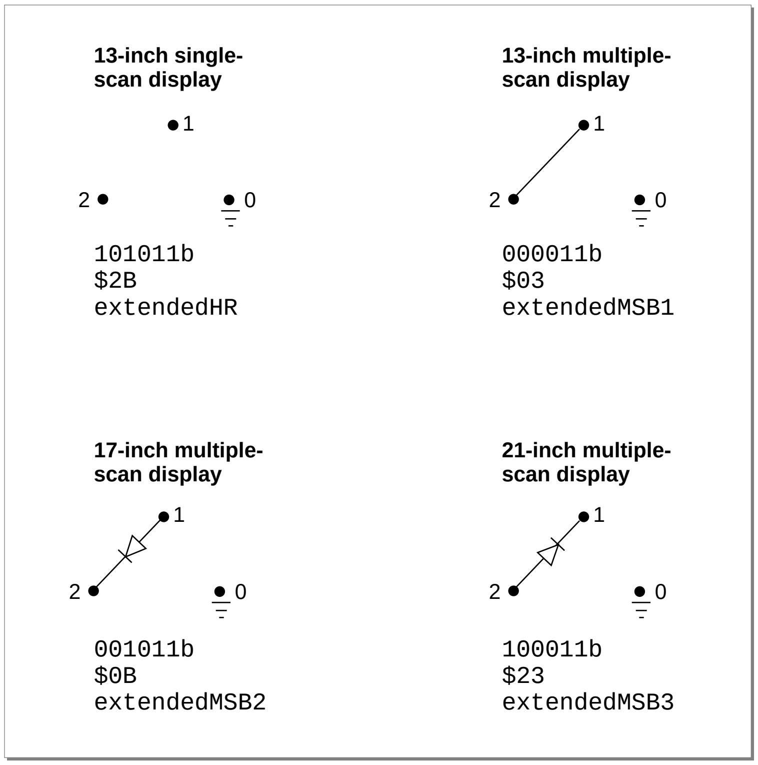
Extended sense codes are 6-bit binary numbers that define how displays' sense lines should respond in order for displays to be recognized by your video device. Before the introduction of these extended sense codes, a video device determined the display type by reading its three sense lines and simply comparing the signal value of each to ground. By convention, the sense lines were identified as 0, 1, and 2. Given the three lines and the two different states, on or off, there were a total of eight possible sense codes. Your device reads a sense line connected to ground as a binary 0 and an ungrounded sense line as a binary 1.

Seven combinations of the sense line states were assigned to early displays. To support additional display types, extended sense codes have been added. Multiple-scan displays indicate that they support **type 6 extended sense codes** by leaving sense lines 1 and 2 ungrounded and sense line 0 grounded. (The eighth sense line combination, where none of the three sense lines is connected to ground, signals the use of type 7 extended sense codes, which are not discussed in this chapter.)

When your video device reads the states of the display's sense lines and finds that only line 0 is grounded, your device should interpret the states of the sense lines as a type 6 extended sense code. If your device supports regular but not extended sense codes, it will interpret this sense line combination as the sense code for a 13-inch single-scan display. In this case, the user will be able to use the display in only one mode: that which supports a screen resolution of 640 by 480 pixels.

Multiple-scan displays communicate a type 6 sense code and its extended sense code by connecting sense lines 1 and 2 with either a wire or a diode. As shown in Figure 1-1, a straight wire between these sense lines signals a 13-inch multiple-resolution display, a diode connecting line 1 to line 2 signals a 17-inch multiple-resolution display, and a diode connected line 2 to line 1 signals a 21-inch multiple-resolution display. If there are no connections between these sense lines, then the display is a 13-inch display that does not support multiple scan timings.

Figure 1-1 Sense line connections for type 6 sense codes


13-inch single-scan display
1
2
0
101011b
$2B
extendedHR
13-inch multiple-scan display
1
2
0
000011b
$03
extendedMSB1
17-inch multiple-scan display
1
2
0
001011b
$0B
extendedMSB2
21-inch multiple-scan display
1
2
0
100011b
$23
extendedMSB3

To determine an extended sense code, your device pulls down each sense line while reading the other two sense lines. When reading a line, your device compares it with the down line instead of comparing it with ground. If a line has the same state as the down line, your device assigns it a binary value of 0; otherwise, your device assigns it a binary value of 1. Starting with the third sense line, your device pulls down each line and compares it with the others as follows:

1. Of the three sense lines (0, 1, and 2), start by pulling down line 2; compare it with line 0 and then with line 1 to derive the first 2 bits of the sense code. For a 13-inch multiple-scan display, for example, you will derive the binary value 00.
2. After pulling down line 2, pull down line 1; compare it with line 0 and then with line 2 to derive the next 2 bits. For a 13-inch multiple-scan display, for example, you will derive the binary value 00.
3. Use the binary value 11 for the last 2 bits, which are identical for all type 6 sense codes. For a 13-inch multiple-scan display, then, you derive a sense code value of 00 00 11b (where *b* denotes binary), which equates to hexadecimal value $03 and the constant `extendedMSB1`.

Enabling and Disabling Functional sResources

Video devices typically have functional sResources to support a variety of display modes. A **functional sResource,** stored in a device's declaration ROM, describes a specific function of the device—for example, controlling a display at a certain display mode. Your device should use sResources to describe display modes for its attached display. The `PrimaryInit` code for your video device driver should not delete from the slot resource table any of the functional sResources supported by the display attached to your video device. Instead, your device driver should enable one functional sResource, which Color QuickDraw uses to build the `GDevice` record for your device, and your driver should disable the other sResources.

When the Display Manager sends your device driver the `cscSwitchMode` control call, your control routine should use the Slot Manager function `SetSRsrcState` once to disable the functional sResource describing the existing display mode and then again to enable the functional sResource specified by `cscSwitchMode`, as illustrated in Listing 1-1.

Listing 1-1 Disabling and enabling sResources for display modes

```
;code fragment illustrating how to disable and enable functional
; sResources while switching display modes

WITH     VSCVidParams,SpBlock
MOVEM.L  D0-D1/A0-A1,-(SP)        ;save work registers--D1
                                  ; contains the spID field for
                                  ; the sResource to enable, and
                                  ; D1.b contains the sResource ID
```

```
                                ; that is about to swapped in;
                                ; A1 points to a device control
                                ; block
MOVE.B   dCtlSlotID(A1),D0      ;save the sResource to disable
MOVE.B   D1,dCtlSlotId(A1)      ;save the sResource to enable,
                                ; update dCtlSlotId
         ;if necessary, update dCtlDevBase and dCtlExtDev too
SUBA.W   #spBlockSize,SP        ;allocate slot parameter block
MOVE.L   SP,A0                  ;point to block with A0
MOVE.B   dCtlSlot(A1),spSlot(A0) ;set up the right slot number
CLR.B    spExtDev(A0)           ;it is necessary to clear this
MOVE.B   D0,spID(A0)            ;write out spID of sResource to
                                ; disable
_SRsrcInfo                      ;(update SpBlock for below)
MOVE.L   #1,spParamData(A0)     ;the sResource to disable
_SetSRsrcState                  ;disable the current sResource
MOVE.B   dCtlSlotID(A1),spID(A0) ;write out spID of sResource to
                                ; enable
MOVE.L   #0,spParamData(A0)     ;the sResource to enable
_SetSRsrcState                  ;enable the specified sResource
_SUpdateSRT                     ;update slot resource table
```

As shown in Listing 1-1, your control routine should use the Slot Manager function `SUpdateSRT` after enabling the sResource for the new display mode; using the `SUpdateSRT` function updates the device driver reference number for this sResource in the slot resource table.

Your control routine must update the `dCtlSlotID` field (and—if your driver uses it—the `dCtlExtDev` field) of your driver's device control entry. If disabling and enabling sResources moves the pixel map's base address in the `GDevice` record for your video device, your control routine must also update the `dCtlDevBase` field of the device control entry.

Responding to Control and Status Requests

Using control and status requests, the Display Manager informs your device driver about user changes to the display environment and asks your device driver about its attached display.

When the user opens the Monitors control panel and changes the display mode for the display attached to your video device, the Display Manager sends your device driver the `cscSwitchMode` control request, described on page 1-19. To support the Display Manager, your driver must provide a control routine to respond to this request.

The Display Manager uses the `cscGetCurMode` status request, described on page 1-21, to determine whether your video device driver supports the Display Manager. If your driver returns the `statusErr` result code to this status request, system software assumes that your driver cannot switch display modes without making the user restart the computer. The Display Manager and other parts of system software also make a `cscGetCurMode` status request to save the current display mode. For example, the Monitors control panel uses `cscGetCurMode` to save the current display mode in case it becomes necessary to undo a change made by the user.

The Display Manager uses the `cscGetConnection` status request, described on page 1-22, to gather information about the display capabilities of the display attached to your video device. For example, when the Display Manager sends your device driver the `cscGetConnection` status request, your video device driver reports whether the display modes represented by the sResources for the attached display are optional or required.

If your video device's declaration ROM does not include a timing directory describing the scan timings for the display attached to your device, the Display Manager uses the `cscGetModeTiming` status request to gather this information. The next section describes how your device can report scan timings to the Display Manager.

Reporting Scan Timings

Your video device must report timing information for its attached display. If you are creating a new ROM for your device, you might want to include scan timings in your declaration ROM; otherwise, your driver must return a display's scan timings by supporting the `cscGetModeTiming` status request.

To gather scan timing information, the Display Manager makes a `cscGetModeTiming` status request to your driver. If your driver returns the `statusErr` result code, the Display Manager uses the Slot Manager to examine the declaration ROM on your video device. The Display Manager looks in your declaration ROM for a timing directory (`sVidParmDir` = 123) in the **board sResource,** which is the data structure that identifies your device to the Slot Manager. The timing directory should contain the scan timings for all the functional sResources in the declaration ROM (just as the video mode name directory should contain the names of all the functional sResources). For each functional sResource in the declaration ROM, the timing directory should contain an entry with a value represented by one of the following constants:

```
enum {
   timingUnknown     = 0,     /* unknown timing */
   timingApple12     = 130,   /* 512x384 (60 Hz) 12" RGB */
   timingApple12x    = 135,   /* 560x384 (60 Hz) */
   timingApple13     = 140,   /* 640x480 (67 Hz) 13" RGB */
   timingApple13x    = 145,   /* 640x400 (67 Hz) */
   timingAppleVGA    = 150,   /* 640x480 (60 Hz) VGA */
   timingApple15     = 160,   /* 640x870 (75 Hz) full page display */
   timingApple15x    = 165,   /* 640x818 (75 Hz) full page display 818 */
   timingApple16     = 170,   /* 832x624 (75 Hz) 16" RGB */
   timingAppleSVGA   = 180,   /* 800x600 (56 Hz) SVGA */
   timingApple1Ka    = 190,   /* 1024x768 (60 Hz) VESA 1K-60Hz */
```

```
    timingApple1Kb    = 200,   /* 1024x768 (70 Hz) VESA 1K-70Hz */
    timingApple19     = 210,   /* 1024x768 (75 Hz) Apple 19" RGB */
    timingApple21     = 220,   /* 1152x870 (75 Hz) Apple 21" RGB */
    timingAppleNTSC_ST
                      = 230,   /* 512x384 (60 Hz, interlaced, nonconvolved) */
    timingAppleNTSC_FF
                      = 232,   /* 640x480 (60 Hz, interlaced, nonconvolved) */
    timingAppleNTSC_STconv
                      = 234,   /* 512x384 (60 Hz, interlaced, convolved) */
    timingAppleNTSC_FFconv
                      = 236,   /* 640x480 (60 Hz, interlaced, convolved) */
    timingApplePAL_ST
                      = 238,   /* 640x480 (50 Hz, interlaced, nonconvolved) */
    timingApplePAL_FF
                      = 240,   /* 768x576 (50 Hz, interlaced, nonconvolved) */
    timingApplePAL_STconv
                      = 242,   /* 640x480 (50 Hz, interlaced, nonconvolved) */
    timingApplePAL_FFconv
                      = 244    /* 768x576 (50 Hz, interlaced, nonconvolved) */
};
```

The Display Manager uses these constants to look up display modes in an internal table of modes supported by various displays.

If your board sResource doesn't include a timing directory, your device driver must supply a status routine that processes and responds to the `cscGetModeTiming` status request. The `cscGetModeTiming` status request is described in detail on page 1-24.

Device Support for the Display Manager Reference

This reference section describes the records and the control and status requests used by the Display Manager. "Data Structures" shows the C language data structures for the `VDSwitchInfoRec`, `VDDisplayConnectInfoRec`, and `VDTimingInfoRec` records. "Control Requests" describes the `cscSwitchMode` control request. "Status Requests" describes the `cscGetCurMode`, `cscGetConnection`, and `cscGetModeTiming` status requests.

Data Structures

This section shows the C language data structures for the `VDSwitchInfoRec`, `VDDisplayConnectInfoRec`, and `VDTimingInfoRec` records. The Display Manager sends the `cscSwitchMode` control request to your video device driver to switch to the display mode specified by a `VDSwitchInfoRec` record. Your device driver also uses a

`VDSwitchInfoRec` record to return information about the current display mode in response to a `cscGetCurMode` status request from the Display Manager. Using a `VDDisplayConnectInfoRec` record, your device driver returns information about the capabilities of your attached display in response to a `cscGetConnection` status request. In response to a `cscGetCurMode` status request, your device driver uses a `VDTimingInfoRec` record to return information about the scan timings available on the display.

VDSwitchInfoRec

The Display Manager uses the `cscSwitchMode` control request, described on page 1-19, to switch your video device to the display mode specified by a `VDSwitchInfoRec` record. Your video device driver is responsible for setting your video device and your device control entry information to match the display mode specified in this record.

Your device driver also uses a `VDSwitchInfoRec` record to return information about the current display mode in response to a `cscGetCurMode` status request (described on page 1-21) from the Display Manager.

```
struct VDSwitchInfoRec {
   unsigned short csMode;     /* depth mode (also called video
                                 mode) */
   unsigned long  csData;     /* functional sResource for display
                                 mode */
   unsigned short csPage;     /* number for a video page */
   Ptr            csBaseAddr; /* pointer to the base address for
                                 the video page specified in the
                                 csPage field */
   unsigned long  csReserved; /* reserved for future expansion;
                                 currently set to 0 by the
                                 Display Manager */
};
```

Field descriptions

`csMode` The **depth mode** (also called *video mode*), which describes the pixel depth and is specified by a constant or its value from the following enumerated list:

```
enum {
firstVidMode      = 128,   /* first depth mode,
                              representing lowest
                              supported pixel
                              depth */
secondVidMode     = 129,   /* second depth mode,
                              representing next
```

```
                                                highest depth */
               thirdVidMode      = 130,   /* third depth mode,
                                             representing next
                                             highest depth */
               fourthVidMode     = 131,   /* fourth depth mode,
                                             representing next
                                             highest depth */
               fifthVidMode      = 132,   /* fifth depth mode,
                                             representing next
                                             highest depth */
               sixthVidMode      = 133,   /* sixth depth mode,
                                             representing next
                                             highest depth */
};
```

In response to the `cscSwitchMode` control request, your control routine should switch your video device to use the pixel depth specified by this mode. In response to the `cscGetCurMode` status request, your status routine should use the `csMode` field to return the current depth mode.

A depth mode specified by the `firstVidMode` constant represents the lowest supported pixel depth—typically, 1 bit per pixel. A depth mode specified by the `secondVidMode` constant represents the next highest supported pixel depth—often, but not necessarily, 2 bits per pixel. If your video device supports 4 bits per pixel instead of 2 as its next highest pixel depth, then its driver uses the `secondVidMode` constant to represent 4 bits per pixel. In this manner, the remaining constants signifying depth modes specify an ordered set of increasingly higher pixel depths.

`csData`	The number used by your video device to identify the functional sResource describing a particular display mode. In response to the `cscSwitchMode` control request, your control routine should enable the functional sResource specified in this field after disabling the current sResource. In response to the `cscGetCurMode` status request, your status routine should use this field to return the number for the sResource that describes the current display mode.
`csPage`	The number for a video page. In response to the `cscSwitchMode` control request, your control routine should switch the display to this video page. In response to the `cscGetCurMode` status request, your status routine should use this field to return the current video page. Remember that the first video page is always number 0.
`csBaseAddr`	A pointer to the base address for the video page specified in the `csPage` field. In response to both the `cscSwitchMode` control request and the `cscGetCurMode` status request, your routine should use this field to return a pointer to the base address for the video page specified in the `csPage` field.

`csReserved` Reserved for future expansion. The Display Manager currently sets this field to 0.

VDDisplayConnectInfoRec

To gather information about the capabilities of the display attached to your video device, the Display Manager makes the `cscGetConnection` status request to your video device driver. (The `cscGetConnection` status request is described on page 1-22.) Your driver must provide a status routine to respond to this request by returning information in two of the fields of a `VDDisplayConnectInfoRec` record.

```
struct VDDisplayConnectInfoRec {
   unsigned short csDisplayType;       /* type of connected
                                          display */
   unsigned short csConnectTagged;     /* reserved; currently set
                                          by Display Manager to
                                          0 */
   unsigned long  csConnectFlags;      /* support (required or
                                          optional) for the
                                          display modes for this
                                          display */
   unsigned long  csDisplayComponent;  /* reserved for future;
                                          currently set to 0 by
                                          the Display Manager */
   unsigned long  csConnectReserved;   /* reserved for future;
                                          currently set to 0 by
                                          the Display Manager */
};
```

Field descriptions

`csDisplayType` The type of display attached to your video device. If your video device controls Apple displays or displays similar to those from Apple, your status routine can return one of the following constants or the value it represents:

```
enum {
kPanelTFTConnect
        = 2,  /* fixed-in-place LCD (TFT, aka
                 "active matrix") panels */
kFixedModeCRTConnect
        = 3,  /* very limited displays */
kMultiModeCRT1Connect
        = 4,  /* 12" optional, 13" default,
                 16" required */
```

```
                kMultiModeCRT2Connect
                        = 5,  /* 12" optional, 13" required,
                                 16" default, 19" required */
                kMultiModeCRT3Connect
                        = 6,  /* 12" optional, 13" required,
                                 16" required, 19" required,
                                 21" default */
                kMultiModeCRT4Connect
                        = 7,  /* expansion to large multimode (not
                                 yet implemented) */
                kModelessConnect
                        = 8,  /* expansion to modeless model
                                 (not yet implemented) */
                kFullPageConnect
                        = 9,  /* 640x818 (to get 8bpp in 512K
                                 case) and 640x870 (nothing else
                                 supported) */
                kVGAConnect
                        = 10, /* 640x480 VGA default --
                                 nothing else required */
                kNTSCConnect
                        = 11, /* NTSC ST (default), FF, STconv,
                                 FFconv */
                kPALConnect
                        = 12, /* PAL ST (default), FF, STconv,
                                 FFconv */
                kHRConnect
                        = 13, /* 640x400 (to get 8bpp in 256K
                                 case) and 640x480 (nothing else
                                 supported) */
                kPanelFSTNConnect
                        = 14  /* fixed-in-place LCD FSTN (aka
                                 "supertwist") panels */
};
```

If your video device controls a display unlike those previously listed, your status routine can return the following constant or the value it represents:

```
enum {
kUnknownConnect = 1  /* display unlike Apple’s */
}
```

`csConnectTagged`
Reserved for future expansion. The Display Manager currently sets this field to 0.

`csConnectFlags` Flag bits indicating whether all the display modes for this display are required or optional. Your status routine should set or clear the bits represented by the following constants:

```
                enum {
                kAllModesValid
                  = 0,  /* all display modes not deleted by
                           PrimaryInit code are optional */
                kAllModesSafe
                  = 1,  /* all display modes not deleted by
                           PrimaryInit code are required;
                           if you set this bit, set the
                           kAllModesValid bit, too */
                kHasDirectConnect
                  = 3   /* for future expansion, setting this bit
                           means that your driver can talk
                           directly to the display (e.g., there is
                           a serial data link via sense lines) */
                kIsMonoDev
                  = 4   /* this display does not support color */
                kUncertainConnect
                  = 5   /* there may not be a display; Monitors
                           control panel makes the user confirm
                           some operations--like moving the menu
                           bar--when this bit is set */
};
```

`csDisplayComponent`
Reserved for future expansion. The Display Manager currently sets this field to 0.

`csConnectReserved`
Reserved for future expansion. The Display Manager currently sets this field to 0.

VDTimingInfoRec

Unless your video device's declaration ROM includes a timing directory as described in "Reporting Scan Timings" beginning on page 1-9, your driver must provide a status routine that responds to the `cscGetModeTiming` status request. Your status routine must return timing information in the fields of the `VDTimingInfoRec` record that `cscGetModeTiming` passes to your driver. (The `cscGetModeTiming` status request is described on page 1-24.)

```
struct VDTimingInfoRec {
   unsigned long  csTimingMode;     /* sResource describing
                                       display mode */
   unsigned long  csTimingReserved; /* reserved for future;
                                       currently set to 0 by the
                                       Display Manager */
   unsigned long  csTimingFormat;   /* format for timing info;
                                       currently, only the format
                                       represented by the
                                       constant kDeclROMtables is
                                       valid */
   unsigned long  csTimingData;     /* scan timings data for
                                       sResource passed in
                                       csTimingMode field */
   unsigned long  csTimingFlags;    /* flag bits indicating
                                       whether the display mode
                                       with these scan timings
                                       is optional or required */
};
```

Field descriptions

`csTimingMode` The functional sResource describing a display mode. The Display Manager uses this field to specify an sResource. In the rest of this record, your status routine should supply the timing information for this display mode.

`csTimingReserved`
Reserved for future expansion. The Display Manager currently sets this field to 0.

`csTimingFormat` Format of the information in the `csTimingData` field. Currently, only the format represented by the constant `kDeclROMtables` is valid.

```
/* use information in this record instead of
   looking in the declaration ROM for timing info;
   used for patching existing card without
   updating declaration ROM */
#define kDeclROMtables  'decl'
```

`csTimingData` Scan timings data for the sResource specified in the `csTimingMode` field. If your video device controls Apple displays or displays similar to those from Apple, your status routine can return one of the following constants or the value it represents:

```
enum {
timingApple12
   = 130,   /* 512x384 (60 Hz) 12" RGB */
timingApple12x
   = 135,   /* 560x384 (60 Hz) */
timingApple13
   = 140,   /* 640x480 (67 Hz) 13" RGB */
timingApple13x
   = 145,   /* 640x400 (67 Hz) */
timingAppleVGA
   = 150,   /* 640x480 (60 Hz) VGA */
timingApple15
   = 160,   /* 640x870 (75 Hz) full page display */
timingApple15x
   = 165,   /* 640x818 (75 Hz) full page display
               818 */
timingApple16
   = 170,   /* 832x624 (75 Hz) Apple 16" RGB */
timingAppleSVGA
   = 180,   /* 800x600 (56 Hz) SVGA */
timingApple1Ka
   = 190,   /* 1024x768 (60 Hz) VESA 1K-60Hz */
timingApple1Kb
   = 200,   /* 1024x768 (70 Hz) VESA 1K-70Hz */
timingApple19
   = 210,   /* 1024x768 (75 Hz) Apple 19" RGB */
timingApple21
   = 220,   /* 1152x870 (75 Hz) Apple 21" RGB */
timingAppleNTSC_ST
   = 230,   /* 512x384 (60 Hz, interlaced,
               nonconvolved) */
timingAppleNTSC_FF
   = 232,   /* 640x480 (60 Hz, interlaced,
               nonconvolved) */
timingAppleNTSC_STconv
   = 234,   /* 512x384 (60 Hz, interlaced,
               convolved) */
timingAppleNTSC_FFconv
   = 236,   /* 640x480 (60 Hz, interlaced,
               convolved) */
timingApplePAL_ST
   = 238,   /* 640x480 (50 Hz, interlaced,
               nonconvolved) */
timingApplePAL_FF
   = 240,   /* 768x576 (50 Hz, interlaced,
               nonconvolved) */
timingApplePAL_STconv
   = 242,   /* 640x480 (50 Hz, interlaced,
               nonconvolved) */
timingApplePAL_FFconv
   = 244    /* 768x576 (50 Hz, interlaced,
               nonconvolved) */
};
```

If your video device controls a display unlike those previously listed, your status routine can return the following constant or the value it represents:

```
enum {
timingUnknown     = 0,     /* unknown timing */
}
```

If you would like unique values assigned to displays that you manufacture, contact Macintosh Developer Technical Support.

`csTimingFlags` Flag bits indicating whether the display mode with these scan timings is optional or required. Your status routine should set or clear the bits represented by the following constants:

```
enum {
kModeValid     = 0,  /* this display mode is
                        optional */
kModeSafe      = 1,  /* this display mode is
                        required; if you set this
                        bit, you should also set
                        the kModeValid bit */
kModeDefault   = 2,  /* this display mode is
                        the default for the
                        attached display; if you
                        set this bit, you should
                        also set the kModeSafe and
                        kModeValid bits */
kShowModeNow   = 3   /* show this mode in Monitors
                        control panel; useful for
                        SVGA modes */
};
```

Control Requests

The Display Manager uses one control request, `cscSwitchMode`. When issued this control request, the Display Manager switches to the display mode specified by a `VDSwitchInfoRec` record. Your driver is responsible for setting your video device and your device control entry information to match the new display mode.

Video devices typically have functional sResources to support a variety of display modes. To support the Display Manager, the `PrimaryInit` code in the declaration ROM for your video device *should not* delete from the slot resource table any functional sResources describing display modes supported by the attached display. Instead, your device driver should enable one functional sResource (which Color QuickDraw then uses to build the `GDevice` record for your device) and you should leave the other sResources disabled.

cscSwitchMode

When the user opens the Monitors control panel and changes the display mode for the display attached to your video device, the Display Manager sends your device driver the `cscSwitchMode` control request using the `PBControlSync` call, as illustrated in the following sample function:

```
/*
The CallcscSwitchMode function demonstrates the means for using
the PBControl call to send the cscSwitchMode call to the
specified video driver whose driver refNum is passed in
deviceRefNum.  The second parameter, the VDSwitchInfoPtr is set
in the csParam field where the Display Manager aware driver will
expect this pointer.  The routine sets up the parameter block,
makes a synchronous call to the driver and returns the result.
This call must be made synchronously at System Task time
*/

OSErr CallcscSwitchMode(short deviceRefNum, VDSwitchInfoPtr
                        vdSwitchInfoPtr)
{
   ParamBlockRecpb;     /* allocate param block on stack.*/

                        /* specify the driver to make the
                        control call to */
   pb.cntrlParam.ioCRefNum = deviceRefNum;

                        /* make a cscSwitchMode control call.*/
   pb.cntrlParam.csCode = cscSwitchMode;
```

```
                            /* set csParam to the value of
                            vdSwitchInfoPtr; */

                            /* have to do some fancy typecasting to
                            get things to assign correctly */

   *(Ptr*)&pb.cntrlParam.csParam = (Ptr)vdSwitchInfoPtr;

                            /* make the synchronous call
                            and return the result */

   return (PBControlSync(&pb));
}
```

Your driver must provide a control routine to respond to this request. Summarized here is the information that the Display Manager passes and the value that your control routine must return in the fields of the `VDSwitchInfoRec` record; the `VDSwitchInfoRec` record and the values passed in its fields are described in detail on page 1-11:

→	`csMode`	depth mode to switch to
→	`csData`	the number used by your video device to identify the functional sResource describing the new display mode
→	`csPage`	number of the video page to switch to
←	`csBaseAddr`	pointer to the base address for the video page specified in the `csPage` field

As illustrated in Listing 1-1 on page 1-7, your control routine should use the Slot Manager function `SetSRsrcState`, once to disable the functional sResource describing the existing display mode and then again to enable the functional sResource specified in the `csData` field passed in the `VDSwitchInfoRec` record. Your control routine should use the Slot Manager function `SUpdateSRT` after enabling the sResource for the new display mode; using the `SUpdateSRT` function updates the device driver reference number for this sResource in the slot resource table. Your control routine must update the `dCtlSlotID` field (and—if your driver uses it—the `dCtlExtDev` field) of the device control entry for your driver. It must also update the `dCtlDevBase` field of the device control entry if disabling and enabling sResources changes the pixel map's base address in your video device's `GDevice` record.

Your control routine must also return one of the following result codes for the control request:

RESULT CODES

`noErr`	0	No error; your device driver successfully processed this control request
`controlErr`	–17	Your driver does not respond to this control request

Status Requests

The Display Manager uses three status requests to gather information from your device driver. To gather information about the current display mode, the Display Manager makes the `cscGetCurMode` status request to your driver. The Display Manager also uses this status request to determine whether your video device driver supports the Display Manager. To gather information about the display capabilities of the display attached to your video device, the Display Manager uses the `cscGetConnection` status request. Your device driver should provide status routines that process and respond to these status requests. If your video device's declaration ROM does not include a timing directory describing the scan timings for the display attached to your device, the Display Manager uses the `cscGetModeTiming` status request to gather this information from your device driver.

cscGetCurMode

To gather information about the current display mode, the Display Manager and other parts of system software, make a `cscGetCurMode` status request to your driver using `PBStatusSync`, as illustrated in the following sample function:

```
/*
The CallcscGetCurMode function demonstrates the means for using
the PBStatus call tosend the cscGetCurMode call to the specified
video driver whose driver refNum is passed in deviceRefNum. The
second parameter, the VDSwitchInfoPtr is set in the csParam field
where the Display Manager aware driver will expect this pointer.
The routine sets up the parameter block, makes a synchronous call
to the driver and returns the result.  This call must be made
synchronously at System Task time.
*/

OSErr CallcscGetCurMode(short deviceRefNum, VDSwitchInfoPtr
                        vdSwitchInfoPtr)
{
   ParamBlockRecpb;     /* allocate param block on stack. */

                        /* specify the driver to make the
                        control call to */
   pb.cntrlParam.ioCRefNum = deviceRefNum;

                        /* make a cscGetCurMode control call */
   pb.cntrlParam.csCode = cscGetCurMode;
```

```
                        /* set csParam to the value of
                        vdSwitchInfoPtr; */

                        /* have to do some fancy typecasting
                        to get things to assign correctly */
   *(Ptr*)&pb.cntrlParam.csParam = (Ptr)vdSwitchInfoPtr;

                        /* make the synchronous call
                        and return the result */
   return (PBStatusSync(&pb));
}
```

For example, the Monitors control panel uses this call to save the current display mode in case it becomes necessary to undo a change made by the user. Your driver must provide a status routine to respond to this request. Summarized here is the information that your status routine must return in the fields of the `VDSwitchInfoRec` record passed by this status request; the `VDSwitchInfoRec` record and the values you can return in its fields are described in detail on page 1-11.

←	`csMode`	depth mode of the current pixel depth
←	`csData`	The number used by your video device to identify the functional sResource describing the current display mode
←	`csPage`	number of the current video page
←	`csBaseAddr`	pointer to the base address for the video page specified in the `csPage` field

The Display Manager also uses the `cscGetCurMode` status request to determine whether your video device driver supports the Display Manager. If your driver returns the result code `statusErr`, system software assumes that your driver does not support the Display Manager. If your driver supports the Display Manager, it should return the result code `noErr` instead.

RESULT CODES

`noErr`	0	No error; your device driver supports the Display Manager and successfully processed the status request
`statusErr`	–18	Your driver does not respond to this status request

cscGetConnection

To gather information about the display capabilities of the display attached to your video device, the Display Manager makes the `cscGetConnection` status request to your driver, as illustrated in the following sample function:

```
/*
The CallcscGetConnection function demonstrates the means for using the
PBStatus call to send the cscGetConnection call to the specified video
driver whose driver refNum is passed in deviceRefNum. The second parameter,
the VDDisplayConnectInfoPtr is set in the csParam field where the Display
Manager aware driver will expect this pointer. The routine sets up the
parameter block, makes a synchronous call to the driver and returns the
result. This call must be made synchronously at System Task time
*/

OSErr CallcscGetConnection(short deviceRefNum, VDDisplayConnectInfoPtr
                        vdDisplayConnectInfoPtr)
{
   ParamBlockRecpb;     */ allocate param block on stack. */

                        /* specify the driver to make the control call to */
   pb.cntrlParam.ioCRefNum = deviceRefNum;

                        /* making a cscGetCurMode control call. */
   pb.cntrlParam.csCode = cscGetConnection;

                        /* set csParam to the value of
                        vdDisplayConnectInfoPtr; */
                        /* have to do some fancy typecasting to get things
                        to assign correctly */
   *(Ptr*)&pb.cntrlParam.csParam = (Ptr)vdDisplayConnectInfoPtr;

                        /* make the synchronous call and
                        return the result */
   return (PBStatusSync(&pb));
}
```

Your driver must provide a status routine to respond to this request. Summarized here is the information that your status routine must return in the fields of the `VDDisplayConnectInfoRec` record passed by this status call; that record and the values that your routine can return in its fields are described in detail on page 1-13.

←	`csDisplayType`	type of display
←	`csConnectFlags`	flag bits indicating whether the display modes for this display are required or optional

In addition, your status routine should return one of the following codes as a result of the status request.

RESULT CODES

`noErr`	0	No error; your device driver successfully processed the status request
`statusErr`	–18	Your driver does not respond to this status request

cscGetModeTiming

Your video device must report timing information for its attached display. If you are creating a new ROM for your device, you might want to include scan timings in a timing directory in your declaration ROM, as described in “Reporting Scan Timings” beginning on page 1-9; otherwise, your driver must return a display’s scan timings by supporting the `cscGetModeTiming` status request.

To gather scan timing information, the Display Manager makes the `cscGetModeTiming` status request to your driver, as illustrated here:

```
/*
CallcscGetModeTiming demonstrates the means for using the
PBStatus call to send the cscGetModeTiming call to the specified
video driver whose driver refNum is passed in deviceRefNum.  The
second parameter, the VDTimingInfoPtr is set in the csParam field
where the Display Manager aware driver will expect this pointer.
The routine sets up the parameter block, makes a synchronous call
to the driver and returns the result.  This call must be made
synchronously at System Task time
*/

OSErr CallcscGetModeTiming(short deviceRefNum, VDTimingInfoPtr
                        vdTimingInfoPtr)
{
   ParamBlockRecpb;     /* allocate param block on stack. */

                        /* specify the driver to make
                        the control call to */
   pb.cntrlParam.ioCRefNum = deviceRefNum;

                        /* make a cscGetCurMode control call. */
   pb.cntrlParam.csCode = cscGetModeTiming;

                        /* set csParam to the value of
                        vdTimingInfoPtr;
                        /* have to do some fancy typecasting
                        to get things to assign correctly */
   *(Ptr*)&pb.cntrlParam.csParam = (Ptr)vdTimingInfoPtr;

                        /* make the synchronous call
                        and return the result */
   return (PBStatusSync(&pb));
}
```

If your device driver routine returns the `statusErr` result code, then the Display Manager examines the declaration ROM on your video device. The Display Manager looks for the timing directory (`sVidParmDir` = 123) in the board sResource.

If your board sResource doesn't include a timing directory, your device driver must supply a status routine that processes and responds to the `cscGetModeTiming` status request. Summarized here is the information that the Display Manager passes and that your status routine must return in the fields of the `VDTimingInfoRec` record passed by this status call; the `VDTimingInfoRec` record and the values passed in its fields are described in detail on page 1-15.

→	`csTimingMode`	the sResource describing a display mode, the timing information for which should be supplied in the rest of this record
←	`csTimingFormat`	format of the information in the `csTimingData` field; currently, only the format represented by the constant `kDeclROMtables` is valid
←	`csTimingData`	scan timings for the sResource specified in the `csTimingMode` field
←	`csTimingFlags`	flag bits indicating whether the display mode with these scan timings is required or optional

RESULT CODES

`noErr`	0	No error; your device driver successfully processed the status request
`statusErr`	–18	Your driver does not respond to this status request

Summary of Device Support for the Display Manager

Constants

```
/* csMode values describing pixel depth in VDSwitchInfoRec */
enum {
   firstVidMode      = 128,   /* first depth mode, representing lowest
                                 supported pixel depth */
   secondVidMode     = 129,   /* second depth mode, representing next highest
                                 pixel depth */
   thirdVidMode      = 130,   /* third depth mode, representing next highest
                                 pixel depth */
   fourthVidMode     = 131,   /* fourth depth mode, representing next highest
                                 pixel depth */
   fifthVidMode      = 132,   /* fifth depth mode, representing next highest
                                 pixel depth */
   sixthVidMode      = 133,   /* sixth depth mode, representing next highest
                                 pixel depth */
};

/* csDisplayType values in VDDisplayConnectInfoRec */
enum {
   kUnknownConnect         = 1,  /* reserved */
   kPanelTFTConnect        = 2,  /* fixed-in-place LCD (TFT, aka
                                    "active matrix") panels */
   kFixedModeCRTConnect    = 3,  /* very limited displays */
   kMultiModeCRT1Connect   = 4,  /* 12" optional, 13" default,
                                    16" required */
   kMultiModeCRT2Connect   = 5,  /* 12" optional, 13" required,
                                    16" default, 19" required */
   kMultiModeCRT3Connect   = 6,  /* 12" optional, 13" required, 16" required,
                                    19" required, 21" default */
   kMultiModeCRT4Connect   = 7,  /* expansion to large multimode (not yet
                                    implemented) */
   kModelessConnect        = 8,  /* expansion to modeless model (not yet
                                    implemented) */
   kFullPageConnect        = 9,  /* 640x818 (to get 8bpp in 512K case) and
                                    640x870 (nothing else supported) */
   kVGAConnect             = 10, /* 640x480 VGA default -- nothing else
                                    required */
   kNTSCConnect            = 11, /* NTSC ST (default), FF, STconv, FFconv */
   kPALConnect             = 12, /* PAL ST (default), FF, STconv, FFconv */
   kHRConnect              = 13, /* 640x400 (to get 8bpp in 256K case) and
                                    640x480 (nothing else supported) */
   kPanelFSTNConnect       = 14  /* fixed-in-place LCD FSTN (aka
                                    "supertwist") panels */
};

/* csConnectFlags values in VDDisplayConnectInfoRec */
enum {
   kAllModesValid    = 0,  /* all display modes not deleted by PrimaryInit
                              code are optional */
   kAllModesSafe     = 1,  /* all display modes not deleted by PrimaryInit
                              code are required; if you set this
```

```
                           bit, set the kAllModesValid bit, too */
   kHasDirectConnect = 3   /* for future expansion, setting this bit
                           means that your driver can talk directly
                           to the display (e.g., there is a
                           serial data link via sense lines) */
   kIsMonoDev        = 4   /* this display does not support color */
   kUncertainConnect
                     = 5   /* there may not be a display; Monitors control
                           panel makes the user confirm some
                           operations--like moving the menu bar--when
                           this bit is set */
};

/* csTimingFormat value in VDTimingInfoRec */
#define kDeclROMtables  'decl'   /* use information in this record instead of
                                 looking in the declaration ROM for
                                 timing info; used for patching existing
                                 card without updating declaration ROM */

/* csTimingData values in VDTimingInfoRec */
enum {
   timingUnknown     = 0,     /* unknown timing */
   timingApple12     = 130,   /* 512x384 (60 Hz) 12" RGB */
   timingApple12x    = 135,   /* 560x384 (60 Hz) */
   timingApple13     = 140,   /* 640x480 (67 Hz) 13" RGB */
   timingApple13x    = 145,   /* 640x400 (67 Hz) */
   timingAppleVGA    = 150,   /* 640x480 (60 Hz) VGA */
   timingApple15     = 160,   /* 640x870 (75 Hz) full page display */
   timingApple15x    = 165,   /* 640x818 (75 Hz) full page display 818 */
   timingApple16     = 170,   /* 832x624 (75 Hz) 16" RGB */
   timingAppleSVGA   = 180,   /* 800x600 (56 Hz) SVGA */
   timingApple1Ka    = 190,   /* 1024x768 (60 Hz) VESA 1K-60Hz */
   timingApple1Kb    = 200,   /* 1024x768 (70 Hz) VESA 1K-70Hz */
   timingApple19     = 210,   /* 1024x768 (75 Hz) Apple 19" RGB */
   timingApple21     = 220,   /* 1152x870 (75 Hz) Apple 21" RGB */
   timingAppleNTSC_ST
                     = 230,   /* 512x384 (60 Hz, interlaced, nonconvolved) */
   timingAppleNTSC_FF
                     = 232,   /* 640x480 (60 Hz, interlaced, nonconvolved) */
   timingAppleNTSC_STconv
                     = 234,   /* 512x384 (60 Hz, interlaced, convolved) */
   timingAppleNTSC_FFconv
                     = 236,   /* 640x480 (60 Hz, interlaced, convolved) */
   timingApplePAL_ST
```

```
                    = 238,   /* 640x480 (50 Hz, interlaced, nonconvolved) */
   timingApplePAL_FF
                    = 240,   /* 768x576 (50 Hz, interlaced, nonconvolved) */
   timingApplePAL_STconv
                    = 242,   /* 640x480 (50 Hz, interlaced, nonconvolved) */
   timingApplePAL_FFconv
                    = 244    /* 768x576 (50 Hz, interlaced, nonconvolved) */
};

/* csTimingFlags values in VDTimingInfoRec */
enum {
   kModeValid     = 0,  /* this display mode is optional */
   kModeSafe      = 1,  /* this display mode is required; if you set this
                           bit, you should also set the kModeValid bit */
   kModeDefault   = 2,  /* this display mode is the default for the
                           attached display; if you set this bit, you should
                           also set the kModeSafe and kModeValid bits */
   kShowModeNow   = 3   /* show this mode in Monitors control panel; useful
                           for SVGA modes */};

/* code for Display Manager control request*/
enum {
   cscSwitchMode = 10, /* switch to another display mode */
};

/* codes for Display Manager status requests */
enum {
   cscGetCurMode     = 10, /* save the current display mode */
   cscGetConnection  = 12, /* return information about display capabilities
                              of connected display */
   cscGetModeTiming  = 13  /* return scan timings data for a display mode */
};
```

Data Structures

```
struct VDSwitchInfoRec {
   unsigned short csMode;     /* depth mode (also called video mode) */
   unsigned long  csData;     /* functional sResource for display mode */
   unsigned short csPage;     /* number for a video page */
   Ptr            csBaseAddr; /* pointer to the base address for the video
                                 page specified in the csPage field */
   unsigned long  csReserved; /* reserved for future expansion; currently
```

```
                                 set to 0 by the Display Manager */
};
typedef struct VDSwitchInfoRec VDSwitchInfoRec;
typedef VDSwitchInfoRec *VDSwitchInfoPtr;

struct VDDisplayConnectInfoRec {
   unsigned short csDisplayType;       /* type of connected display */
   unsigned short csConnectTagged;     /* reserved; currently set to 0 by the
                                          Display Manager */
   unsigned long  csConnectFlags;      /* support (required or optional) for
                                          display modes for this display */
   unsigned long  csDisplayComponent;  /* reserved for future;
                                          currently set to 0 by the
                                          Display Manager */
   unsigned long  csConnectReserved;   /* reserved for future;
                                          currently set to 0 by the
                                          Display Manager*/
};
typedef struct VDDisplayConnectInfoRec VDDisplayConnectInfoRec;
typedef VDDisplayConnectInfoRec *VDDisplayConnectInfoPtr;

struct VDTimingInfoRec {
   unsigned long  csTimingMode;     /* sResource describing display mode */
   unsigned long  csTimingReserved; /* reserved for future expansion;
                                       currently set to 0 by the
                                       Display Manager */
   unsigned long  csTimingFormat;   /* format for timing info; currently,
                                       only the format represented by the
                                       constant kDeclROMtables is valid */
   unsigned long  csTimingData;     /* scan timings for the sResource
                                       passed in the csTimingMode field */
   unsigned long  csTimingFlags;    /* flag bits indicating whether the
                                       display mode with these scan timings
                                       is optional or required */
};
typedef struct VDTimingInfoRec VDTimingInfoRec;
typedef VDTimingInfoRec *VDTimingInfoPtr;
```

Result Codes

`noErr`	0	No error; your device driver successfully processed the request
`controlErr`	–17	Your driver does not respond to this control request
`statusErr`	–18	Your driver does not respond to this status request

Glossary

board sResource In the declaration ROM of a device, such as an expansion card, a unique data structure that describes the device so that the Slot Manager can identify it. A device can have only one board sResource. The entries in this data structure include the device's identification number, vendor information, board flags, initialization code, and so on.

default display mode A **required display mode** that appears the first time a user turns on a display.

depth mode A constant (or its value from an enumerated list) that represents a pixel depth supported by a video device. Also called *video mode*.

display An output device—such as a video monitor or flat-panel display—on which an application can show interactive visual information—such as text for a document or data for a spreadsheet. Also called a *monitor*.

Display Manager A set of system software routines that allow users to dynamically change the arrangement and display modes of the displays attached to their computers. Users can move displays, add or remove displays, switch multiple-scan displays to show more or less of the desktop, and move the menu bar from one display to another—without restarting their computers.

display mode A combination of several interrelated capabilities for a display that can be altered by the Display Manager without restarting the computer. These capabilities include the refresh rate, number of onscreen pixels, scan timings, and pixel depth.

extended sense code A 6-bit binary identification code that is read by the primary initialization code of a video device from the sense lines of an attached display. Compared with the original **sense codes,** extended sense codes allow a greater number of display types to identify themselves to video devices. Compare **type 6 extended sense code**.

functional sResource A data structure in a device's declaration ROM that describes a specific function of the device—for example, controlling a display at a certain display mode.

horizontal scan timing The time required by a display to draw one horizontal line, including retrace.

monitor See **display.**

multiple-scan display A display capable of operating at multiple horizontal and vertical scan timings.

optional display mode A display mode that is not required by the Display Manager for a particular type of display (such as a 19-inch video monitor) but that can be supported by the display anyway. A list of optional display modes appears when the user holds the Option key and clicks the Options button in the Monitors control panel. Compare **required display mode.**

required display mode A display mode that a particular type of display (such as a 19-inch video monitor) is required by the Display Manager to support. Compare **optional display mode.**

scan timing The time required by a display to draw either one vertical or horizontal line, including retrace.

screen resolution The number of pixels which a display can render. Screen resolution is specified as pixels in the *x* and *y* directions—for example, 680 by 400 pixels.

sense code An identification code that is read by the primary initialization code of a video device from the sense lines of an attached display. The video device uses the sense code to determine how to control the display.

type 6 extended sense code A 6-bit binary identification code that is read by the primary initialization code of a video device from the sense lines of an attached multiple-scan display. The type 6 extended sense code causes video devices that don't support the Display Manager to nevertheless support multiple-scan displays as 640-by-480 dpi displays.

vertical scan timing The time required by a display to draw one vertical line, including retrace.

video device A piece of hardware, such as a plug-in video card or a built-in video interface, that controls a display.

video device driver The program that controls a video device.

video mode See **depth mode.**

video timing See **scan timing.**

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe Illustrator™ and Adobe Photoshop™.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.